# MATHEMATICA ДЛЯ НЕМАТЕМАТИКА

Н.А. Вавилов, В.Г. Халин, А.В. Юрков

M

M A M

M A T A M

M A T H T A M

M A T H E H T A M

M A T H E M E H T A M

M A T H E M A M E H T A M

M A T H E M A T A M E H T A M

M A T H E M A T I T A M E H T A M

M A T H E M A T I C I T A M E H T A M

M A T H E M A T I C A C I T A M E H T A M

Москва
Издательство МЦНМО
2021

УДК 330.4
ББК 65в6
В13

При поддержке Благотворительного
фонда Владимира Потанина

*Рецензенты:*
**Емельянов А.А.**, доктор экономических наук, профессор,
Смоленский филиал Московского энергетического института (МЭИ)
**Гердт В.П.**, доктор физико-математических наук, профессор,
Объединенный институт ядерных исследований, Дубна
**Васильев Н.Н.**, кандидат физико-математических наук, старший
научный сотрудник, Санкт-Петербургское отделение Математического
института им. В.А. Стеклова Российской академии наук

Вавилов Н.А., Халин В.Г., Юрков А.В.
`Mathematica` для нематематика: учебное пособие для вузов
Электронное издание
М.: МЦНМО, 2021
483 с.
ISBN 978-5-4439-3584-3

Настоящий учебник посвящен системе `Mathematica` — прикладному пакету компьютерной алгебры, при помощи которого можно решать любые задачи, в которых в той или иной форме встречается математика. Учебник возник из желания соавторов материализовать разделяемое ими убеждение, что нельзя учить математике, натаскивая на рутинных операциях, которые студенты в своей будущей жизни никогда не применят. Современные математические пакеты — а `Mathematica` среди них безусловно выдающийся — лучше многих решат уравнения и выполнят вычисления (в умелых руках). Научить будущего исследователя-нематематика применять сообразно решаемой задаче этот доступный даже школьнику инструмент — цель, к которой, создавая учебник, стремились авторы. Эта цель обрела реальность благодаря поддержке Благотворительного фонда Владимира Потанина, реализующего масштабные проекты в сфере образования и культуры.

Книга издается в авторской редакции

Оригинал макет подготовлен с использованием системы макрорасширений $\mathcal{AMS}$-TEX ver. 2.2 стандартного TEX'a ver. 3.14159625
Лицензия `Wolfram Mathematica` 4081–2263

Издательство Московского центра
непрерывного математического образования
119002, Москва, Большой Власьевский пер., 11,
тел. (499) 241–08–04. `http://www.mccme.ru`



**ISBN 978-5-4439-3584-3**

Computers are not intelligent. They only think they are. — Компьютеры не могут мыслить; они лишь думают, что могут мыслить.

Рене Декарт

The purpose of computing is insight, not numbers! — Целью вычисления является понимание, а не числа!

Richard Hamming

If you can't learn to do it well, learn to enjoy doing it badly! — Если вы не можете научиться делать это хорошо, научитесь получать удовольствие от того, что вы делаете это как умеете!

The Tao of Real Programming

# СОДЕРЖАНИЕ

Предыстория и благодарности ..... 7

Введение ..... 8

Модуль 1. Первое знакомство с системой Mathematica ..... 9

Глава 1. Что такое компьютерная алгебра? ..... 10
§ 1. Математика и компьютеры ..... 12
§ 2. Компьютерная алгебра ..... 14
§ 3. Влияние компьютеров на математическое мышление 16
§ 4. Возможности систем компьютерной алгебры ..... 18
§ 5. Об “ошибках” систем компьютерной алгебры ..... 20

Глава 2. Что такое Mathematica ..... 24
§ 1. Достоинства и особенности Mathematica ..... 24
§ 2. Структура системы Mathematica ..... 28
§ 3. Главное меню Mathematica ..... 31
§ 4. Система помощи Mathematica ..... 33
§ 5. Палитры ..... 36
§ 6. Сессии и вычисления ..... 37
§ 7. Блокноты и ячейки ..... 38
§ 8. Общие советы и типичные ошибки ..... 40

Глава 3. Практическое введение в систему Mathematica ... 44
§ 1. Арифметика ..... 47
§ 2. Многочлены и рациональные дроби ..... 52
§ 3. Алгебраические уравнения ..... 59
§ 4. Системы уравнений и неравенств ..... 69
§ 5. Элементарные функции ..... 77
§ 6. Графики функций ..... 85
§ 7. Суммы, произведения, пределы ..... 108
§ 8. Производные ..... 117
§ 9. Интегралы ..... 123
§ 10. Вектора и матрицы ..... 130
§ 11. Линейная алгебра ..... 138

Модуль 2. Основы синтаксиса ..... 147

Глава 4. Объекты и выражения ..... 148
§ 1. Выражения: FullForm, Head, Part, Level, Length, Depth . 149
§ 2. Что такое квадратный трехчлен? ..... 152
§ 3. Выделение уровней ..... 154
§ 4. Имена объектов ..... 156
§ 5. Группировка и скобки в математике ..... 158
§ 6. Группировка и скобки в Mathematica ..... 160
§ 7. Числовые домены ..... 162
§ 8. Универсальные инструменты:
= Simplify, FullSimplify, Refine ..... 165
§ 9. Целые и рациональные числа ..... 166
§ 10. Запись вещественного числа ..... 169
§ 11. Константы ..... 171
§ 12. Непрерывные дроби и рациональные приближения . 173
§ 13. Комплексные числа ..... 175
§ 14. Генерация случайных чисел ..... 176
§ 15. Бесконечность и неопределенность ..... 179

Глава 5. Переменные и их значения ..... 181

§ 1. Математика как неточная наука ..... 181
§ 2. Переменные и их значения ..... 183
§ 3. Многочлены ..... 185
§ 4. Немедленное и отложенное присваивание:
= Set versus := SetDelayed ..... 188
§ 5. Секунды, такты и шаги ..... 190
§ 6. Модификация значения переменной ..... 193
§ 7. Немедленная и отложенная подстановка:
-> Rule versus :> RuleDelayed ..... 195
§ 8. Просто замены и сугубые замены:
Replace, /. ReplaceAll, //. ReplaceRepeated ..... 199
§ 9. Чистка: Unset, Clear, ClearAll, Remove ..... 201
§ 10. Создание локальных и уникальных переменных .... 203
§ 11. Равенство и тождество: == Equal versus === Same .... 207
§ 12. Решение алгебраических уравнений ..... 209

Глава 6. Функции ..... 212

§ 1. Функциональная и операторная запись функций .... 212
§ 2. Специфика функций компьютерной алгебры ..... 216
§ 3. Основные классы функций языка Mathematica ..... 217
§ 4. Функции нескольких аргументов ..... 223
§ 5. Аргументы функций ..... 227

§ 6. Алгебраические операции ..... 230
§ 7. Арифметические операции ..... 233
§ 8. Приоритет!!! ..... 236
§ 9. Итераторы ..... 238
§ 10. Делимость целых чисел ..... 240
§ 11. Сравнение вещественных чисел ..... 242
§ 12. Предикаты ..... 242
§ 13. Булевы функции и кванторы ..... 246
§ 14. Реляционные операторы ..... 247
§ 15. Числовые функции: экспонента и логарифм ..... 248
§ 16. Тригонометрические и гиперболические функции ... 250

Модуль 3. Задачи ..... 252

Глава 7. Арифметика и теория чисел ..... 263

§ 1. Целые числа ..... 263
§ 2. Рациональные числа ..... 277
§ 3. Вещественные числа ..... 288
§ 4. Комплексные числа ..... 307
§ 5. Простые числа ..... 318
§ 6. Модулярная арифметика ..... 339

Глава 8. Комбинаторика и дискретная математика ..... 352

§ 1. Комбинаторика ..... 353
§ 2. Списки и последовательности ..... 369
§ 3. Функции ..... 399
§ 4. Перестановки ..... 421
§ 5. Циклы ..... 440

Глава 9. Многочлены и матрицы ..... 458

§ 1. Многочлены ..... 459
§ 2. Запись матриц ..... 465
§ 5. Определители ..... 476

Литература ..... 483

## Предыстория и благодарности

Учебник был задуман в начале “нулевых”, когда авторы читали лекции по курсам “Математические пакеты” и “Математика и компьютеры” на экономическом факультете СПбГУ и накапливали материалы для поддержки занятий в цикле учебных пособий, имеющих ограниченное распространение. С той поры система `Mathematica` получила значительное развитие, включая онлайн версию, и круг ее потенциальных пользователей существенно расширился, благодаря Интернету. Среди прочего это укрепляет нашу уверенность, что великолепный инструмент, коим является система `Mathematica`, может и должен изучаться и применяться всеми исследователями, использующими математический аппарат в своей деятельности, и нематематиками в том числе.

Идея обобщить накопленный опыт и подготовить книгу для широкого круга читателей, не имеющих специального математического образования, обрела реальность в 2018 году, когда наша заявка на учебник “`Mathematica` для нематематика” получила грант Благоворительного фонда Владимира Потанина, реализующего масштабные проекты в сфере образования и культуры.

Желание облечь разработанные ранее учебные материалы в форму полноценного учебника потребовало решения и чисто технических вопросов, обусловленных необходимостью подключения к системе верстки достаточно раритетных в настоящее время макрорасширений $\mathcal{AMS}$-TEX стандартного TEX’а, с использованием которых написаны исходные тексты прежних лет. В связи с этим мы не можем не отметить с признательностью внимание и помощь в преодолении порой почти недокументированных ретропроблем, оказанные авторам коллегой по СПбГУ и соавтором ряда совместных работ Андреем Алексеевичем Семёновым.

Работая над настоящим учебником, мы не можем не вспомнить с благодарностью и ту поддержку, которую имели в прежние годы, благодаря соучастию в проектах, непосредственно посвященных развитию теоретических аспектов компьютерной алгебры: РФФИ 00-10-00441, РФФИ 03-01-00349, Минвуз 2003.10.3.03.Д, Минвуз 2004.10.1.03.Д. Кроме того, наша работа была поддержана грантами INTAS 00-01-00441 и 03-51-3251 и несколькими другими проектами Министерства образования и науки Российской Федерации, Российского научного фонда и Российской Академии наук.

# ВВЕДЕНИЕ

Бывает, что человек обладает знанием, но не умеет пользоваться им. Бывает и так, что владеющий искусством сам не знает его секрета. В царстве Вэй жил человек, умевший хорошо считать. Перед смертью он поведал секрет своего искусства сыну. Сын запомнил слова, но не знал, как применить их. Какой-то человек стал его расспрашивать об этом, и он передал ему те слова. И тот человек научился считать не хуже, чем это делал отец.

Ле-цзы, Гл. VIII. Рассказы о совпадениях

Человек может пользоваться только тем, чем он умеет пользоваться.
Человек может использовать только то, что он знает, как использовать

Идрис Шах, ‘Сказки дервишей’

Was man nich versteht, besitzt man nicht. — Чего человек не понимает, то ему не принадлежит.[1]

Johann Wolfgang von Goethe, Maximen und Reflexionen

Nessuna certezza è dove non si pò applicare la matematica. — Там, где нельзя применить `Mathematica`, не может быть никакой достоверности.

Leonardo da Vinci

They spell it `da Vinci` and pronounce it `da Vinchy`. Foreigners always spell better than they pronounce. — Они пишут `da Vinci`, а произносят `да Винчи`. Иностранцы всегда пишут гораздо лучше, чем говорят.

Mark Twain

[1]Канонический русский перевод этого афоризма таков: “Чем человек не владеет, тем он не обладает” — однако при этом вводится игра слов, которой нет в оригинале. Между тем, господин тайный советник абсолютно категоричен: “Чем человек не умеет пользоваться, **того у него нет**” — was du ererbt von deinen Vätern hast, erwirb es, um es zu besitzen.

# МОДУЛЬ 1. Первое знакомство с системой Mathematica

Умножая свои знания, умножаешь чью-то скорбь.

Соломон

— Ну скажи, почему этот “мерседес” реальный? — спросил Володин.
Несколько секунд Мария мучительно думал.
— Потому что он из железа сделан, — сказал он, — вот почему. А это железо можно подойти и потрогать.
— То есть ты хочешь сказать, что реальным его делает некая субстанция, из которой он состоит?
Мария задумался.
— В общем, да, — сказал он.
— Вот поэтому мы Аристотеля и рисуем. Потому что до него никакой субстанции не было, — сказал Володин.
— А что же было?
— Был главный небесный автомобиль, — сказал Володин, — по сравнению с которым твой шестисотый “мерседес” — говно полное. Этот небесный автомобиль был абсолютно совершенным. И все понятия и образы, относящиеся к автомобильности, содержались в нем одном. А так называемые реальные автомобили, которые ездили по дорогам Древней Греции, считались просто его несовершенными тенями. Как бы проекциями. Понял?

Виктор Пелевин, Чапаев и Пустота

Я также полагаю, что термин **интеллект** следует употреблять исключительно в связи с пониманием. Некоторые же теоретики искусственного интеллекта берутся утверждать, что их робот вполне может обладать **интеллектом** не испытывая при этом никакой необходимости в действительном **понимании** чего-либо. На мой взгляд словосочетание **интеллект без понимания** есть лишь результат неверного употребления терминов. Следует, впрочем, отметить, что иногда что-то вроде частичного моделирования подлинного интеллекта без какого бы то ни было реального понимания оказывается до определенной степени возможным. В самом деле, не так уж редко встречаются *человеческие* существа, способные на некоторое время одурачить нас демонстрацией некоторого понимания, хотя, как в конце концов выясняется, оно им в принципе не свойственно!

Роджер Пенроуз, Тени разума. Гл.1. Сознание и вычисление

Предположим, что Вы получаете анонимное письмо, содержащее лишь листок бумаги с четырьмя строчками поэтического текста. Например, это может быть стихотворение Уильяма Блейка, “Eternity”:

He who binds to himself a joy
Does the winged life destroy;
But he who kisses the joy as it flies
Lives in eternity’s sun rise.

Предположим, что кому-то удалось воссоздать всю последовательность возбуждений нейронных цепей в мозгу поэта (в тот момент,

> когда он написал эти строки), в Вашем мозгу (в тот момент, когда Вы прочли эти строки) и даже в мозгу отправителя этого письма! Предположим, повторяю, что кому-то удалось понять всю эту фантастическую и прекрасную сложность биохимических реакций, обеспечивающих соответствующие возбуждения в нейронных цепях, и описать (а значит постичь) всю совокупность физических и химических актов, из которых состоят эти биохимические реакции! Какое невероятное и восхитительное `знание` можно получить при таком анализе! Какое количество Нобелевских премий можно было бы получить сразу! Но ... давайте также поймем, что все это знание не дает *ничего* для понимания смысла стихов Блейка, по крайней мере в том значении, который мы обычно вкладываем в слово `понимание`, обращаясь к поэзии. Понимание поэзии Блейка следует искать на уровне того языка, на котором они были созданы, и с учетом психологии автора и читателя. Картина возбуждения нейронных сетей никак не может помочь при этом, т.е. не дает `понимания` знания.
>
> Роальд Хоффманн. The same and not the same

# Глава I. ЧТО ТАКОЕ КОМПЬЮТЕРНАЯ АЛГЕБРА?

Компьютерная алгебра является одной из самых мифологизированных областей. Не только большинство пользователей, но и многие профессиональные математики и программисты не имеют представления о реальной силе, возможностях и специфике имеющихся систем, не говоря уже о ближайших перспективах этой области. Мы постараемся развеять некоторые из этих мифов.

Для нас не представляет сомнения, что:

• При помощи систем компьютерной алгебры уже сегодня возможно проводить все обычные в математике и ее приложениях вычисления. Все импликации этого факта не только не осознаны, но даже не начинали еще всерьез рассматриваться.

• Основные системы компьютерной алгебры являются в первую очередь языками программирования сверхвысокого уровня, приближающимися по своей выразительной силе к живому языку, и их следует изучать именно как языки, а не как обычные компьютерные приложения.

• Математикам свойственно недооценивать то, в какой степени развитие математики зависит от внешних обстоятельств, в первую очередь от доступных вычислительных средств. Развитие компьютерной алгебры уже сегодня оказывает радикальное воздействие на исследования во многих областях чистой математики (таких, как теория групп, комбинаторика, теория чисел, коммутативная алгебра, алгебраическая геометрия и т.д.). В самое ближайшее время это влияние распространится на **всю** математику и приведет к кардинальному пересмотру основных направлений исследований, переоценке всех ценностей и полному изменению стиля работы математиков.

• Бешеное сопротивление, которое вызывает развитие компьютерной алгебры среди методистов и многих преподавателей математики, связано с тем, что ДАЛЬНЕЙШЕЕ РАЗВИТИЕ ЭТИХ СИСТЕМ уже в ближайшие 10–15 лет ПРИВЕДЕТ К ПОЛНОМУ ОБЕСЦЕНИВАНИЮ ВСЕХ ТРАДИЦИОННЫХ ВЫЧИСЛИТЕЛЬНЫХ НАВЫКОВ и необходимости полного пересмотра преподавания математики на школьном и университетском уровне.

• Бешеное сопротивление, которое вызывает развитие компьютерной алгебры среди многих представителей `Computer Science`, связано с тем, что эти системы полностью обесценивают и подавляющую часть традиционных программистских навыков. ПРИ ПОМОЩИ ЭТИХ СИСТЕМ ЛЮБОЙ ГРАМОТНЫЙ ЛЮБИТЕЛЬ МОЖЕТ ЗА НЕСКОЛЬКО МИНУТ НАПИСАТЬ ПРОГРАММУ, АНАЛОГ КОТОРОЙ НА `Fortran` ИЛИ `C` ПОТРЕБОВАЛ БЫ НЕСКОЛЬКИХ ДНЕЙ РАБОТЫ ПРОФЕССИОНАЛЬНОГО ПРОГРАММИСТА.

Мы думаем, что имеется еще одно чрезвычайно существенное обстоятельство, объясняющее неистовое эмоциональное неприятие систем компьютерной алгебры и побуждающее многих игнорировать их возможности — и даже само их существование. Дело в том, что эти системы непринужденно решают задачи, которые, как традиционно считалось, являются чисто человеческими и требуют `интеллекта` и `мышления`, задачи, на которых основано все традиционное преподавание, задачи, представляющие серьезные трудности для большинства *человеческих* существ! Опыт общения с этими системами побуждает отбросить шоры европейской рационалистической философии и заново обдумать **все**, что связано с `интеллектом` и `мышлением`, полностью разделив те уровни, на которых происходит `вычисление` и те, на которых происходит `понимание`, те, на которых функционирует `интеллект` (=`мышление`?) и те, на которых функционирует `сознание`.

Начиная с 1950-х годов чрезвычайно популярна дискуссия на тему “может ли компьютер мыслить?” Усилия физиков были направлены на то, чтобы доказать, что компьютер *может* мыслить, в то время как аргументы лириков каждый раз основывались на таком переопределении понятия `мышления`, которое позволяло игнорировать каждый новый успех физиков. Исследования в области компьютерной алгебры вплотную подвели нас к такой точке, где никакое дальнейшее переопределение понятия `мышления` не представляется возможным и мы вынуждены *констатировать*, что компьютер может мыслить. Тем самым, подлинный вопрос искусственного интеллекта должен теперь ставиться так: “может ли компьютер *понять*, что он может *мыслить*?” — или, по Декарту, `cogito cogitare`.

## § 1. Математика и компьютеры

Anyone who cannot cope with mathematics is not fully human. At best he is a tolerable subhuman who has learned to wear shoes, bathe and not make messes in the house.

Lazarus Long, “Time Enough for Love”

Проблема N 1 кибернетики: Каким местом человек думает?
Проблема N 2 кибернетики: Как он это этим местом делает?

А.Соловьев, ‘Ишкушштвенный интеллект’

Прежде чем переходить к обсуждению собственно системы `Mathematica`, мы сделаем несколько общих замечаний о роли компьютера и, в особенности, о роли символьных вычислений в научном исследовании и преподавании математики как на школьном, так и на университетском уровне. Нам кажется, что этот вопрос является сегодня не просто важной, а **центральной** проблемой для всех, кто *всерьез* задумывается над тем, чему и как нужно учить школьников и студентов технических, экономических и естественнонаучных (а может быть и гуманитарных!!!) специальностей в курсах математики и информатики.

Сегодняшнее построение курса математики в общеобразовательной школе отягощено двухтысячелетней традицией и, в целом, не находится более даже на уровне потребностей XVI века!!! Чуть лучше обстоит дело в нескольких специализированных физико-математических школах, но и здесь, с нашей точки зрения, необходим дальнейший *радикальный* пересмотр всей программы, в сторону ее углубления и модернизации. Курс же информатики в значительной степени унаследовал свойственное 50-м и началу 60-х годов — и совершенно абсурдное с сегодняшней точки зрения!!! — отождествление любого серьезного использования компьютера с традиционным программированием.

Высказанные в предыдущем абзаце утверждения могут показаться чересчур драматичными, и нуждаются в пояснении. С нашей точки зрения, существующий сегодня школьный курс математики ориентирован, прежде всего, на выработку вычислительных навыков и *механического* использования небольшого числа стандартных алгоритмов — кстати, в большинстве своем чрезвычайно неэффективных с вычислительной точки зрения! Не следует думать, конечно, что это является чисто Российской проблемой; насколько мы можем судить, во всех странах Западной Европы — не говоря уже про США!! — преподавание математики в школе в целом *либо* поставлено еще значительно хуже, чем в России, *либо* страдает крайним формализмом (как во Франции). Не следует думать также, что это является исключительно проблемой средней школы — почти в такой же степени архаично и ни в малейшей степени не отвечает сегодняшним потребностям и преподавание математики в университетах и технических ВУЗах. Традиционные вузовские курсы — в первую очередь уже *абсолютно застойные* курсы математического анализа, но в значительной степени также и

*архаичные* курсы линейной алгебры, дифференциальных уравнений, теории вероятностей и дискретной математики — также направлены почти исключительно на механическое овладение *рудиментарными* вычислительными навыками, без всякого понимания подлинной структуры предмета, его современного состояния и более широкого контекста.

В прошлом подобные вычислительные навыки имели несомненную ценность, но сегодня необходимость массового обучения им более чем сомнительна. Многие разделы математики и вычислительные приемы, которые изучаются сегодня в школе, по своей функциональности в современном мире подобны добыванию огня при помощи трения. Мы не говорим, что это полностью лишает их ценности, вызывает сомнение лишь необходимость систематического упражнения в их применении. Мы провели бы границу следующим образом: появление калькуляторов не отменяет необходимость в заучивании таблицы умножения. Однако появление калькуляторов делает *абсолютно* бессмысленным СИСТЕМАТИЧЕСКОЕ УПРАЖНЕНИЕ в операциях над многозначными числами — никому из сегодняшних школьников в нормальных условиях не придется выполнять подобные операции вручную, просто потому, что любое, самое примитивное вычислительное устройство делает это БЫСТРЕЕ, ЭФФЕКТИВНЕЕ И НАДЕЖНЕЕ.

**Комментарий.** И сегодня в Японии придается чрезвычайно большое значение развитию у детей МЛАДШЕГО ВОЗРАСТА навыков устного счета, вплоть до заучивания наизусть таблицы умножения $100 \times 100$, что в сочетании с правильным алгоритмом умножения многозначных чисел (разбиение на блоки и замена умножений сложениями и сдвигами) позволяет им легко умножать в уме восьмизначные числа. Однако никто из японских эдукационистов не говорит, что это делается с какой-то практической целью. Единственная цель этих упражнений — тренировка памяти и лучшее кровоснабжение мозга. Американские эдукационисты придерживаются противоположной теории, в большинстве американских школ не учат даже таблицу умножения $10 \times 10$, а для улучшения мозгового кровообращения используются уроки физкультуры. Из личного опыта мы знаем, что рядовой американский студент не в состоянии перемножить без калькулятора 7 на 8. Однако нам трудно решить для себя, как следует относиться к этому факту.

## § 2. Компьютерная алгебра

Одной из первых областей применения компьютерной алгебры была небесная механика. Классическим примером, упоминаемым во всех обзорах, служит пересчет Депритом, Хенрардом и Ромом результатов Делоне. Введение новой техники позволило им пересчитать гамильтониан в теории Луны. Делоне потратил 20 лет, выполняя вычисления вручную. Деприту, Хенрарду и Рому понадобилось всего лишь 20 часов работы небольшой ЭВМ. Следует отдать должное аккуратности Делоне, в работах которого, опубликованных в 1867 году и содержащих результаты вплоть до 9-го порядка по малым параметрам, все было вычислено безошибочно, за исключением одного сложения в 7-м порядке. Это вычисление лунной орбиты требует тщательного рассмотрения многих физических эффектов, таких как несферичность Земли, наклон эклиптики и влияние Солнца. Гамильтониан, описывающий рассматриваемую систему, занимает несколько страниц; к каждому члену нужно применять до 600 преобразований.

Я.А.ван Хюльзен, Ж.Калме[2]

Появление современных систем **символьных вычислений** или, как часто говорят, **компьютерной алгебры**, сокращенно **CA** (`Computer Algebra`), ставит — с еще большей остротой — тот же самый вопрос в применении к большинству разделов школьного и вузовского курса математики. Название `компьютерная алгебра` хорошо отражает суть дела, но не слишком удачно с точки зрения маркетинга и рекламы. Оно создает у несведущего человека впечатление, что речь идет исключительно о проведении *алгебраических* вычислений на компьютере. На самом деле оно указывает лишь на то, что большинство используемых в этих системах *алгоритмов* основано на использовании методов современной алгебры и теории чисел, предметная же область этих систем гораздо шире, при помощи них можно анализировать все в области нашего опыта и умозрения, что поддается точному определению. Ядром большинства современных систем компьютерной алгебры действительно являются следующие три блока вычислений:

• численные вычисления неограниченной точности (так называемые *безошибочные вычисления*) с целыми, рациональными, вещественными и комплексными числами;

• собственно алгебраические (алиас *символьные*) вычисления с многочленами, перестановками, векторами, матрицами etc.;

• логические и структурные манипуляции с высказываниями, последовательностями, списками, множествами etc.

---

[2] Я.А.ван Хюльзен, Ж.Калме, Применения компьютерной алгебры. — В кн. Компьютерная алгебра, М., Мир, 1986, с.308–325. Речь в этом отрывке идет о работе A.Deprit, J.Henrard, A.Rom, Lunar ephemeris: Delaunay's theory revisited. — Science, 1970, vol.168, p.1569–1570. Сегодня это вычисление выполняется `Mathematica` за несколько минут.

Однако в действительности *кроме этого* при помощи систем компьютерной алгебры можно проводить **все** обычные в математике и ее приложениях аналитические вычисления:

• численное и символьное дифференцирование, интегрирование, решение дифференциальных уравнений и уравнений в частных производных и тому подобное;

• доказательство несложных теорем, включая доказательство **всех** теорем из школьного курса геометрии;

• логическую обработку и преобразование текстов ЛЮБОЙ ПРИРОДЫ: текстов на естественных языках, шифров, музыкальных текстов, etc.;

• анализ и редактирование изображений, все геометрические и графические построения — вплоть до создания мультфильмов;

• статистическую обработку численных, текстовых, логических и графических данных;

• создание баз данных, электронных энциклопедий, интерактивных справочников, учебников и задачников по любой области знания;

• математическое моделирование любых процессов, компьютерный эксперимент;

• разработку, тестирование и анализ алгоритмов, компьютерных программ и прикладных пакетов;

• и многое другое.

Показательно в этой связи, что в США и Западной Европе такие системы компьютерной алгебры общего назначения как `Maple` или `Mathematica` имеют даже большее распространение среди физиков, химиков, биологов, инженеров, экономистов и других представителей естественных и математических наук, чем среди собственно математиков. Кстати, профессиональные математики часто предпочитают высоко эффективные сугубо специализированные системы типа `Cayley`, `GAP`, `MAGMA`, `Singular`, `Lie`, `Chevie`, `CoCoA`, `Fermat`, `PARI`, `DELiA`, `GRep`, `Magnus`, `Macaulay`, `Schur`, `FELIX` и другие, направленные собственно на алгебраические или теоретико-числовые вычисления, численные или матричные вычисления неограниченной точности, решение дифференциальных уравнений и все такое, без всякой графики, меню, палитр, мультфильмов и прочих глупостей.

Появление систем компьютерной алгебры должно оказать и громадное влияние на преподавание курса информатики. Уже давно стало ясно, что для подавляющего большинства пользователей `компьютерная грамотность` состоит отнюдь не в навыках программирования, а в умении эффективно ориентироваться в существующих системах математического обеспечения и квалифицировано их использовать. Лучшие же системы символьных вычислений вообще *полностью* упраздняют необходимость в традиционном программировании в стиле **ПошелНа**: `If ... Then ...` и `GoTo ...` Дело в том, что они являются не компилирующими, а интерпретирующими и

сами превращают определение объекта, написанное на языке, по сути достаточно близком к реальному математическому английскому языку, но с более жесткими правилами синтаксиса (расстановка скобок, знаков препинания, прописные и строчные буквы и пр.), в программу для его вычисления.

## § 3. Влияние компьютеров на математическое мышление

> Science is what we understand well enough to explain to a computer. Art is everything else we do.
>
> Donald Knuth

> The great advances in science usually result from new tools rather than from new doctrines.
>
> Freeman Dyson

> Mathematics is much less formally complete and precise than computer programs.
>
> William P. Thurston

> To be a scholar of mathematics you must be born with talent, insight, concentration, taste, luck, drive and the ability to visualize and guess. — Чтобы стать профессиональным математиком, нужно родиться с талантом, проницательностью, сосредоточенностью, стилем, удачей, настойчивостью, внутренним зрением и воображением.
>
> Paul R. Halmos

С нашей точки зрения абсолютно не поняты теоретические основы использования систем компьютерной алгебры в преподавании и исследованиях за пределами математики и теоретической физики, а также те изменения, которые эти системы предполагают в собственно математическом мышлении пользователя. Даже многие авторы учебников по *компьютерной математике*, не говоря уже о широких кругах пользователей-неспециалистов не отличают настоящие системы компьютерной алгебры, такие как `Maple` или `Mathematica`, ни от специализированных пакетов *численных* вычислений, таких как `MatLab` или `Statistica`, ни даже от чисто учебных программ, графических калькуляторов или систем для работы с текстом, содержащим формулы, таких как `Mathcad`.

В большинстве случаев принятое сегодня изложение математики — не только в школе, но и на математических факультетах университетов — возникло в докомпьютерную эпоху и совершенно неудовлетворительно с алгоритмической точки зрения. Между тем, во многих случаях даже небольшое изменений определений делает их значительно более пригодными для практических вычислений. Скажем, не только в школьном, но и в университетском курсе, степень определяется рекурсивно как $x^n = x^{n-1}x$. Между тем, самое незначительное изменение этого определения может драматически увеличить его применимость. Например, можно определить степень указанной выше формулой в случае, когда $n$ нечетно и формулой $x^n = \left(x^{n/2}\right)^2$

в случае, когда $n$ четно. Для объектов, умножение которых занимает заметное время (матрицы или многочлены высокой степени), вычисление, скажем, 1000-й степени с использованием второго определения может быть в сотни раз быстрее, чем с помощью первого. Заметим, что речь не идет о наиболее эффективном профессиональном алгоритме для вычисления степени — в большинстве случаев достаточно просто задумываться над подобного рода нюансами.

Еще более интересные феномены связаны с умножением матриц. Обычный алгоритм умножения матриц размера $2 \times 2$ требует 8 умножений коэффициентов. Ф.Штрассен предложил алгоритм требующий лишь 7 умножений. Для больших степеней этот алгоритм дает еще более драматическую редукцию числа умножений. Примерно так же можно упростить и гауссово исключение. Важность этого открытия трудно переоценить — некоторые специалисты считают его одним из 10 наиболее важных математических открытий XX века. При решении серьезных вычислительных задач (например, при приближенном решении дифференциальных уравнений движения, в задачах оптимизации), где приходится умножать и обращать тысячи матриц порядка 1000 и более, этот алгоритм дает громадную экономию времени, делающую возможными немыслимые ранее вычисления (NASA использует этот алгоритм при управлении в реальном времени космическими аппаратами).

Другой столь же впечатляющий пример, связанный с умножением матриц, где четко видна граница между возможным и невозможным — вычисления в конечных группах. Одним из самых замечательных достижений математики XX столетия явилась классификация конечных простых групп. Как выяснилось, кроме нескольких бесконечных серий таких групп имеется еще ровно 26 “маленьких исключений”, так называемых спорадических групп. Самая большая из этих спорадических групп, называемая “Большим Монстром” или “Дружественным Гигантом”, имеет порядок

$$2^{46} \cdot 3^{20} \cdot 5^9 \cdot 7^6 \cdot 11^2 \cdot 13^3 \cdot 17 \cdot 19 \cdot 23 \cdot 29 \cdot 31 \cdot 41 \cdot 47 \cdot 59 \cdot 71$$

и наименьшая степень ее точного матричного представления равна 196883. Умножение двух матриц такого порядка на может оказаться непосильным для лучших современных рабочих станций, поскольку время, необходимое для вычислений в этой группе традиционными методами, на много порядков превосходит возраст Вселенной (кстати, уже просто хранение большого числа таких матриц и результатов промежуточных вычислений в памяти машины представляет собой достаточно серьезную проблему). В то же время умножение столбца высоты 196883 на матрицу размера $196883 \times 196883$ требует лишь минут машинного времени и вполне осуществимо, так что в последние годы многие специалисты занимаются разработкой алгоритмов “матричных вычислений без матриц”. Для матриц с полиномиальными коэффициентами подобные эффекты наступают уже значительно раньше, скажем на очень важном с точки зрения приложений в физике случае матриц размера $248 \times 248$.

В компьютерной алгебре значительно (бесконечно?) больше математики, чем в традиционном программировании и значительно больше программирования, чем в традиционной математике. В действительности даже для профессионального математика овладение основными принципами компьютерной алгебры на начальном этапе требует значительного интеллектуального усилия. Дело в том, что *кроме* всех обычных математических операций компьютерная алгебра включает в себя громадное количество структурных операций, связанных с манипуляцией с выражениями, и алгоритмических операций, связанных с управлением ходом вычислений (`flow of calculation`). Большинство этих структурных и алгоритмических операций не имеют стандартных названий и обозначений в традиционной математике. Более того, многие из этих новых операций и проводимых при их осуществлении различий в классической математике вообще отсутствуют, по крайней мере на сознательном уровне. Поэтому даже для профессионального математика изучение компьютерной алгебры является ценнейшим опытом, который позволяет взглянуть на уже известные ему математические явления совершенно другими глазами: THE PURPOSE OF TRAVEL IS NOT VISITING NEW PLACES, BUT HAVING NEW EYES.

## § 4. Возможности систем компьютерной алгебры.

> Ленин, как известно, любил гимнастику. Но все же иногда некоторых тяжестей не осиливал. Так, бывало, схватится за бревно, а поднять от земли — слабо.
>
> Виктор Тихомиров. Легенды о революции

> Say, aliens invade the Earth and threaten to obliterate it in one year's time unless we compute $R(5,5)$. If we marshalled the world's best minds and fastest computers, then within a year we could probably calculate the value. If the aliens demanded $R(6,6)$, however, we would be better off preparing for interstellar war.
>
> Paul Erdös

Воспроизведем из книги Вольфрама (глава "The Limits of `Mathematica`") список операций, которые универсальная система компьютерной алгебры такая, как `Mathematica`, в состоянии **за секунды** выполнить на современном персональном компьютере.

• Производить арифметические операции с целыми числами содержащими несколько сотен миллионов десятичных цифр;

• Породить миллион десятичных знаков таких чисел, как $\pi$ или $e$;

• Разложить по степеням многочлен, содержащий миллион слагаемых;

• Разложить на множители многочлен от четырех переменных, содержаший сто тысяч слагаемых;

• Решить систему квадратичных неравенств, имеющую несколько тысяч независимых компонент;

• Найти целые корни разреженного многочлена степени миллион;

• Применить рекуррентное правило миллион раз;

• Вычислить все простые до десяти миллионов;

• Найти численную обратную плотной матрицы размера $1000 \times 1000$;

• Решить разреженную систему линейных уравнений с миллионом неизвестных и ста тысячами ненулевых коэффициентов;

• Вычислить определитель целочисленной матрицы размера $250 \times 250$;

• Вычислить определитель символьной матрицы размера $25 \times 25$;

• Найти приближенные значения корней многочлена степени 200;

• Решить разреженную задачу линейного программирования с несколькими сотнями тысяч переменных;

• Найти преобразование Фурье списка из миллионов элементов;

• Изобразить миллион графических примитивов;

• Отсортировать список из десяти миллионов элементов;

• Найти фрагмент в строке из десяти миллионов знаков;

• Загрузить несколько десятков мегабайт численных данных;

• Отформатировать сотни страниц вывода в традиционной форме.

По этому поводу стоит отметить несколько обстоятельств. Прежде всего стоит указать, что производительность другой `high-end` системы, `Maple` может незначительно отличаться в ту или иную сторону, но в любом случае, время выполнения тех же задач в `Maple` также измеряется несколькими секундами. Тем самым, `Mathematica` и `Maple` могут решить *любую* задачу, которая может реально возникнуть у нематематика, за `half no time`, и выбор между ними не может определяться никакими практическими соображениями, а диктуется исключительно индивидуальными предпочтениями. Поскольку `MatLab Extended Symbolic Math Toolbox` представляет собой клон `Maple`, от него также можно ожидать сопоставимой производительности (при несколько большем, чем собственно у `Maple`, расходовании системного ресурса). Разумеется, на рабочих станциях с несколькими гигабайтами оперативной памяти под операционной системой типа `UNIX`, из всех этих систем можно выжать *гораздо* больше! – количественные характеристики, актуальные для вашего компьютера можно получить в системе помощи `Mathematica` в разделе `Wolfram System Information \ Benchmark`.

От себя добавим в этот список еще один пункт, который Вольфрам не включил ввиду полной очевидности:

• Решение **всех** *вычислительных* задач по математике, которые были выполнены за 15 лет обучения математике в школе и университете.

Подчеркнем, что в этом пункте речь идет о *вычислительных* задачах — таково подавляющее большинство задач, предлагаемых школьникам и студентам. Кроме того, разумеется, учитывается только собственно время работы `CPU`, без ввода условий задач с клавиатуры и вывода на экран.

**Комментарий.** Интересно отметить, что решение задач “на доказательство” требует в сотни раз больше времени. Дело в том, что единственным систематическим методом

доказательства геометрических фактов является введение координат, истолкование геометрических теорем как задач вещественной алгебраической геометрии и последующее применение полиномиальных алгоритмов, связанных с базисами Гребнера. Известно, что уже при нескольких десятках переменных проверка принадлежности многочлена идеалу может представлять собой достаточно серьезную задачу.

С другой стороны, легко придумать и чисто вычислительные задачи, которые поставят в тупик любой современный компьютер и любую систему компьютерной алгебры. Такова, например, задача разложения на множители целого числа со 100 цифрами. Вообще, стоит иметь в виду, что увеличив объем вычислений в задаче, на решение которой требуется несколько секунд, всего в тысячу раз, мы получим задачу, на решение которой требуется час, а в миллион раз — несколько месяцев. При росте объема вычислений всего в миллиарды раз на ее решение потребуются уже столетия! Поэтому полезно понимать, хотя бы в самых общих чертах, как при росте объема данных или значений параметров растет объем вычислений.

## § 5. Об "ошибках" систем компьютерной алгебры

> At the source of every error which is blamed on the computer you will find at least two human errors, including the error of blaming it on the computer.
>
> The Tao of Real Programming

> It seems that creating man God has grossly overestimated his abilities.
>
> Oscar Wilde

> Не только газета, но и короткая запись где-нибудь на стенке в местах общего пользования иногда дает сильный толчок воображению.
>
> Виктор Конецкий, Никто пути пройденного у нас не отберет

Многие критики введения компьютерной алгебры в преподавание, часто упоминают об "ошибках", встречающихся в подобных системах. С нашей точки зрения, все слухи о таких ошибках основаны на устаревшей информации и носят чисто пропагандистский характер, а люди, выдвигающие подобные аргументы, либо не владеют реальной ситуацией, либо сознательно лицемерят.

Разумеется, компьютеру свойственно ошибаться, этим он мало отличается от бога. Следует отдавать себе отчет, что определяющим является не возможность наличия ошибок, а вероятность их осуществления и, в особенности, их этиология. Следует четко отличать несколько причин ошибок:

**1. Ошибки системы.** Часть ошибок относится на счет самих систем компьютерной алгебры.

• **Ошибки в математике и алгоритмах**, использованных в системах компьютерной алгебры. Такие ошибки возможны, но исключительно маловероятны. Например, было обнаружено, что первые версии систем `Maple` или `Mathematica` неправильно считают некоторые интегралы. Выяснилось однако, что ошибки в таблицах интегралов!

• **Программистские ошибки** в системах компьютерной алгебры также возможны, но для коммерческих систем, прошедших многолетнее тестирование и имеющих миллионы пользователей, крайне маловероятны.

• **Конфликты с операционной системой** и другими приложениями. Известно, что в начале и середине 1990-х годов использование систем компьютерной алгебры на бытовых компьютерах очень часто приводило к системным ошибкам и коллапсу системы. Основная причина этого состояла в том, что когда такой системе не хватало памяти для записи результатов промежуточных вычислений, она начинала писать их поверх собственного ядра и даже поверх системных файлов `Windows`. Однако никто не слышал про подобные явления для `Unix`'овских рабочих станций, так что все подобные конфликты следует рассматривать не как дефект систем компьютерной алгебры, а как изъян операционной системы. Кроме того, начиная с `Windows NT` эта проблема устранена.

Мы уверены, что если даже ошибки в таких системах, как `Maple` и `Mathematica` и имеются, то вероятность встретиться с ними настолько мала, что ей можно *полностью* пренебречь.

**2. Ошибки пользователя.** Гораздо более вероятными представляются нам ошибки, допущенные самим пользователем. Значительная часть этих ошибок ничем не отличается от ошибок, возникающих при любой попытке программирования математических задач.

• **Опечатки.** Известно, что из каждых двадцати символов, введенных человеком с клавиатуры компьютера, по крайней мере один является ошибочным. Опечатка в одном символе в большинстве случаев либо делает выражение бессмысленным, либо приводит к вычислению совершенно не того, что имелось в виду. В отличие от многих примитивных языков программирования в больших системах компьютерной алгебры прописные буквы отличаются от строчных. Таким образом, `TexForm` значит совсем не то же самое, что `TeXForm`. Однако в подобных случаях эти системы предупреждает о возможных опечатках: `possible spelling error`.

• **Синтаксические ошибки.** Сюда относится любая попытка вычислить выражение, составленное с нарушением правил языка. Типичными ошибками такого рода являются несбалансированность выражений, вызов в них функций с неправильным числом аргументов или аргументами неправильных форматов, и многое другое. Большая часть этих ошибок моментально обнаруживается, так как система просто откажется вычислять синтаксически неправильное выражение.

• **Программистские ошибки.** Типичные программистские ошибки, возникающие при использовании систем компьютерной алгебры, это использование переменных, которым не были присвоены значения, либо использование старых значений переменных, несогласованность форматов, выясняющаяся при вычислении (т.е. не на лингвистическом, а на семантическом уровне) и многое другое. В большинстве случаев подобного рода ошибки либо порождают сообщение об ошибке, либо приводят к бесконеч-

ной рекурсии. Однако в некоторых случаях (в особенности при использовании старых значений переменных!!) могут возникать чрезвычайно трудно отслеживаемые *невоспроизводимые* ошибки. Имеются несколько стандартных приемов: чистка и локализация переменных, явное задание начальных значений *всех* используемых итераторов и т.д., которые позволяют резко уменьшить вероятность возникновения неотслеживаемых ошибок. Кроме того, как всегда, правильно вначале тестировать любую написанную программу на совсем простых примерах. Если Вы хотите вычислить 10000!, убедитесь вначале, что написанная Вами программа правильно вычисляет 3! – большинство ошибок будет обнаружено уже на этом этапе.

• **Математические ошибки.** На начальном этапе программирования на языках компьютерной алгебры чрезвычайно велика вероятность возникновения *математических* ошибок. Впрочем, такого рода ошибки характерны для начального этапа *любой* попытки уточнения понятий и их перевода с одного языка на другой. Типичными ошибками такого рода являются неправильное определение порядка выполнения операций, неправильная интерпретация функций и т.д. После нескольких месяцев интенсивного использования системы и выработки устойчивого навыка тестировать все программы на легко проверяемых примерах, количество подобных ошибок резко снижается.

Однако, с нашей точки зрения перечисленные в этом пункте ошибки не являются специфичными для компьютерной алгебры, а возникают в любом вычислении достаточно большого объема. /

**3. Непонимание основных принципов компьютерной алгебры.** Основным и с нашей точки зрения НАИБОЛЕЕ СЕРЬЕЗНЫМ ИСТОЧНИКОМ *реальных* ОШИБОК ЯВЛЯЕТСЯ НЕЗНАНИЕ И/ИЛИ НЕПОНИМАНИЕ ОСНОВНЫХ ПРИНЦИПОВ ОРГАНИЗАЦИИ ВЫЧИСЛЕНИЙ В СИСТЕМАХ КОМПЬЮТЕРНОЙ АЛГЕБРЫ. В первую очередь это относится к следующим моментам:

• **Непонимание разницы между формой и значением выражения.** В отличие от всех традиционных вычислительных систем, системы компьютерной алгебры производят вычисление с *формой* выражения, а не только с его *значением*. Это значит, что, система тщательно различает не только сами объекты, но и их имена, имена имен, имена имен имен, и т.д. Например, с точки зрения внутреннего представления данных в системе, выражения $(x+1)(x-1)$ и $x^2-1$ следует рассматривать как абсолютно различные!!! Скажем, неосторожно использовав условный оператор

```
If[(x+1)*(x-1)==x^2-1,1,0],
```

не следует надеяться получить в ответе 1. А вычисление

```
If[(x+1)*(x-1)===x^2-1,1,0]
```

и вовсе вернет 0.

• **Применение правил преобразования.** Системы компьютерной алгебры автоматически производят некоторые типы преобразований, но **не** производят других типов преобразований. В большинстве случаев у конструкторов систем были чрезвычайно серьезные принципиальные и/или

практические основания для принятия подобного рода решений. В то же время не только начинающий, но даже профессиональный программист, не знакомый, однако, со спецификой символьных вычислений, скорее всего, не осознает разницу между, скажем, применением ассоциативности и применением дистрибутивности, и исходит их того, что система должна проводить вычисление так же, как это делал бы в аналогичной ситуации человек. При некотором опыте в большинстве случаев этого действительно можно добиться. Однако то, что системы компьютерной алгебры не всегда делают то, что от них ожидают, совсем не означает, что они бесполезны!!!

• **Использование приближенных вычислений.** Приближенные вычисления представляют собой меч без ручки и проводящий их вынужден держаться за лезвие. Серьезнейшей концептуальной ошибкой, лежащей в основе большинства реально описанных случаев, когда проведение вычисления приводило к неправильному ответу, является применение приближенных вычислений к задачам с точными условиями. Задачи с точными условиями должны обрабатываться только безошибочными алгоритмами. Никаких округлений в процессе вычисления производиться не должно, округляться может только окончательный ответ. Точно так же, приближенные вычисления (без контроля точности) внутри рекуррентной или итеративной процедуры в большинстве случаев абсолютно бессмысленны и *с необходимостью* приводят к ошибочному результату[3].

• **Разбухание промежуточных выражений.** Начинающий обычно не знает, в какой форме задавать вопрос, и пытается вывести на экран то, что с точки зрений целей вычисления является *промежуточным* выражением. Например, в действительности его интересует *длина* некоторого списка, но он пытается вывести сам этот список. Так как форматирование ответа для его вывода на экран в большинстве случаев занимает значительно больше времени, чем само вычисление, такое поведение снижает эффективность использования систем компьютерной алгебры, и даже делает невозможным решение некоторых типов задач, которые при грамотной постановке вопроса и/или организации вычислений решаются за доли секунды.

[3]Совершенно поразительные примеры численной неустойчивости, замечательно иллюстрирующие эту мысль, приведены в статье О.А.Иванов, Современная математика в школьных задачах. — Соросовский Образ. Ж., 2000, т.6, N.6, с.1–7.

# Глава 2. ЧТО ТАКОЕ Mathematica?

> Математическое понимание представляет собой нечто, в корне отличное от вычислительных процессов; вычисления не могут *полностью* заменить понимание. Вычисление способно оказать пониманию чрезвычайно ценную помощь, однако само по себе вычисление *подлинного* понимания не дает. Однако математическое понимание часто оказывается направленно на *отыскание* алгоритмических процедур для решения тех или иных задач. В этом случае алгоритмические процедуры могут взять управление на себя, предоставив интеллекту возможность заняться чем-то другим. Приблизительно таким образом работает хорошая система обозначений — такая, например, как та, что принята в дифференциальном исчислении, или же всем известная десятичная система счисления. Овладев алгоритмом, скажем, умножения чисел, Вы можете выполнять операцию умножения совершенно бездумно, алгоритмически, при этом в процессе умножения Вам совершенно ни к чему "понимать", почему в данной операции применяются именно эти алгоритмические правила, а не какие-то другие.
>
> Роджер Пенроуз. Тени разума. Гл.3. О невычислимости в математическом мышлении

Тот, кто прочел Главу 1, знает, что Mathematica является системой компьютерной алгебры общего назначения, при помощи которой можно решать **любой** тип задач, в которых в той или иной форме встречается математика. При этом система Mathematica наряду с Maple является **единственной** такой high-end[4] системой, которая настолько проста в использовании, что доступна школьникам и студентам младших курсов.

## § 1. Достоинства и особенности системы Mathematica

> Одной из самых удивительных сторон компьютеров является то, что они становятся все лучше и лучше, в то время как все остальное становится все хуже и хуже.
>
> Дональд Кнут. Санкт-Петербургский Университет, N.15, 1994.

По удобству использования, продуманности интерфейса и встроенной помощи, унификации формата применяемых командных слов и конструкций, их предсказуемости и близости к реальному математическому английскому языку, Mathematica значительно удобнее **всех** других систем, включая Maple. Другими принципиальными моментами, которые заставили нас сделать выбор в пользу системы Mathematica в нашей собственной работе, являются поддерживаемый ей более гибкий стиль программирования и более высокое качество графики.

Огромными достоинствами системы Mathematica являются

---

[4]Словарь дает следующие переводы компьютерного термина high-end: мощный, профессиональный, высококачественный; высокого класса; с широкими функциональными возможностями. Поскольку ни один из этих переводов не отражает всего пафоса и всех коннотаций оригинала, мы оставляем этот термин as is.

- Простота использования
- Высочайшая вычислительная эффективность
- Эффективная генерация графики высочайшего качества
- Близость используемого языка к реальной математической практике
- Богатство и гибкость языка
- Высочайшая степень унификации
- Высокая предсказуемость
- Неограниченная расширяемость
- Независимость от платформы
- Совместимость различных версий
- Использование явных форматов

В том, что касается трех последних пунктов, мы можем подтвердить их следующим примером из личного опыта. Один их авторов этой книги впервые познакомился с системой `Mathematica` на презентации фирмы `Wolfram Research` на Международном Математическом Конгрессе в Киото в 1990 году и начал *систематически* использовать ее с 1991 года, в 1991–1992 годах главным образом на платформе `Макинтош`, начиная с 1992 года на `UNIX`'овских рабочих станциях (`DEC`, `Sun`, `HP`) для научных вычислений, а с 1995 года, кроме того, и на `PC` для небольших вычислительных задач и учебных целей. При этом **все** программы, написанные начиная с 1991 года (версии 2.0 и 2.2), под `Макинтош` и `UNIX` оказались *полностью* работоспособными и на `PC` под версии 3.0, 4.0 и 5.0 начала 2000-х. Более того, в блокнотах, написанных под `Макинтош`, правильно воспроизводились даже форматирование, свойства ячеек и экранная графика. Программы написанные под `UNIX` в чисто текстовом режиме первоначально нуждались в некотором дополнительном форматировании, чтобы стать полноценным блокнотом, однако в середине 1990-х годов с появлением версий `Mathematica` под `XWin` и эта проблема была решена. Чтобы быть справедливыми, необходимо отметить, что полной совместимости современных версий с предыдущими нет, и некоторые, казалось бы удобные функции, как например, `InequalityPlot` (до версии 5.2), позволявшая изобразить определяемую неравенствами область на плоскости, прекратили свое существование. Однако, разработчики всегда предлагают новые функции, перекрывающие реализованные ранее возможности. Так, функция `RegionPlot` (версия 6 и позже) включает функциональность предшественницы и близка по правилам обращения. Пример использования этой функции приведен в § 6 Главы 3. Подробная информация о совместимости версий системы `Mathematica` представлена в разделе `Version Advisory` системы Помощи `Mathematica` (см. ниже в § 4), в частности, в материалах `Incompatible Changes since Mathematica Version 1` и `Standard Packages Compatibility Guide`. На сайте www.wolfram.com в разделе `Краткая история изменений системы Mathematica`[5] перечислены нововведения от версии к версии, а система Помощи

[5] https://www.wolfram.com/mathematica/quick-revision-history.html

Mathematica при поиске справки о функции, которая более не поддерживается, сообщает название и правила использования ее заменившей, причем с обязательным примером, как использовалась прежняя функция.

**Концептуальное программирование вместо процедурного и рекурсивного**

> A language that doesn't affect the way you think about programming is not worth knowing.
>
> Dennis M. Ritchie

Основная догма традиционного процедурного программирования в стиле ПошелНа (GoTo) состоит в том, что компилируемая программа **всегда** исполняется быстрее, чем интерпретируемая. Язык Mathematica поддерживает все стили программирования, включая, конечно, и *процедурное* программирование, хотя в гораздо большей степени ему свойственны *функциональное* программирование, основанное на рекурсии и *концептуальное* программирование, состоящее в том, что мы даем прямое *математическое* определение того, что хотим вычислить.

Проиллюстрируем различные стили программирования на примере вычисления $n!$. Конечно, в ядре системы Mathematica есть функция Factorial, вычисляющая факториал, поэтому этот пример приводится исключительно с тем, чтобы сравнить на простейшем материале разные стили программирования. Вот как, примерно, могла бы выглядеть в языке Mathematica программа для определения факториала в стиле ПошелНа:

```
In[1]:=factor1[n_]:=Block[{m=1},For[i=1,i<=n,i++,m=m*i];m]
```

Однако гораздо быстрее написать рекуррентное определение факториала в стиле функционального программирования

```
In[2]:=factor2[0]=1; factor2[n_]:=factor2[n-1]*n
```

Конечно, более опытный программист, который экономит свое время, а не время компьютера, именно так и поступит. Однако одной из самых сильных сторон языка Mathematica является обилие мощных встроенных функций для работы с выражениями, в частности, списками и применения функций к различным их уровням. Поэтому тот, кто не является программистом, но приобщился к *йоге* системы Mathematica определит $n!$ либо как произведение элементов списка $\{1, \ldots, n\}$:

```
In[3]:=factor3[n_]:=Apply[Times,Range[n]]
```

либо просто как произведение чисел от 1 до $n$:

```
In[4]:=factor4[n_]:=Product[i,{i,1,n}]
```

А теперь сравним, сколько времени занимает вычисление 50000! при помощи четырех этих программ. Мы думаем, что результат окажется ошеломляющим для сторонников традиционного программирования. Вот как выглядит фактический результат[6] (для получения которого в зависимости

[6]Приведены результаты вычислений на стандартном персональном компьютере класса Pentium выпуска начала 2000-х. Ноутбук с процессором x64 класса Intel Core I7

от того, как сконфигурировано ядро Вашей версии, может понадобиться изменить глубину рекурсии, сделав ее *достаточно* большой, например, бесконечной `$RecursionLimit=Infinity`):

```
In[5]:=Timing[factor1[50000];]
Out[5]={2.534,Null}
In[6]:=Timing[factor2[50000];]
Out[6]={2.493,Null}
In[7]:=Timing[factor3[50000];]
Out[7]={0.251,Null}
In[8]:=Timing[factor4[50000];]
Out[8]={0.231,Null}
```

Производит впечатление, не так ли? Рекурсивная программа работает не медленнее, а даже чуть быстрее, чем процедурная, а работа со списком **в десять раз быстрее**, чем каждая из них!!! А вот с какой, примерно, скоростью вычисляется оптимизированная внутренняя функция `Factorial`:

```
In[9]:=Timing[Factorial[50000];]
Out[9]={0.121,Null}
```

Мы видим, что это еще примерно в два раза быстрее, чем работа со списком, что, впрочем, неудивительно, так как код, описывающий встроенные команды, использует быстрые алгоритмы и оптимизирован по скорости.

Философский вывод из этого состоит в следующем: ЧЕМ ПРОЩЕ НАПИСАНА ПРОГРАММА, ТЕМ БЫСТРЕЕ ОНА РАБОТАЕТ! Быстрее всего работают внутренние функции системы `Mathematica`, потом математические определения, которые Вы даете в терминах этих функций. Программы в традиционном стиле — притом как процедурном, так и функциональном!!! — работают значительно медленнее. УСТОЙЧИВЫЕ НАВЫКИ ТРАДИЦИОННОГО ПРОГРАММИРОВАНИЯ ЯВЛЯЮТСЯ СКОРЕЕ ПОМЕХОЙ, ЧЕМ ПОМОЩЬЮ ДЛЯ ЭФФЕКТИВНОГО ПРОГРАММИРОВАНИЯ на языке `Mathematica`, зато понимание смысла и *математической* структуры используемых объектов решающим образом ускоряет не только процесс написания программ, но их их работу.

**Недостатки системы `Mathematica`.** Первые версии системы `Mathematica` обладали рядом серьезных недостатков, начиная с невозможностью прервать вычисление. В сочетании с отсутствием комбинации из трех пальцев `Ctrl+Alt+Del` в тогдашней операционной системе `MacOS`, это часто приводило к драматическим последствиям. Однако в дальнейшем разработчики системы планомерно устраняли все *очевидные* недостатки: *there are two ways to make a perfect piece of software. One is to make it so simple, that there obviously are no deficiencies. Another one is to make it so complicated, that*

---

выпуска 2018 года выполняет расчет на порядок быстрее, но качественно результаты сохраняются.

*there are no obvious deficiencies.* Создатели системы Mathematica пошли по второму пути.

Единственным очевидным недостатком системы является высокая стоимость лицензии (в мае 2019 года - 985$ за Stardard Desktop/Cloud). Впрочем, компания Wolfram Research предлагает систему студенческих и академических скидок и версия 12 для персонального компьютера обойдется студенту в 160$. Кроме того, имеется возможность купить подписку для он-лайн доступа на год и даже на семестр, которые стоят в разы дешевле.

Если позволить себе пошутить, то можно упрекнуть Mathematica в том, что она не умеет варить кофе. Впрочем, NOBODY IS PERFECT, ведь даже emacs, который умеет *все*, варит очень плохой кофе.

## § 2. Структура системы Mathematica

> There are things on heaven and earth, Horatio, Man was not meant to know. — На свете есть много такого, Горацио, чего человеку знать не положено.
>
> William Shakespear. Hamlet

Mathematica является одной из *самых* сложных до сих пор написанных *общедоступных* систем программного обеспечения. Более крупные системы как правило создавались лишь для каких-то очень специальных целей, таких как астрономические, ядерные, космические исследования, проектирование, геологоразведка, военное планирование, крупные экономические задачи и т.д.

**1. Быстрый старт.** Запуск системы осуществляется стандартным образом: щелчком по иконке в строке быстрого доступа на экране персонального компьютера или через меню приложений операционной системы. После запуска появляется окно приветствия, предоставляющее доступ к списку последних открытых на локальном компьютере или в "облаке" файлов (Recent files), разделу документации (Documentation), ресурсам пакета на сайте разработчика (Resources)[7] и сайту сообщества (Community) в интернете, объединяющем пользователей системы Mathematica. Переключателем Show at Startup показ окна приветствия можно отменить, однако, учитывая, что раздел документации является по сути прекрасным гипертекстовым учебником по языку и системе Mathematica, содержащим замечательные примеры и иллюстрации, и к тому же исполненным на языке системы, для изучающих Mathematica регулярный старт с просмотра разделов документации может оказаться небесполезным. Ресурсы системы Mathematica содержат целый спектр разделов: от книги создателя системы Стивена Вольфрама "Элементарное введение в язык Wolfram Language"[8] и демонстраций, содержащих полнофункциональные интерактивные примеры с исходным кодом, до материалов для программистов и каталога книг по системе Mathematica и языку Wolfram Language.

---

[7] http://www.wolfram.com

[8] www.wolfram.com/language/elementary-introduction/2nd-ed/

**2. Структура системы.** Полная установка системы Mathematica для операционной системы MS Windows занимает около 10 гигабайт дисковой памяти и состоит из более, чем 40000 файлов, сгруппированных в почти 4000 папок. По умолчанию система устанавливается в папку

C:\Program_Files\Wolfram_Research\Mathematics\<номер версии>

Папка содержит большие поддиректории SystemFiles, Documentation и AddInd, служебный каталог Configuration, запускающий файл Mathematica.exe и файлы MathKernel.exe / WolframKernel.exe и math.exe / wolfram.exe, обеспечивающие разными способами доступ к ядру системы, реализованному динамической библиотекой WolframEngine.dll. Внимательный читатель обратит внимание, что объединенные в пары файлы одинаковы по размеру. Более подробно о структуре системы Mathematica можно прочитать в учебнике на сайте Wolfram Research в статье "The Structure of the Wolfram System"[9].

Система Mathematica построена по модульному принципу и её ядро, фактически выполняющее вычисления, отделено от интерфейса, который обрабатывает взаимодействие с пользователем. Структурно система состоит из следующих основных компонентов:

- MathKernel – **ядро системы**, обеспечивающее вычисления;
- FrontEnd – **интерфейс**, отвечающий за диалог с пользователем;
- **процедуры** обмена данными MathLink, JLink, NETLink и т.д.
- **пакеты расширений** Add-ons, Packages, Dictionaries, Graphics.

Ниже мы чуть подробнее опишем некоторые аспекты, связанные с реализацией и функциями ядра, а также стандартные пакеты.

**3. MathKernel.** Основной частью системы, определяющей ее вычислительные возможности, является, конечно, ядро MathKernel. В свою очередь ядро всех высокоуровневых систем компьютерной алгебры состоит из двух частей. Во-первых, это скомпилированный код на каком-то *низкоуровневом* языке, как правило, C или Lisp. Во-вторых, это высокоуровневый код в языке самой системы, содержащий таблицы (основные математические формулы, таблицы интегралов, интегральных преобразований и т.д.) В системах Maple и Mathematica и большинстве других новых систем низкоуровневый код написан на C или C++, в то время как в Axiom и большинстве более старых систем — на Lisp.

В настоящее время ядро Mathematica включает миллионы строк кода на языке C. К этому количеству следует добавить сотни тысяч строк кода, написанных собственно в системе Mathematica, т.е. на языке существенно более высокого уровня. Стоит представить себе, что 1 500 000 строк в C это около 50 тысяч печатных страниц, т.е. 100 книг по 500 страниц. Подчеркнем, что речь идет исключительно о коде в C.

[9] reference.wolfram.com/language/tutorial/TheStructureOfTheWolframSystem.html

**4. FrontEnd.** Реализация интерфейса `FrontEnd` занимает — в зависимости от платформы — еще несколько сотен тысяч строк кода на `C`. Значительная часть этого кода посвящена поддержанию многочисленных форматов ввода–вывода. Кроме того, `FrontEnd` содержит достаточно мощный текстовый редактор (`WordProcessor`). Достаточно сказать, что `Mathematica` содержит словарь и `spell checker` на 125 000 слов, из которых около 100 000 — обычные слова английского языка, около 20 000 — математические и научные термины и около 5 000 — внутренние и системные команды, определенные в ядре `Mathematica` и ее *стандартных* расширениях.

Интерфейс `FrontEnd` работает независимо от ядра системы `MathKernel`. Эти две части системы могут работать по отдельности, например, первая - на компьютере пользователя с хорошим монитором для вывода качественных изображений, а ядро - на удаленном компьютере с мощным процессором, для выполнения собственно вычислений. Интерфейс `FrontEnd` может запускать и несколько ядер для выполнения параллельных высчислений на нескольких компьютерах. Взаимодействие интерфейсной части и ядра системы обеспечивается компонентом пакета `MathLink`.

При старте системы `Mathematica` файл `Mathematica.exe` запускает интерфейс `FrontEnd` и Вы фактически имеете дело со специализированным *текстовым и графическим редактором*, а вовсе не с системой символьных вычислений. Так происходит до тех пор, пока Вы первый раз не набрали на клавиатуре комбинацию клавиш `Shift+Enter`. В этот момент происходит вызов ядра, которое может загружаться, в зависимости от конфигурации и других выполняемых системой процессов, несколько секунд. С другой стороны, Вы можете просто запустить на выполнение программу `MathKernel.exe`. В этом случае Вы обращаетесь *непосредственно* к вычислительному ядру системы, минуя `FrontEnd`, и общение с ядром происходит в текстовом режиме, так что графика выглядит несколько ностальгически, напоминая об удаленных терминалах начала 1990-х годов.

**5. Версии системы.** Начиная с 1988 года, до апреля 2019 года выпущено основных 12 версий системы `Mathematica`. Разработчики требовательно подходят к выпуску очередного релиза и каждая новая версия действительно оказывается функциональнее, эффективнее, полнее и удобней предыдущих. Описание текущей версии системы и список ранее выпущенных с дополнениями и изменениями представлены на сайте www.wolfram.com в разделе “Краткая история изменений системы Mathematica”[10].

[10] http://www.wolfram.com/mathematica/quick-revision-history.html/

## § 3. Главное меню Mathematica

> Documentation is like sex: when it is good, it is very, very good; and when it is bad, it is better than nothing.
>
> Dick Brandon

При запуске системы Mathematica открывается новый блокнот с рабочим названием Untitled-1. Рабочее окно содержит системную кнопку стандартного назначения в верхнем левом углу и строку **главного меню**, содержащую 10 разделов. В зависимости от настроек системы, при запуске может открываться приветствие Welcome Screen и одна или несколько палитр Palettes. В настоящем параграфе мы опишем общую структуру главного меню и пройдемся коротко по его разделам, опуская при этом описание тех очевидных команд, которые в системе Mathematica действуют точно так же как в любой другой программе под MacOS, Windows или XWin. Тот, кто использовал одну такую программу, использовал их все и поэтому нет нужды повторять очевидные вещи. Каждый, кто сам хоть раз включал компьютер, помнит и общеупотребимые комбинации "горячих" клавиш от Ctrl+A до Ctrl+Z.

• File — этот раздел содержит обычные команды: New, Open, Close, Save / Save As, Print, Exit и три подменю, на которые следует обратить специальное внимание. Специфичными для системы Mathematica являются подменю Save to Wolfram Cloud, Publish to Cloud и CDF Prewiew. Первое позволяет сохранить файл блокнота в интернете в поддеживаемом Wolfram Research "облаке" с возможностью в дальнейшем доступа из сети. С помощью второго можно выложить создаваемый файл в интернете для всех или с доступом по паролю. Третье меню предназначено для сохранения блокнота формате CDF (Computable Document Format) - разработанном компанией Wolfram Research формате документов для создания динамические генерируемого интерактивного контента с возможностями управления, элементами анимации и другими характерными для презентаций атрибутами.

• Edit содержит обычные команды Cut, Copy, Paste, Clear, Select All, Find, Undo / Redo доступные также с помощью стандартных комбинаций горячих клавиш. Отметим, что команда Copy As позволяет копировать выделенный фрагмент в различных форматах, включая LaTeX и MathML. Специфичной для системы Mathematica является команда Check Balance, позволяющая контролировать требуемую правилами синтаксиса системы парность скобок в выражениях. В меню Edit также включены команда вызова системы проверки правописания Check Spelling и ссылка доступа к параметрам системы.

• Insert наряду обычными для текстовых редакторов командами вставки символа, картинки, разрыва страницы или гиперссылки содержит также специфичные для системы Mathematica команды Input from Above и Output from Above, копирующие и помещающие в позицию курсора соответствующее содержание - ввод Input или Ouput ближайшей предшествующей

ячейки Cell. Команда Inline Free-form Input предназначена для ввода произвольного текста в позицию курсора в поле Input для последующего преобразования в конструкции Wolfram Language. Меню Insert также содержит большое количество разнообразных шаблонов для ввода элементов текста, характерных для научных публикаций: надстрочных и подстрочных индексов, таблиц и матриц, библиографических ссылок и др. С помощью подменю Automatic Numbering можно создавать различные нумеруемые структуры: разделы документа, формулы и пр.

• Format содержит большое количество подменю, описывающих стили, шрифты, размеры, цвет, выравнивание и пр. и пр. Style, Font, Face, Size, Text Color, Background Color, Text Alignment, Text Justification, Word Wrapping. Следует отметить, что в меню формат Format представлены стили, как применяемые к отдельным элементам блокнота, так и предназначенные для оформления документа блокнота как целого: Stylesheet и Screen Environment. Подменю Cell Dingbat позволяет выбрать разнообразные графические символы, которыми можно отмечать элементы в блокноте. Доступ ко всей системе опций предоставляет специальный инструмент Option Inspector, который предъявляет их сгруппированными в соответствии с функциями. Данное подменю позволяет не только осуществить настройку конкретного параметра, но и установить уровень, на котором эта настройка будет действовать: глобально, для всего блокнота или для области выбора.

• Cell содержит специфическое для интерфейса системы Mathematica управление ячейками. Подменю Convert To преобразуют вид выбранного фрагмента, установленный по умолчанию. При этом основными возможностями являются четыре формата: два явных InputForm и OutputForm и два неявных: StandardForm и TraditionalForm. Например, если Вы в качестве формата по умолчанию для ячейки ввода установлен StandardForm, то при набираемая в ячейке ввода стрелка -> будет автоматически конвертироваться в → и т.д. Хотя, конечно, ядро системы будет, по-прежнему, видеть то, что Вы *фактически* ввели с клавиатуры, а именно, ->. Существуют различные мнения по поводу того, удобны неявные форматы или, все же, скорее нет. По нашему мнению, явный формат позволяет гораздо легче локализовать синтаксические ошибки. Поэтому если Вы хотите видеть то, с чем *на самом деле* производится вычисление, оставьте InputForm и OutputForm в качестве установок ввода и вывода! В некоторых случаях, предлагаются и другие форматы такие, как Raw InputForm без сопровождающей разметки языка Wolfram Language или платформо-независимый Bitmap. Два других подменю Cell Properties и Cell Grouping связаны со свойствами ячеек и их группировкой. Команды из подменю Cell Properties позволяют изменить такие свойства текущей ячейки, как Open, Editable, Active, Initialization и т.д. С другой стороны, команды из подменю Cell Grouping, позволяет группировать ячейки или разбивать группы, в то время как команды Divide Cell и Merge Cells позволяют разделять и сливать ячейки. Подменю Cell Tags предназначено для по-

иска помеченных ячеек, а подменю Notebook History дает возможность отслеживать историю изменений в блокнотах и ячейках и просматривать эти изменения с привязкой к дате и времени. Последнее может быть полезно, чтобы вспомнить, как пользователь выполнял свои исследования, если например, возникла необходимость изменить их ход. Для специалиста может оказаться полезна и команда Show Expression, демонстрирующая код на языке Wolfram Language, реализующий содержащующуюся в ячейке запись.

• Graphics содержит средства для работы с изображениями, включая графический редактор и средства расположения и группировки.

• Evaluation содержит команды Evaluate для выполнения введенных в ячейку команд, а также вычислений, содержащихся в группе ячеек или всем блокноте. Имеются средства отладки Debugger, команды управления ядром Kernel, организации параллельной обработки и т.д.

• Palette - палитры операторов и функций, предназначенные для облегчения ввода. Это панели инструментов с пиктограммами для ввода математических символов, функций и команд управления системой. Для удобства пользователя инструменты палитр сгруппированы, их можно перемещать по экрану. В систему могут быть установлены дополнительные палитры, в загруженные или разработанные пользователем. В меню Palette в стиле палитр включены также средства выбора вида диаграмм, цветовых схем, специальных символов и средства настройки презентаций, подготовленных на языке Wolfram Language.

• Window содержит средства настройки панели инструментов блокнота, перечисляет все открытые в системе окна, управляет их расположением и размером, а также позволяет увеличить (Magnification) изображение в окне.

• Help обеспечивает доступ к различным системам информации и помощи, а также другим ресурсам, поддерживающим работу системы. Встроенная помощь более подробно описана в следующем параграфе.

## § 4. Система помощи Mathematica

Лев Толстой очень любил играть на балалайке (и, конечно, детей), но не умел.

Даниил Хармс, Веселые ребята

В системе Mathematica есть несколько способов получить помощь, в частности

• возникающее при запуске FrontEnd **приветствие** = Welcome Screen алиас Startup Palette,

• вызываемая из меню FrontEnd **встроенная помощь** = Help Browser,

• вызываемые непосредственно из блокнота или другого документа MathKernel **информационные запросы** ? = Query и ?? = Information.

• справочные команды `Definition`, `FullDefinition`, `Names`.

Обсудим чуть подробнее, что это такое.

**1. Приветствие.** При каждом запуске системы появляется - если Вы его не отключили - приветствие, о котором мы говорили в пункте `1. Быстрый старт` предыдущего параграфа. Необходимую информацию можно найти в разделе документации, на сайте разработчика www.wolfram.com и в материалах сообщества `Wolfram` в интернете. Помимо ответов на всевозможные вопросы пользователей системы `Mathematica` рекомендованные в Приветствии ресурсы интернета позволяют скачать *несколько тысяч* дополнительных пакетов, блокнотов, статей, графических объектов и других документов, содержащих определения *десятков тысяч* функций сверх примерно 5000 тысяч стандартных функций.

**2. Встроенная помощь.** Система `Mathematica` предлагает чрезвычайно детальную интерактивную помощь, которая вызывается через рубрику `Help` главного меню, либо нажатием клавиши `F1`, позволяющей получить справку о выделенной функции.

Структура меню `Help` следующая:

○ `Documentation Center` — обеспечивает доступ к документации по `Wolfram Language`.

○ `Find Selected Function` — показывает информацию по выделенной функции, включая примеры.

○ `Wolfram Website` — ссылка ведет на сайт `Wolfram Research`.

○ `Demonstrations` — ведет на сайт, где представлены избранные демонстрации, выполненные в системе `Mathematica`.

○ `Internet & Mail Settings` — список функций `Mathematica`, требующих подключения к интернету, и параметры настройки подключения.

○ `Give Feedback` — позволяет письменно обратиться к службе поддержки со страницы сайта `Wolfram Research`.

○ `Register this Mathematica` — позволяет зарегистрировать вашу копию системы `Mathematica` и Вас как пользователя.

○ `Why the Beep?` — подскажет, почему раздался звуковой сигнал интерфейса `FrontEnd`.

○ `Why the Coloring?` — поясняет подсветку синтаксиса.

○ `Welcome Screen` — открывает экран приветствия.

○ `About Mathematica` — дает информацию о вашей системе, включая возможность измерить производительность системы на Вашем компьютере.

Следует особо отметить разнообразные и обширные материалы, представленные в `Documentation Center`. По сути это и справочные ресурсы, и прекрасные интерактивные учебники по различным разделам системы `Mathematica` и другим продуктам `Wolfram Research`. В контексте нашего учебника нельзя не упомянуть материалы, представленные в разделе `Resources`: он-лайн учебники, объединенные названием “Fast Introduction”:

для тех, кто осваивает возможности программирования на языке Wolfram Language, и для студентов, изучающих математику. Последний учебник[11] охватывает широкий спектр математических дисциплин для средней школы и за ее пределами - от арифметики до математического анализа и матричной алгебры. Замечательная подборка ярких профессионально исполненных примеров, понятных и начинающему, предназначена для быстрого освоения возможностей языка Wolfram Language для расчетов, графиков и презентаций.

**3. Информационные запросы.** Как и в большинстве других систем информационные запросы ?name и ??name дают информацию об объекте с именем name. А именно, ответ на запрос ?name воспроизводит определение и описывает использование этого объекта. Ответ на запрос ??name, кроме того, перечисляет его атрибуты и опции вместе с их *текущими* значениями. Полная форма запроса Information[name]. Например, если Вы хотите узнать, что делает функция Plot, то Вы можете напечатать ??Plot или Information[Plot]

Если Вы точно не помните имя интересующего Вас объекта, то в запросах, как обычно, можно использовать знак *, называемый в этом случае WildCard или MetaCharacter. Этот знак может ставиться в любое место запроса и заменяет любую конечную последовательность букв, появляющихся в этом месте имени. Например, ответом на запрос ??Plot* является список из 18 имен Plot, PlotLabels, PlotRangeClipping Plot3D, PlotLayout, PlotRangeClipPlanesStyle Plot3Matrix, PlotLegends, PlotRangePadding PlotDivision, PlotMarkers, PlotRegion PlotJoined, PlotPoints, PlotStyle PlotLabel, PlotRange, PlotTheme; в то время как ответом на запрос ??*Plot будет показан список из 64 функций от ArrayPlot до WaveletMatrixPlot, включая, разумеется и Plot.

Информацию об определении объекта можно получить также при помощи команд Definition или FullDefinition. А именно, Definition[name] дает определение объекта name, а FullDefinition[name] — определение самого этого объекта и всех объектов, от которых зависит его определение. Для того, чтобы узнать определение встроенного объекта, нужно использовать команду FullDefinition[name], так как в этом случае Definition[name] даст только список атрибутов и опций вместе с их текущими значениями.

Еще один народный способ увидеть список всех имен, содержащих фрагмент blabla состоит в том, чтобы напечатать Names["*blabla*"]. Например, напечатав в начале сессии Names["*"] Вы получите список **всех** функций, откомпилированных при загрузке ядра (скажем, в нашей версии Mathematica 11.3 при первом вызове ядра компилируются 6293 функции). В дальнейшем в течение сессии при подгрузке дополнительных пакетов и по мере того, как Вы определяете новые объекты, список имен будет увеличиваться.

---

[11] https://www.wolfram.com/language/fast-introduction-for-math-students/en/

## § 5. Палитры

А что нам с этих трехсот грамм будет? Мы же гиперборей.

Венедикт Ерофеев

Из подменю `Palettes` главного меню доступны 9 палитр, влючая 3 в подменю `Other`. Эти палитры весьма неравноценны и по своим возможностям палитры `Basic Math Assistant` и `Classroom Assistant` существенно превосходят все остальные. Фактически эти две палитры являются **виртуальными расширениями клавиатуры**, которые позволяют вводить **все** обычные символы и проводить **все** обычные математические вычисления, из тех, что могут понадобиться студенту младших курсов технических, экономических и естественнонаучных специальностей, вообще не зная языка `Mathematica`!

• `Basic Math Assistant` вызывается как одна палитра, но фактически является блоком из нескольких палитр, содержащих **основные** операций, которые могут понадобиться студенту: от математических констант до средств набора сложных математических выражений.

• `Classroom Assistant` в дополнение к предыдущей палитре имеет блоки с символами навигации, средства форматирования и палитру клавиатуры

• `Algebraic Manipulation` вызывает полтора десятка простейших команд манипуляции с многочленами, рациональными дробями, тригонометрическими и экспоненциальными выражениями. Абсолютно бесполезно тому, кто минут пять работал с системой.

• `Basic Math Input` позволяет вводить в традиционной форме верхние и нижние индексы, дроби, радикалы, суммы и произведения, интегралы и частные производные, матрицы, греческие буквы и несколько десятков наиболее употребительных математических знаков.

• `Basic Typesetting` — примерно то же самое, что `Basic Math Input`, но несколько обширнее и позволяет вводить в традиционной форме не только матрицы, но и таблицы, системы уравнений и пр., около сотни специальных знаков, и некоторые другие фишки и дингбаты, символы специальных клавиш, и тому подобное.

• `Chart Element Schemes` предназначен для удобного выбора всевозможных двумерных и 3D диаграмм

• `Chart Element Schemes` дает возможность выбирать средства установки цветов в документе блокнота

• `Special Characters` позволяет вводить буквы с диакритическими знаками, используемые в романских, славянских, германских, и некоторых других языках, символы валют и т.д.

При желании можно создать свои собственные палитры, установить дополнительные и настроить главное меню так, чтобы необходимые автоматически вызывались при запуске системы.

## § 6. Сессии и вычисления

Если Вы работаете с системой `Mathematica` под `Windows`, `MacOS` или `XWin`, то, скорее всего, основными понятиями, в терминах которых происходит Ваше взаимодействие с системой, на внутреннем уровне (`BackEnd`) являются **сессии** и **вычисления**, а на внешнем уровне (`FrontEnd`) — **блокноты** и **ячейки**.

**Предостережение.** Разумеется, это не относится к случаям профессионального использования системы, скажем, в чисто текстовом режиме (в этом случае блокноту отвечал бы файл, а ячейке — командная строка), в многопроцессорном или многосессионном режиме и т.д. Однако все это вряд ли представляет интерес для начинающего.

Между внутренними и внешними понятиями можно установить следующее приблизительное соответствие:

| `Kernel` | `FrontEnd` |
|---|---|
| `Session` = сессия | `Notebook` = блокнот |
| `Evaluation` = вычисление | `Cell` = ячейка |

Разумеется, это соответствие, а не биекция, во время одной сессии можно открывать несколько блокнотов и, наоборот, записанный блокнот может использоваться на протяжении нескольких сессий. Точно так же одна и та же ячейка может в разные моменты сессии вызываться для *различных* вычислений.

**Комментарий.** Непосвященному предыдущее заявление может показаться нелепым, как же один и тот же текст может порождать разные вычисления. Ну, во-первых, текст ячейки может редактироваться. Во-вторых, результат любого вычисления зависит не только от того, что мы вычисляем, но и от того, в каком состоянии находится система в момент вычисления. Например, могут измениться определения использованных в этом вычислении функций — или функций, фигурирующих в определении этих функций, — а также значения различных переменных или параметров, опций и атрибутов. Среди этих параметров могут быть и такие, которые при нормальных условиях не видны пользователю, например, значение затравки генератора случайных чисел, зависящее от времени суток. Кроме того, поведение системы зависит от фаз луны — `You are lucky, full Moon tonight!` — и многих других факторов.

Обсудим эти ключевые понятия чуть подробнее. С внешней точки зрения **сессия** представляет собой период непрерывной работы ядра `MathKernel` между его вызовом и прекращением его работы (безразлично, по причине сознательного выхода или самопроизвольного коллапса). Все функции сохраняют свои определения, а все переменные — свои значения на протяжении всей сессии, по крайней мере до тех пор, пока эти функции или переменные не были удалены, или их определения и значения не были вычищены или модифицированы. Все введенные во время сессии, но не сохраненные в блокнот или файл определения и все вычисленные, но не сохраненные значения теряются!!!

С внутренней точки зрения сессия представляет собой последовательность **вычислений** (`Evaluation`). Типичная сессия состоит из нескольких десятков, нескольких сотен или, в исключительных случаях, нескольких тысяч вычислений. Для того, чтобы вычислить какое-то выражение, нуж-

но мышкой, либо посредством навигационных клавиш ↑ и ↓ поместить курсор в содержащую его ячейку и нажать Shift+Enter.

**Предостережение.** Разумеется, в тот момент, когда мы начинаем *всерьез* использовать Mathematica как язык программирования в традиционном стиле, многие вычисления в этом смысле будут состоять в изменении установок каких-то функций, постановке меток, передаче управления, удержании или вбрасывании каких-то значений или даже в том, чтобы *отложить вычисление* какой-то функции. Тем не менее, с точки зрения ядра каждое из этих действий, в том числе удержание и откладывание, является **вычислением** некоторого специального вида и как внутренне, так и внешне оформляется по тому же регламенту, что вычисление 1+1.

• Любое вычисление можно **приостановить** либо программно, командой Interrupt[], либо в диалоговом режиме выбором в разделе Kernel главного меню команды InterruptEvaluation, горячий ключ Alt-, — это легко запомнить, если знать, что alt = halt представляет собой основную команду из лексикона итальянских карабинеров.

• Любое вычисление можно **прервать** программно командой Abort[], либо в диалоговом режиме выбором в разделе Kernel главного меню команды AbortEvaluation, горячий ключ Alt-. — ibid.

• Первый из этих способов применяется, например, при отладке сложных программ, а второй — в случае, когда вычисление занимает неожиданно много времени и у нас возникло подозрение, постепенно переходящее в уверенность, что либо определения каких-то используемых в нем функций ошибочны, либо значения каких-то параметров слишком велики.

## § 7. Блокноты и ячейки

Начиная с версии 3.0 интерфейс Mathematica организован в форме **блокнотов** (Notebook). Блокнотом называется интерактивный документ, содержащий программу, текст, результаты вычислений, сообщения, графику, таблицы и т.д. Блокнот может являться аналогом рабочей записной книжки, законченной научной статьи или текста учебного характера. Существующий блокнот открывается при помощи команды Open меню File, команда New того же меню создает *новый* блокнот. Во время сессии Вы можете открывать или создавать несколько блокнотов и, наоборот, один и тот же блокнот может использоваться в большом количестве сессий. В конце сессии не забудьте записать (Save) изменения во всех открытых Вами блокнотах.

В свою очередь, каждый блокнот организован как иерархическая структура, состоящая из ячеек или клеток (Cell). Каждая ячейка включает одну или несколько строк, соединенных стоящей справа **скобкой** (Bracket). Некоторые ячейки создаются пользователем, в то время как другие ячейки, содержащие сообщения об ошибках, результаты вычисления, сигналы подтверждения, статус системы и другие виды вывода, создаются самой системой в процессе сессии.

Ячейки объединяются в **группы** (Groups). Входящие в группу ячейки соединяются общей скобкой, включающей скобки нескольких объединяемых ячеек. Например, система *автоматически* объединяет в одну группу

**ячейку ввода** (`Input Cell` или `Evaluatable Cell`) и получающиеся при ее исполнении **ячейки вывода** (`Output Cell`). При помощи содержащихся в меню `Cell` подменю `Cell Grouping` пользователь может вручную произвольным образом сгруппировать ячейки (`Group`) или, напротив, разбить имеющиеся группы (`Ungoup`).

Чтобы начать **новую** ячейку, переместите курсор в такую позицию, где он становится *горизонтальным* — внутри *существующей* ячейки курсор всегда *вертикален*. Щелкнув в этот момент по *левой* кнопке мыши, Вы создите горизонтальную черту (`cell insertion bar`). Начав печатать, Вы создадите новую ячейку. По умолчанию эта новая ячейка всегда имеет формат *ячейки ввода*. Чтобы изменить ее формат, щелкните по скобке, после чего найдите нужный формат в меню `Cell`, подменю `Cell Properties`. Другой народный способ начать новую ячейку состоит в том, чтобы нажать `Alt+Enter`.

Для того, чтобы **вычислить** (`evaluate`) содержимое ячейки ввода, поместите курсор в эту ячейку и нажмите `Shift+Enter`. При этом ячейке автоматически будет присвоен **промпт ввода** (`Input Prompt`) формата `In[n]:=`, а результат вычисления *через какое-то время* появится в ячейке вывода, имеющей заголовок `Out[n]=` с тем же номером. В дальнейшем Вы можете ссылаться на $n$-й ввод как `In[n]`, а на результат $n$-го вывода как `Out[n]`. Обратите внимание на разницу в формате: `In[n]` представляет собой *отложенное* значение и на протяжении сессии его вычисление может приводить к различным результатам, в то время как `Out[n]` представляет собой фактическое значение $n$-го вывода.

| | |
|---|---|
| `Enter` | начать новую строку в текущей ячейке |
| `Shift+Enter` | вычислить содержание текущей ячейки |
| `Alt+Enter` | начать новую ячейку |

На начальном этапе работы придерживайтесь правила

ОДИН ПРОМПТ — ОДНА ФУНКЦИЯ — ОДНА ЯЧЕЙКА

Иными словами, это значит, что в большинстве случаев, когда Вам хочется просто нажать `Enter`, то, что Вам в действительности нужно — это `Alt+Enter`!!!

**Предостережение.** При интерпретации инпута `Mathematica` имеет обыкновение игнорировать большую часть пробелов, табулирования и перенос строки!!! Как показывает наш опыт преподавания, именно непонимание этого фундаментального обстоятельства приводит к половине всех возникающих у студентов ошибок, некоторые из которых трудно отслеживаются. А именно, определив одну функцию, они два раза нажимают на `Enter`, после чего начинают **в той же ячейке** определять то, что они считают другой функцией. Но `Mathematica` продолжает интерпретировать все дальнейшее содержание той же ячейки, не отделенное точкой с запятой, частью определения первой функции!!! Это значит, что при обращении к этому определению либо (как правило!) обнаружится синтаксическая ошибка,

либо начнутся другие увлекательные явления типа бесконечной рекурсии, либо произойдет что-либо еще более драматическое. Довольно часто единственный способ выпутаться из получающегося положения состоит в том, чтобы закончить сессию и перезапустить ядро.

## § 8. Общие советы и типичные ошибки

> Nessun effetto è in natura sanza ragione; intendi la ragione e non ti bisogna sperienza. — В природе ничто не происходит без причины; пойми эту причину и тебе не будут нужны никакие эксперименты.
>
> Leonardo da Vinci

> `Mathematica` is a complex piece of software, and coming to terms with this complexity can impose a steep learning curve upon a student whose primary interest is in learning some mathematics. In my experience of teaching with `Mathematica`, it seems to be a product whic can too easily divide students very quickly into lovers and haters — not least I'm sure because of its rich but strict syntax.
>
> Philip Kent

В настоящем параграфе мы опишем несколько простейших правил синтаксиса системы `Mathematica`. Все эти правила детально обсуждаются и иллюстрируются в дальнейшем, но по нашему опыту по крайней мере 90–95% всех ошибок, совершаемых начинающими, связано именно с нарушением этих простейших правил. Всякий, кто научился писать `Sin[x*y]^2` вместо `sin^2(xy)`, может решить при помощи системы `Mathematica` *любую* задачу из вузовского курса математики за первые три курса.

• Все имена встроенных функций состоят из полных английских слов или общепринятых сокращений и начинаются с заглавной буквы, например, экспонента обозначается `Exp`, а не `exp`, логарифм — `Log`, а не `log` и т.д. Если имя функции состоит из нескольких слов, они пишутся слитно, без знаков препинания, причем каждое из них начинается с заглавной буквы.

• Аргументы функций всегда пишутся в квадратных скобках, $\cos(x)$ обозначается `Cos[x]`, а не `cos(x)`. Различные аргументы разделяются запятой, скажем, $f(x,y,z)$ вводится как `f[x,y,z]`. Круглые скобки используются только для группировки.

• Фигурные скобки используются для обозначения списков, наборов и множеств. Векторы и матрицы в `Mathematica` интерпретируются как списки. Например, вектор с $(x,y,z)$ вводится как `{x,y,z}`, а матрица $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ как `{{a,b},{c,d}}`.

• Произведение обозначается через $*$. Выражение `xy` интерпретируется не как произведение `x` и `y`, а как новая переменная. Правильное обозначение произведения в операторной форме `x*y`, а в полной форме `Times[x,y]`.

**Предостережение.** В действительности *иногда* — но далеко не всегда!!! — пробел тоже интерпретируется как умножение, так что `x y` будет истолковано точно так же, как `x*y`. Единственный способ узнать, как ядро на самом деле понимает Ваш ввод, состоит, конечно, в том, чтобы взглянуть на полную форму того, что Вы печатаете. Так вот,

```
FullForm[x y]===FullForm[x*y]===Times[x,y],
```

но начинающему лучше не знать об этом и всюду ставить $*$. Есть несколько особых случаев, когда отстутствие пробела после цифры, после специального знака или перед ним тоже интерпретируется как умножение. Например, `2x` значит то же самое, что `2*x`, а `x(y+z)` — то же самое, что `x*(y+z)`, но начинающему лучше вообще не пользоваться подобными сокращениями, тем более, что `x2` значит совсем не то же самое, что `x*2`. Иными словами, `FullForm[x2]===x2`, а `FullForm[x 2]===FullForm[x*2]===Times[2,x]`).

• Потенцирование записывается как `x^y`, или, в полной форме как `Power[x,y]`. Конечно, Вы можете вызвать потенцирование в традиционной форме $x^y$ через любую из палитр `Basic Math Assistant`, `Basic Math Input` или `Basic Typesetting`, однако через полчаса работы с системой Вы убедитесь, что гораздо быстрее и удобнее набирать его в форме `x^y`.

• Не жалейте скобок. Представления системы `Mathematica` о приоритете выполнения арифметических операций могут не совпадать с Вашим замыслом. Поэтому за исключением тех случаев, когда Вы абсолютно точно знаете, что происходит, никогда не применяйте два оператора подряд, не указав, в каком порядке они выполняются. Например, вместо `x/y*z` пишите `(x/y)*z` или `x/(y*z)`, as appropriate.

• Мало напечатать текст, нужно ввести его в ядро. До тех пор, пока Вы фактически не предложили системе произвести вычисление, т.е. не нажали `Shift+Enter` на текущей ячейке, содержимое этой ячейки является достоянием интерфейса, но не ядра, блокнота но не сессии! Тем самым, все введенные в этой ячейке функции остаются с точки зрения программы неопределенными, а все переменные, которым в этой ячейке присвоено значение — символами.

• По умолчанию одна ячейка должна содержать одно вычисляемое выражение. Все пробелы, табулирования и переносы игнорируются. Для отделения двух выражений они должны находиться не в разных строках, а в разных ячейках. Для создания новой ячейки нажмите `Alt+Enter`, либо переведите курсор на одну позицию вниз посредством $\downarrow$ и начните новый ввод.

• Если Вы все же хотите поместить два или несколько вычисляемых выражений в одну ячейку, то они должны разделяться точкой с запятой, скажем `f[x]; g[x]`. Однако в этом случае будет отображен только результат последнего вычисления. Если Вы хотите увидеть оба результата, необходимо оформлять вычисление как список, в этом случае все вычисляемые выражения заключаются в фигурные скобки и разделяются запятой {`f[x],g[x]`}.

### Несколько общих советов в завершение

> Experience is that marvelous thing that enables you recognize a mistake when you make it again.
>
> F. P. Jones
>
> Experience is what causes a person to make new mistakes instead of old ones.
>
> Oscar Wilde

• Пишите простые программы. Экономьте свое время, а не время компьютера. Во всех случаях ясность и правильность программы важнее ее эффективности.

• Проверяйте, что Вы правильно понимаете использование внутренних функций и тестируйте работу каждой написанной Вами функции на совсем маленьких примерах, когда Вы можете проверить правильность ответа в уме, и на примерах с известным ответом.

• Главное правило оптимизации программ: don't do it!!! Если программа работает, не пытайтесь ее улучшить.

• Присваивайте глобальным и редко встречающимся переменным длинные имена, а локальным и часто встречающимся — короткие имена.

• Объясняйте системе то, что Вы хотите посчитать, а не то, как она должна это сделать. По возможности используйте встроенные функции. Не пытайтесь предписывать системе конкретные процедуры. Если Вы не являетесь профессиональным вычислителем, то любой встроенный алгоритм лучше **любого** алгоритма, который Вы можете придумать.

• Разбивайте любой алгоритм на последовательности определений. Любая программа, в которой больше, чем две или три строчки, неправильно написана и должна быть разбита на последовательность более коротких программ.

• Применяйте к промежуточным результатам функции упрощения `Simplify`, `FullSimplify`, `Refine` и т.д. Система, как правило, не производит упрощений автоматически.

• Команды решения уравнений такие как `Solve` как правило дают не все, а лишь *какие-то* решения уравнений. В особенности это относится к трансцендентным уравнениям. Проверяйте полученные решения. В случае, если Вам нужно настоящее решение, применяйте команды `Reduce`, `Eliminate` и т.д.

• В рекуррентных процедурах запоминайте промежуточные результаты при помощи конструкции `Remember f[n_]:=f[n]=...`

• Заменяйте циклы работой со списками или встроенными итерационными процедурами такими как `Do`, `Sum`, `Product` и т.д.

• Никогда не применяйте приближенные вычисления к задачам с точными условиями. Задачи в целых или рациональных числах должны обрабатываться только алгоритмами бесконечной точности. Приближенные вычисления могут применяться только к задачам с приближенными условями.

• Не производите никаких округлений в процессе вычисления. Проводите все вычисления с бесконечной точностью и переходите к численным значениям только на последнем шаге

• Никогда не используйте приближенные вычисления в рекуррентных или итерационных процедурах!! Результат итерационной процедуры, выполненной без контроля сходимости и устойчивости, без изучения обусловленности и оценок ошибок, не может быть ничем иным, кроме артефакта.

# Глава 3. ПРАКТИЧЕСКОЕ ВВЕДЕНИЕ В СИСТЕМУ Mathematica

Das sieht schon besser aus! Man sieht doch wo un wie! — А вот это уже лучше! По крайней мере видно, куда и как!

Johann Wolfgang von Goethe, Faust I

Математика может быть определена как единственная наука, в которой мы

- знаем, о чем мы говорим;
- знаем, что мы говорим;
- знаем, верно ли то, что мы говорим[12].

Bertran Russell

— Вибрационализм, — сказал Никсим Сколповский, обращаясь к нескольким пожилым женщинам, по виду — работницам фабрики “Буревестник”, непонятно как оказавшимся на авангардной выставке, — это направление в искусстве, исходящее из того, что мы живем в колеблющемся мире и сами являемся совокупностью колебаний.
Женщины испуганно притихли. Никсим поправил непрозрачные очки с узкими прорезями и продолжил: — Но простое отражение этой концепции в артефакте еще не приведет к появлению произведения вибрационалистического искусства. Чистая фиксация идей неминуемо отбросит нас на исхоженный пустырь концептуализма. С другой стороны, возможность вибрационалистической интерпретации любого художественного объекта приводит к тому, что границы вибрационализма оказываются размытыми и как бы несуществующими. Поэтому задача художника-вибрационалиста — проскочить между Сциллой концептуализма и Харибдой теоретизирования постфактум.

Виктор Пелевин. Встроенный напоминатель

В связи с экспансией митьковского движения на север, юг, запад и восток давно пора подготовиться к массовому паломничеству иностранных туристов в места скопления митьков.
Халатное, поверхностное знакомство с митьковской лексикой приводит к быстрому искажению и, в конечном счете, вырождению смысла цитат и выражений.
Отсюда видна необходимость дополнить список употребляемых выражений и адаптировать его до уровня разумения иностранца.
Итак, что должен знать тот же мистер Майер, американский миллионер, если он является начинающим митьком?

[12]Именно в таком порядке: мы *только* потому знаем, верно ли то, что мы говорим, что мы точно знаем, что именно мы говорим. С другой стороны, мы *только* потому знаем, что мы говорим, что мы абсолютно точно знаем, о чем именно мы говорим. В ответ на весь слабоумный бред, который по этому поводу несли философы от Гегеля до Виттгенштейна, в этой главе мы констатируем, что у математики есть субстрат. Более того, из этого субстрата она и произрастает.

> Ниже приводятся новые митьковские слова, выражения м цитаты, которые м-ру Майеру необходимо выучить для полноценного общения с окружающими. В общении с согражданами м-р Майер может произносить большинство выражений по английски. Александр Флоренский любезно сделал перевод на то, что он считает английским языком. Хотя этот перевод даже увеличивает примитивистскую мощь митьковских выражений, он заставил меня прибегать к обратному переводу.
>
> Владимир Шинкарев. Митьки

В настоящей главе мы описываем работу с системой `Mathematica` как с продвинутым научным калькулятором. Мы объясняем, как проводятся все обычные вычисления с числами, многочленами, рациональными дробями, элементарными функциями, векторами, матрицами, и т.д. Основные излагаемые в этой главе темы включают:

• Решение всех обычных вычислительных задач школьной математики: арифметические операции, алгебраические преобразования, решение уравнений и неравенств, а также систем уравнений и неравенств, исследование функций.

• Построение графиков функций одной и двух переменных, параметрических графиков и графиков неявных функций, графическое решение неравенств и простейшая двумерная графика.

• Решение всех обычно излагаемых на младших курсах университетов задач так называемой "высшей математики": ряды, произведения и пределы, дифференцирование и интегрирование функций одной и нескольких переменных

• Решение всех обычно излагаемых на младших курсах университетов задач линейной алгебры: решение систем линейных уравнений, умножение и обращение матриц, вычисление определителей, собственных чисел и собственных векторов и т.д.

Примеры подобраны так, чтобы после беглого знакомства с содержанием этой главы — даже не вникая в детали — школьник или студент младших курсов был в состоянии *самостоятельно* решать при помощи системы `Mathematica` **все** встречающиеся ему в курсе математики *вычислительные* задачи.

С другой стороны, мы приложили все усилия к тому, чтобы большинство приводимых нами примеров были математически осмысленными и содержательными. Они призваны либо проиллюстрировать уже знакомые читателю явления, либо познакомить его с такими классическими объектами и фактами, которые с большой вероятностью встретятся ему при проведении самостоятельных вычислений.

В отличие от многих наших коллег, настойчиво изгоняющих из преподавания математики всякую конкретику, мы не видим абсолютно **никакого вреда** в том, чтобы начинающий *как можно раньше* услышал о золотом

сечении, биномиальных коэффициентах, числах Стирлинга, числах Бернулли, числах Фибоначчи, числах Каталана, гармонических числах, многочленах Бернулли, гауссовых многочленах, многочленах Чебышева, циклотомических многочленах, гамма-функции Эйлера, дзета-функции Римана, фигурах Лиссажу, эллиптических кривых, плоскости Фано, кватернионах и октавах, формуле Фаа ди Бруно, функциях Бесселя, функциях Эйри, интегральном логарифме, интегральной экспоненте, интегральных синусе и косинусе, полилогарифме, гипергеометрических функциях, интегралах Френеля, стандартных матричных единицах, матрицах Картана, матрице Гильберта, якобиевых матрицах, определителе Вандермонда, определителе Коши, и *десятках* других важнейших вещей. Все это классические объекты, которые возникают в самой математике и ее приложениях в *сотнях* самых различных контекстов.

Наша собственная позиция прямо противоположна: мы считаем, что знание определителя Вандермонда *гораздо* важнее знакопеременной формулы для определителя в общем случае. Более того, мы выскажем уже совершенно крамольную мысль, что не только для физика, но и для любого математика-неспециалиста, знание полутора десятков классических дифференциальных уравнений, структуры и поведения их решений, гораздо важнее всех теорем существования и единственности решений в общем случае, вместе взятых. Математика — чрезвычайно конкретная наука и любые `общие факты` только тогда становятся *общими* фактами, когда они опираются на глубокое понимание **примеров**, конкретных частных случаев, притом на *такое* понимание, которое может быть в целом и во всех деталях доведено до исчерпывающего ответа и объяснено школьнику шестого класса. Кроме всего прочего, большинство этих конкретных объектов с необходимостью появляются уже на самых первых шагах в компьютерной математике и анализе алгоритмов — и каждый, кто хочет *профессионально* использовать вычисления, должен быть готов к тому, что мир *настоящей* математики намного богаче и увлекательнее того скуд[оум]ного набора общих фраз, к которым сводятся курсы “элементарной” и “высшей” математики.

Несколько слов о том, чего нет в этой главе. Мы не обсуждаем компьютерные доказательства геометрических теорем. Дело в том, что хотя с технической точки зрения это совсем несложно, при этом требуется принципиально другой уровень алгебраической культуры (знакомство хотя бы с рудиментами коммутативной алгебры и алгебраической геометрии, базисами Гребнера, etc.) Кроме того, мы лишь коротко упоминаем ряд важнейших общих тем, традиционно вообще не входящих в курсы математики для нематематиков, таких, как комбинаторика, теория чисел и алгебра (вычисления в группах, кольцах, etc.) и ряд более специальных тем, излагаемых на старших курсах: теория вероятностей, оптимизация, etc. Впрочем, некоторые темы из алгебры, комбинаторики и теории чисел ненавязчиво звучат в нашем учебнике — по крайней мере в той степени, в которой это абсолютно необходимо при обсуждении собственно программистских вопросов.

## § 1. Арифметика

Arithmetic is being able to count up to twenty without taking off your shoes.

Mickey Mouse

Системы компьютерной алгебры обесценивают большинство традиционных вычислительных *навыков*, которым обучают в школьном курсе математики. Все задачи школьной математики, вычисления с целыми, рациональными, вещественными и комплексными числами, алгебраические манипуляции, решение уравнений и неравенств, а также систем уравнений и неравенств, построение графиков функций, тригонометрические преобразования, геометрические построения выполняются этими системами за тысячные доли секунды, а доказательство теорем элементарной геометрии — за несколько секунд.

• **Вычисления с целыми числами.** На самом примитивном уровне `Mathematica` является очень мощным калькулятором, работающим с числами неограниченной разрядности, притом работающим с ними гораздо эффективнее, чем подавляющее большинство специализированных численных приложений!!! Сейчас мы предложим `Mathematica` вычислить $180(2^{127}-1)^2+1$ (это самое большое простое число известное к началу 1952 года, оно было открыто Миллером и Уиллером на компьютере `EDSAC1`):

```
In[1]:=180*(2^127-1)^2+1
Out[1]=5210644015679228794060694325390955853335 89
                  84839080564583521838510183725557352 21
```

В арифметических вычислениях знаки `+`, `-` и `/` имеют обычный смысл, однако обратите внимание на использование `*` и `^` для обозначения умножения и возведения в степень. Фактически разрядность арифметических вычислений лимитируется *только* объемом памяти, так как даже на пределе этого объема вычисления занимают доли секунды. Например, возведение 2 в степень 1 000 000 000 — дающее число с 301 029 996 знаками — еще в начале 2000-х занимало на нашем компьютере лишь четверть секунды.

А вот еще один очень поучительный диалог, в котором мы спрашиваем, равны ли два числа и получаем ответ, что они действительно равны:

```
In[2]:=2682440^4+15365639^4+18796760^4==20615673^4
Out[2]=True
```

Иными словами, действительно

$$2682440^4 + 15365639^4 + 18796760^4 = 20615673^4.$$

Эти четыре числа, найденные в 1988 году Ноамом Элкисом[13], представляют собой контрпример к гипотезе Эйлера, который по аналогии с гипотезой

[13] N.D.Elkies, On $A^4 + B^4 + C^4 = D^4$. — Math. Comput., 1988, vol.51, p.825–835.

Ферма предположил, что уравнение $x^4 + y^4 + z^4 = w^4$ не имеет решений в ненулевых целых числах. Обратите внимание на использование двойного знака равенства `==`, который трактуется как уравнение, в данном случае вопрос о том, равна левая часть правой, или нет. `A parte:` довольно удивительно, что Эйлер задал столь наивный вопрос, ведь ему было известно не только наименьшее решение уравнения $x^4+y^4 = z^4+w^4$ в целых числах:

$$59^4 + 158^4 = 635318657 = 133^4 + 134^4,$$

но и параметрические семейства таких решений. Интересно, что уравнение $u^5 + v^5 + x^5 + y^5 = w^5$ тоже имеет решение в совсем маленьких числах:

$$27^5 + 84^5 + 110^5 + 133^5 = 61917364224 = 144^5,$$

хотя, по-видимому, найти вручную даже такое совсем маленькое решение довольно трудно.

• **Вычисления с рациональными числами.** Разумеется, вычисления с рациональными числами ничуть не сложнее вычислений с целыми числами — в действительности, это и есть вычисления с парами целых чисел. Вот, например, таблица первых 20 чисел Бернулли $B_n$[14] :

```
In[3]:=Table[BernoulliB[n],{n,1,20}]
Out[3]={-1/2,1/6,0,-1/30,0,1/42,0,-1/30,0,5/66,0,-691/2730,0,
        7/6,0,-3617/510,0,43867/798,0,-174611/330}
```

А вот их сумма и произведение чисел с четными номерами

```
In[4]:=Sum[BernoulliB[n],{n,1,20}]
Out[4]=-932396477/1939938
In[5]:=Product[BernoulliB[n],{n,2,20,2}]
Out[5]=-19144150084038739/940848823474560000
```

Обратите внимание на естественные названия операций: таблица называется `Table`, сумма — `Sum`, произведение — `Product`. При некотором навыке и небольшом знании английского языка в большинстве случаев Вы будете в состоянии *угадать* название любой команды `Mathematica` с одной, двух, максимум трех попыток. Если команда, осуществляющая интегрирование, не называется `Integral`, то она просто обязана называться `Integrate`. Обратите внимание также на естественную форму итератора: `{n,1,20}` — сумма по $n$ от 1 до 20; `{n,2,20,2}` — произведение по $n$ от 2 до 20 с шагом 2. Эта форма знакома Вам из языка `C`. Для разнообразия `Maple` использует форму итератора, принятую в `Pascal`, а именно, `n=1..20`.

---

[14] Здесь и далее в нашем учебнике результаты расчета `Out[*]` представляются в виде `InputForm` - таком же как и `In[*]`. Это удобно для использования результата "как есть" в последующих преобразованиях. В пакете `Mathematica` изменение вида представления содержимого ячейки доступно пользователю через через меню `Cell-Convert To...` (см. § 3) или через контекстное меню - щелчок правой кнопкой мыши в пределах ячейки.

• **Вычисления с вещественными числами.** Особенностью системы `Mathematica` по сравнению с обычными системами численных вычислений является использование вычислений *бесконечной точности*. Это значит, что никаких округлений при вычислениях с точными вещественными числами такими, как $\sqrt{2}$, $e$ и $\pi$ — или, на языке `Mathematica`, `Sqrt[2]`, `E`, `Pi` — не производится. Если нам нужно десятичное приближение числа $x$ с точностью до $n$ значащих цифр, мы можем просто спросить `N[x,n]`. При помощи `Mathematica` мы можем проделать любое вычисление с вещественными числами, которое можно было бы проделать при помощи научного калькулятора с неограниченным количеством разрядов — и много более того. У многих школьников, когда они впервые знакомятся с числами $\pi$ и $e$, возникает вопрос, что больше, $e^\pi$ или $\pi^e$? Как мы только что объяснили, попытка непосредственно вычислить `E^Pi` и `Pi^E` и посмотреть на то, что получится, не приведет к успеху, так как `Mathematica` трактует эти числа как *точные* вещественные числа, а какие там у точных чисел десятичные знаки? Однако `Mathematica` умеет сравнивать точные вещественные числа. Ввод `E^Pi>Pi^E` дает значение `True`. Конечно, мы можем посмотреть и на любое количество десятичных разрядов, ну, скажем, для начала на 20:

```
In[6]:={N[E^Pi,20],N[Pi^E,20]}
Out[6]={23.140692632779269006,22.459157718361045473}
```

Мы оформили это вычисление в виде списка, так как хотели увидеть оба ответа вместе.

Бесконечная точность совершенно необходима для того, чтобы контролировать явление, известное в компьютерной алгебре как `high-precision fraud`[15]. Дело в том, что недостаточная точность часто приводит к появлению **артефактов**[16]. Известно много поразительных примеров, когда значения двух естественно заданных функций совпадают с точностью до тысяч и даже миллионов знаков после запятой, так что никакие приближенные вычисления не позволяют сказать, равны эти значения или нет. Вот один из самых знаменитых примеров — число $e^{\pi\sqrt{163}}$, *настолько* близко к целому, что даже вычисление 12 знаков после запятой не позволяет сказать, что это число не целое:

```
In[7]:=NumberForm[N[Exp[Pi*Sqrt[163]],30],
                  ExponentFunction->(Null&)]
Out[7]=262537412640768743.999999999999
```

Для профессионалов заметим, что этот факт ошеломителен лишь на первый взгляд, после секундного раздумья каждый компетентный математик

---

[15] J.M.Borwein, P.B. Borwein, Strange series and high precision fraud. — Amer. Math. Monthly, 1992, vol.99, N.7, p.622–640.

[16] `Артефактом` в компьютерной алгебре называется ошибочный результат или наблюдаемое явление, не отвечающее существу рассматриваемой задачи, порожденные недостаточной точностью вычисления и/или использованием в процессе вычисления плохих алгоритмов (ошибочных, неустойчивых или очень медленно сходящихся).

должен увидеть его связь с тем фактом, что кольцо $\mathbb{Z}[\sqrt{-163}]$ является кольцом главных идеалов!!! И действительно, числа $e^{\pi\sqrt{67}}$ и $e^{\pi\sqrt{43}}$ тоже чрезвычайно близки к целым, хотя, конечно, и не с такой изумительной точностью. Мы уже знаем, что `Sqrt[163]` обозначает $\sqrt{163}$, где `Sqrt` — стандартное сокращение от `Square Root`, принятое в большинстве языков программирования, `Exp[x]`, естественно, обозначает экспоненту $e^x$, а функция `N[x,30]` показывает нам первые тридцать десятичных знаков числа $x$. Использование функции `NumberForm` и опции `ExponentFunction->(Null&)` нужно только для того, чтобы `Mathematica` не пыталась выражать эти числа в научной форме, с разделением мантиссы и порядка. Мы могли бы с таким же успехом напечатать просто `N[Exp[Pi*Sqrt[163]],30]`, но тогда, конечно, ответ имел бы менее наглядную форму,

$$2.62537412640768743999999999999 \times 10^{17},$$

в которой нужно было бы еще мысленно сдвинуть десятичную точку на 17 позиций вправо. Однако небольшое дальнейшее увеличение точности показывает, что это число, все-таки, не целое:

```
In[8]:=NumberForm[N[Exp[Pi*Sqrt[163]],40],ExponentFunction->(Null&)]
Out[8]=262537412640768743.999999999992500725972
```

В приведенных выше примерах мы ограничивались небольшим количеством десятичных знаков. В действительности `Mathematica` может работать с сотнями тысяч или миллионами десятичных знаков. Время, нужное для вычисления первого миллиона знаков $\pi$, $e$ и $\sqrt{2}$ или первых ста тысяч знаков $e^\pi$, $\pi^e$ и $\sqrt{2}^{\sqrt{3}}$, исчисляется несколькими секундами или, в худшем случае (профессиональным вычислителям предлагается угадать, какой случай худший!), десятком секунд. Но не пытайтесь выводить все эти знаки на экран, так как форматирование вывода занимает гораздо больше времени, чем само вычисление!!! Таким образом, единственным реальным ограничением для дальнейшего увеличения разрядности становится оперативная память используемого Вами компьютера.

• **Вычисления с комплексными числами.** Ясно, что вычисления с комплексными числами ничуть не сложнее — а на самом деле часто проще! — чем с вещественными. Комплексное число $z \in \mathbb{C}$ можно задавать, например, в алгебраической форме $z = x + iy$, где $x, y \in \mathbb{R}$ суть его вещественная и мнимая части, соответственно, а $i$ — мнимая единица, $i^2 = 1$. Нас не должно удивлять, что в `Mathematica` следует печатать `z=x+I*y` — мы уже привыкли, что имена всех внутренних объектов начинаются с заглавной буквы. Если `Mathematica` уверена, что $x$ и $y$ вещественные числа (например, если она *знает*, что это вещественные числа или если мы специфицировали домен, которому принадлежат $x$ и $y$, и этот домен содержится в домене вещественных чисел), то она истолковывает их как вещественную и мнимую часть $z$:

```
In[9]:=z=E+I*Pi; {Re[z],Im[z],Abs[z],Arg[z],Conjugate[z]}
```

Out[9]={E, Pi, $\sqrt{\text{E\^{}2+Pi\^{}2}}$, ArcTan[Pi/E], E+I*Pi}

Снова мы видим, что в Mathematica все функции называются так, как они на самом деле называются:

- Re[z] — вещественная часть $z$,
- Im[z] — мнимая часть $z$,
- Abs[z] — модуль $z$ (абсолютная величина),
- Arg[z] — аргумент $z$,
- Conjugate[z] — сопряженное к $z$ комплексное число,

и т.д. Именно высочайшая степень предсказуемости и согласованность языка системы Mathematica с обычным математическим языком облегчают ее использование по сравнению со всеми остальными системами сопоставимой силы и относятся к числу ее главных достоинств. Ясно, однако, что Mathematica ориентируется на англоязычную терминологию и типографские традиции. Так, например, в этом примере мы видим, что $\mathrm{arctg}(x)$ обозначается через $\arctan(x)$ или, на языке системы ArcTan[x] — каждый корень, входящий в состав многосложного слова, пишется с заглавной буквы.

В школьной математике принято заучивать значения основных тригонометрических функций углов $60^o$, $45^o$ и $30^o$. Между тем есть еще один столь же замечательный угол, а именно, $36^o$, значения основных тригонометрических функций в котором являются квадратичными иррациональностями. Иными словами, как знали еще древние греки, круг можно разделить на 5 — или, что то же самое, на 10 — равных частей при помощи циркуля и линейки. Определим комплексное число $\eta$, вещественная часть которого равна $\cos(\pi/5)$, а мнимая — $\sin(\pi/5)$:

```
In[10]:=eta=Cos[Pi/5]+I*Sin[Pi/5]
```

$$\text{Out[10]=}\frac{1}{2}\text{*I*}\sqrt{\frac{1}{2}\text{*(5-}\sqrt{5}\text{)}}+\frac{1}{4}\text{*(1+}\sqrt{5}\text{)}$$

Попробовав вычислить $\eta^2$ мы не получим ничего интересного:

```
In[11]:=eta^2
```

$$\text{Out[11]=}\left(\frac{1}{2}\text{*I*}\sqrt{\frac{1}{2}\text{*(5-}\sqrt{5}\text{)}}+\frac{1}{4}\text{*(1+}\sqrt{5}\text{)}\right)^2$$

Дело в том, что — и мы многократно столкнемся с этим в дальнейшем! — ни одна интеллигентная система компьютерной алгебры не проводит преобразований, если она не уверена, что их применение упростит форму исходного выражения. В частности, она автоматически не применяет дистрибутивность ни в ту ни в другую сторону или, пользуясь школьным жаргоном, не раскрывает скобок и не выносит общие множители. В тот момент, когда Вы поймете, *почему* системы компьютерной алгебры устроены таким образом, Вы сделаете решающий шаг к пониманию того, как добиться от системы ответа на любой вопрос в интересующей Вас форме. В частности, для комплексных чисел — и комплексных функций! – в системе имеется функция ComplexExpand, которая заставляет искать вещественную и мнимую часть. Поэтому если Вы хотите увидеть $\eta^2$ в алгебраической форме, нужно просить чуть настойчивее:

In[12]:=`ComplexExpand[eta^2]`

Out[12]=$-\frac{1}{4}+\frac{\sqrt{5}}{4}$+I*$\left(\frac{1}{4}*\sqrt{\frac{1}{2}*(5-\sqrt{5})}+\frac{1}{4}*\sqrt{\frac{5}{2}*(5-\sqrt{5})}\right)$

Теперь Вас, вероятно, уже не удивит, что спросив `ComplexExpand[eta^5]` мы получим `-1`.

## § 2. Многочлены и рациональные дроби

> Чуда не вижу я тут. Генерал-лейтенант Захаржевский
> В урне той дно просверлив, воду провел чрез нее.
>
> Алексей Константинович Толстой, Царскосельская статуя

Каждый, кто заглядывал в ее внутреннее устройство, знает, что сердцевину системы `Mathematica` составляют вычисления с многочленами от нескольких переменных. Здесь `Mathematica` в своей стихии и заведомо превосходит остальные системы общего назначения — `nobody can beat me in the kitchen`. Начнем с чего-нибудь совсем простенького.

• **Вычисления с многочленами.** Любая *новая* переменная, которой до сих пор не присваивалось значения, рассматривается системой как **независимая переменная**. Например, если мы до сих пор не присваивали значений $x, y, z, u, v, w$, то мы можем использовать их как независимые полиномиальные переменные. В действительности, конечно, в предыдущем параграфе мы уже присвоили $z$ значение, так что если мы хотим снова использовать ее как переменную, мы должны удалить ее значения и определение посредством `ClearAll[z]` или, может быть, даже посредством радикального `Remove[z]`. Если мы этого не сделаем, система будет подставлять вместо $z$ ее старое значение `E+I*Pi`, а это, видимо, совсем не то, что мы хотели. Самая страшная тайна компьютерной алгебры состоит в том, что использование старых значений переменных является основным источником ошибок!!! Если нам не хватает букв латинского алфавита, мы можем использовать в качестве имени переменной любое слово, скажем, `xx`, `xxx`, `xy`, `strength`, `dexterity`, `constitution`, `wisdom`, `intelligence`, `charisma`, `hitpoints`, `armourclass`, `experience`, `bandwidth`, и т.д. Иногда, например, если Вы обзовете переменную `length`, `depth` или `power`, система будет *слегка встревожена* (`slightly alarmed`), так как будет считать, что Вы хотели обратиться к какой-то из ее встроенных функций, но сделали опечатку, набрав строчную букву вместо заглавной. Однако после того, как Вы один раз проигнорируете ее жалобу, она внесет это новое имя в глобальный контекст и в дальнейшем уже не будет проявлять по этому поводу никаких признаков беспокойства. Основное правило грамотного программирования состоит в том, чтобы присваивать локальным и часто встречающимся переменным короткие имена, а глобальным и редко встречающимся переменным — длинные имена. Если Вам не хватает букв, Вы можете использовать цифры, например, `x1,x2,x3,x4` могут обозначать $x_1, x_2, x_3, x_4$. Единственное правило, которого при этом

нужно придерживаться, состоит в том, что имя переменной не может *начинаться* с цифры. Однако имейте в виду, что если Вам нужно несколько десятков, сотен или тысяч однородных переменных, то их следует оформлять в виде массива `Array` или списка `List`, `Table` — мы сами *обычно* так и поступаем уже с тремя или четырьмя переменными!

С многочленами можно проделывать все обычные операции:

◦ арифметические операции,

◦ деление с остатком `PolynomialQuotient`, `PolynomialRemainder`, `PolynomialMod`, `PolynomialReduce`,

◦ композицию (подстановку многочлена в многочлен),

◦ отыскание наибольшего общего делителя `PolynomialGCD` и наименьшего общего кратного `PolynomialLCM`,

и т.д. Кроме того, с многочленами можно проделывать различные структурные манипуляции такие как

◦ разложение по степеням какой-то из переменных `Expand`,

◦ разложение на множители `Factor`,

и т.д. Приведем несколько совсем простых примеров.

• **Операции над многочленами.** Проиллюстрируем операции над многочленами на примере **круговых многочленов** $\Phi_n(x)$. Напомним, что

$$\Phi_n(x) = \prod (x - \varepsilon_i),$$

где произведение берется по всем **первообразным** корням $\varepsilon_i$ из 1 степени $n$. Эти многочлены естественно возникают в сотнях различных вычислений, как в самой алгебре и теории чисел, так и в многочисленных приложениях, связанных с теорией конечных полей, включая `Computer Science`, теорию передачи информации и криптографию. В `Mathematica` $n$-й круговой многочлен $\Phi_n(x)$ называется `Cyclotomic[n,x]`. Взглянем на несколько — для круглого счета 16 — первых круговых многочленов:

```
In[13]:=Do[Print[Cyclotomic[n,x]],{n,0,15}]
Out[13]=1
        -1+x
        1+x
        1+x+x^2
        1+x^2
        1+x+x^2+x^3+x^4
        1-x+x^2
        1+x+x^2+x^3+x^4+x^5+x^6
        1+x^4
        1+x^3+x^6
        1-x+x^2-x^3+x^4
```

```
1+x+x^2+x^3+x^4+x^5+x^6+x^7+x^8+x^9+x^10
1-x^2+x^4
1+x+x^2+x^3+x^4+x^5+x^6+x^7+x^8+x^9+x^10+x^11+x^12
1-x+x^2-x^3+x^4-x^5+x^6
1-x+x^3-x^4+x^5-x^7+x^8
```

Команда `Print` выводит результат каждого индивидуального вычисления в отдельной строке — иными словами, вставляет `\newline` после каждого выражения. Она применяется, если мы хотим посмотреть на результаты нескольких вычислений в одной клетке, но не хотим при этом явно структурировать их в виде списка. Обратите внимание на организацию цикла при помощи команды `Do` с обычной в языке `C` формой итератора `{n,0,15}` — по $n$ от 0 до 15 (с шагом 1). Мы могли бы достичь того же результата следующим образом: вначале *породить список* циклотомических многочленов при помощи обсуждаемой в § 10 команды `Table`, а потом **проскан(д)ировать** (`Scan`) элементы этого списка командой `Print`:

```
In[14]:=Scan[Print,Table[Cyclotomic[n,x],{n,0,15}]]
```

Попробуем теперь перемножить два круговых многочлена. Беззастенчивое вычисление `Cyclotomic[5,x]*Cyclotomic[6,x]` ничего не даст, так как `Mathematica` НЕ РАСКРЫВАЕТ СКОБОК АВТОМАТИЧЕСКИ. Правильная форма вопроса, если мы хотим получить ответ, разложенный по степеням $x$, должна включать структурную команду `Expand`:

```
In[15]:=Expand[Cyclotomic[5,x]*Cyclotomic[6,x]]
Out[15]=1+x^2+x^3+x^4+x^6
```

Деление многочленов от одной переменной осуществляется при помощи команд `PolynomialQuotient` — неполное частное и `PolynomialRemainder` — остаток. Остаток можно найти и при помощи более общих команд `PolynomialMod` или `PolynomialReduce`, предназначенных для нахождения остатка многочленов от нескольких переменных по модулю нескольких многочленов. Вот пример деления с остатком:

```
In[16]:=PolynomialQuotient[Cyclotomic[11,x],Cyclotomic[5,x],x]
Out[16]=x+x^6
In[17]:=PolynomialRemainder[Cyclotomic[11,x],Cyclotomic[5,x],x]
Out[17]=1
```

Обратите внимание на синтаксис: команды деления многочленов от одной переменной `PolynomialQuotient` и `PolynomialRemainder` вызываются с *тремя* аргументами, в формате `PolynomialQuotient[f,g,x]`, где $f$ — делимое, $g$ — делитель, а $x$ — переменная, по которой производится деление.

• **Структурные манипуляции.** Из школы все помнят формулу для $(x+y)^n$ известную под кодовым названием “бином Ньютона”. Но кто, кроме профессиональных математиков, видел **мультиномиальную формулу Лейбница** для $(x_1 + \ldots + x_m)^n$, за исключением, может быть, случая $(x + y + z)^3$? Вот один из первых интересных примеров. Как мы уже знаем,

попытка вычислить `(w+x+y+z)^4` не приведет к успеху, так как `Mathematica` автоматически не раскрывает скобок. Это значит, что мы должны попросить ее сделать это:

```
In[18]:=Expand[(w+x+y+z)^4]
Out[18]=w^4+4*w^3*x+6*w^2*x^2+4*w*x^3+x^4+
                4*w^3*y+12*w^2*x*y+12*w*x^2*y+4*x^3*y+6*w^2*y^2+
                12*w*x*y^2+6*x^2*y^2+4*w*y^3+4*x*y^3+y^4+
                4*w^3*z+12*w^2*x*z+12*w*x^2*z+4*x^3*z+
                12*w^2*y*z+24*w*x*y*z+12*x^2*y*z+12*w*y^2*z+
                12*x*y^2*z+4*y^3*z+6*w^2*z^2+12*w*x*z^2+
                6*x^2*z^2+12*w*y*z^2+12*x*y*z^2+6*y^2*z^2+
                4*w*z^3+4*x*z^3+4*y*z^3+z^4
```

А сейчас мы предложим системе проделать эту операцию в обратную сторону. Из школы все помнят "формулы сокращенного умножения" $x^2 - y^2 = (x-y)(x+y)$ и $x^3 - y^3 = (x-y)(x^2+xy+y^2)$, но как раскладывается на множители что-нибудь чуть большей степени, ну хотя бы $x^{100} - y^{100}$? Нет ничего проще:

```
In[19]:=Factor[x^100-y^100]
Out[19]=(x-y)(x+y)(x^2+y^2)
                (x^4-x^3*y+x^2*y^2-x*y^3+y^4)
                (x^4+x^3*y+x^2*y^2+x*y^3+y^4)
                (x^8-x^6*y^2+x^4*y^4-x^2*y^6+y^8)
                (x^20-x^15*y^5+x^10*y^10-x^5*y^15+y^20)
                (x^20+x^15*y^5+x^10*y^10+x^5*y^15+y^20)
                (x^40-x^30*y^10+x^20*y^20-x^10*y^30+y^40)
```

• **Многочлены от нескольких переменных.** Эйлер заметил, что произведение двух сумм четырех квадратов снова является суммой четырех квадратов:

$$(x_1^2+x_2^2+x_3^2+x_4^2)(y_1^2+y_2^2+y_3^2+y_4^2) =$$
$$(x_1y_1-x_2y_2-x_3y_3-x_4y_4)^2+(x_1y_2+x_2y_1+x_3y_4-x_4y_3)^2+$$
$$(x_1y_3+x_3y_1-x_2y_4+x_4y_2)^2+(x_1x_4+x_4y_1+x_2y_3-x_3y_2)^2$$

В 1842 году Гамильтон осознал, что это замечательное тождество, известное как **тождество Эйлера**, можно принять за определение умножения четверок вещественных чисел, которое превращает $\mathbb{R}^4$ в алгебру с делением $\mathbb{H}$, известную как **тело кватернионов**. Попробуем проверить тождество Эйлера с помощью системы `Mathematica`. Для этого совершенно бесхитростно предложим ей провести следующее вычисление:

```
In[20]:=z1=x1*y1-x2*y2-x3*y3-x4*y4; z2=x1*y2+x2*y1+x3*y4-x4*y3;
        z3=x1*y3+x3*y1-x2*y4+x4*y2; z4=x1*y4+x4*y1+x2*y3-x3*y2;
                z1^2+z2^2+z3^2+z4^2
```

```
Out[20]=(x4*y1-x3*y2+x2*y3+x1*y4)^2+(x3*y1+x4*y2+x1*y3-x2*y4)^2+
        (x2*y1+x1*y2-x4*y3+x3*y4)^2+(x1*y1-x2*y2-x3*y3-x4*y4)^2
```

Постараемся понять, что произошло. Во-первых, мы определили новые переменные $z_1, z_2, z_3, z_4$, стоящие в скобках в правой части тождества Эйлера, и задали их выражение через исходные переменные $x_1, x_2, x_3, x_4$ и $y_1, y_2, y_3, y_4$. После этого мы предложили системе *вычислить* $z_1^2+z_2^2+z_3^2+z_4^2$, но она просто подставила сюда выражения $z_i$ через $x_i$ и $y_i$. Мы уже встречались с этим явлением. Дело в том, что система не уверена, приведет ли раскрытие скобок к более короткому выражению, и ждет нашего явного указания сделать это. Обратите внимание, что различные выражения внутри одного ввода должны разделяться точкой с запятой. Конечно, мы можем заставить систему раскрыть скобки при помощи команды `Expand`, но тогда она оставит результат в виде суммы одночленов. После этого мы можем применить к этой сумме одночленов команду `Factor` и она попробует разложить их на целочисленные множители. Однако проще всего довериться ее собственному эстетическому чувству и посредством какой-либо из команд `Simplify`, `FullSimplify` или `Refine` предложить ей *упростить* выражение $z_1^2 + z_2^2 + z_3^2 + z_4^2$. В этом случае она будет пытаться применить к этому выражению известные ей преобразования и искать среди получающихся результатов самый простой. Первая же попытка приводит к тождеству Эйлера:

```
In[21]:=Simplify[z1^2+z2^2+z3^2+z4^2]
Out[21]=(x1^2+x2^2+x3^2+x4^2)*(y1^2+y2^2+y3^2+y4^2)
```

Вот еще одно замечательное тождество, **тождество Лиувилля**, которое было одним из основных шагов в доказательстве того, что каждое натуральное число является суммой 53 четвертых степеней:

```
In[22]:=Factor[Expand[(x+y)^4+(x+z)^4+(x+w)^4+(y+z)^4+
                      (y+w)^4+(z+w)^4+(x-y)^4+(x-z)^4+
                      (x-w)^4+(y-z)^4+(y-w)^4+(z-w)^4]]
Out[22]=6(w^2+x^2+y^2+z^2)^2
```

• **Многочлены Чебышева.** В школьном курсе тригонометрии встречаются формулы $\cos(2\phi) = 2\cos(x)^2 - 1$ и $\cos(3\phi) = 4\cos(x)^3 - 3\cos(\phi)$. Одним из самых важных классических объектов математики являются **многочлены Чебышева** первого рода $T_n$, при помощи которых $\cos(n\phi)$ выражается через $\cos(\phi)$. По определению, $\cos(n\phi) = T_n(\cos(\phi))$. Таким образом, $T_0(x) = 1$, $T_1(x) = x$ и, как мы только что вспомнили, $T_2(x) = 2x^2 - 1$ и $T_3(x) = 4x^3 - 3x$. Естественно, многочлены Чебышева описаны в ядре системы `Mathematica` и, по аналогии с `BernoulliB`, мы можем ожидать, что они называются `ChebyshevT`. Посмотрим на многочлены Чебышева $T_5(x)$ и $T_7(x)$, чтобы немного уменьшить количество печатаемых символов, введем для `ChebyshevT[x,n]` сокращение `T[x,n]`:

```
In[23]:=T[n_,x_]:=ChebyshevT[n,x]; {T[5,x],T[7,x]}
Out[23]={5*x-20*x^3+16*x^5,7*x+56*x^3-112*x^5+64*x^7}
```

В этом тексте встречаются несколько ключевых синтаксических моментов, которые подробнейшим образом обсуждаются в дальнейшем, тем не менее сразу поясним их. В первой строчке мы определяем `T[n,x]`, при этом **бланк** _ после неизвестной означает, что это **фиктивная неизвестная**, вместо которой можно подставить произвольное значение или любую другую неизвестную. Если вместо `T[n_,x_]` мы написали бы просто `T[n,x]`, система знала бы, чему равно `T[n,x]`, но не чему равно `T[m,y]`. С другой стороны, отложенное присваивание `:=` означает, что правая часть `ChebyshevT[n,x]` вычисляется не в тот момент, когда выполнено присваивание, а каждый раз при вызове очережного значения `T[n,x]`. После этого мы видим уже знакомую нам точку с запятой `;` разделяющую выражения или команды. После этого мы вызываем **список** {`T[5,x],T[7,x]`} состоящий из многочленов Чебышева $T_5(x)$ и $T_7(x)$. Мы не могли бы напечатать просто `T[5,x]; T[7,x]`, так как при этом увидели бы только многочлен $T_7(x)$. Подставим теперь $T_7(x)$ в $T_5(x)$:

```
In[24]:=T[5,T[7,x]]

Out[24]=5*(-7*x+56*x^3-112*x^5+64*x^7)-
                  20*(-7*x+56*x^3-112*x^5+64*x^7)^3+
                  16*(-7*x+56*x^3-112*x^5+64*x^7)^5
```

Как всегда, `Mathematica` не раскрывает скобки! Ну что же, попросим ее сделать это:

```
In[25]:=Expand[T[5,T[7,x]]]

Out[25]=-35*x+7140*x^3-434112*x^5+12403200*x^7-
        202585600*x^9+2106890240*x^11-14910300160*x^13+
        74977509376*x^15-275652608000*x^17+754417664000*x^19-
        1551944908800*x^21+2404594483200*x^23-2789329600512*x^25+
        2384042393600*x^27-1456262348800*x^29+601295421440*x^31-
        150323855360*x^33+17179869184*x^35
```

Если то, что получилось, удивительно похоже на многочлен Чебышева $T_{35}(x)$, то это потому, что это и есть многочлен Чебышева $T_{35}(x)$ — IT LOOKS LIKE A CHURCH, IT SMELLS LIKE A CHURCH, IT IS A CHURCH. Иными словами, если мы подставим в только что полученное выражение $\cos(\phi)$ вместо $x$, то мы получим $\cos(35\phi)$. Теперь Вас уже наверное, не особенно удивит, если и `Expand[T[7,T[5,x]]]` даст тот же результат.

В этом можно убедиться также применив к многочлену `T[35,x]` функцию `Decompose`, раскладывающую многочлен в композицию неразложимых многочленов:

```
In[26]:=Decompose[T[35,x],x]

Out[26]={7*x-56*x^3+112*x^5-64*x^7,-5*x+20*x^3-16*x^5}
```

Вообще то, в данном случае нам просто крупно повезло. Попробуйте образовать композицию $T_2$ и $T_3$, а потом разложить ее при помощи `Decompose` и посмотрите, что получится!

Вообще, $T_m(T_n(x)) = T_n(T_m(x))$ причем в смысле, который легко уточнить[17,18,19,20], это единственная *нетривиальная* система многочленов над $\mathbb{C}$ с таким свойством. Только не говорите нам про $f_n(x) = x^n$, мы же сказали, *нетривиальная*!

• **Вычисления с рациональными дробями.** Известно, что любое рациональное число представимо в виде суммы трех кубов рациональных чисел. Это вытекает, например, из следующего тождества, независимо открытого в 1825 году Райли и в 1930 году Ричмондом[21]:

$$x = \left(\frac{x^3 - 3^6}{3^2x^2 + 3^4x + 3^6}\right)^3 + \left(\frac{-x^3 + 3^5x + 3^6}{3^2x^2 + 3^4x + 3^6}\right)^3 + \left(\frac{3^3x^2 + 3^5x}{3^2x^2 + 3^4x + 3^6}\right)^3$$

Попытка доказать это тождество просто напечатав

```
In[27]:=((x^3-3^6)/(3^2x^2+3^4x+3^6))^3+
        ((-x^3+3^5x+3^6)/(3^2x^2+3^4x+3^6))^3+
        ((3^3x^2+3^5x)/(3^2x^2+3^4x+3^6))^3
```

не приведет к успеху, так как `Mathematica` автоматически не раскрывает скобок. Однако в `Mathematica` имеется несколько функций таких, как `Expand`, `ExpandNumerator`, `ExpandDenominator`, `ExpandAll`, `Factor`, `Cancel`, `Together`, `Apart` и т.д., которые позволяют проделывать все обычные структурные манипуляции над дробями. Попробовав, например, привести эти дроби к общему знаменателю

```
In[28]:=Together[((x^3-3^6)/(3^2x^2+3^4x+3^6))^3+
        ((-x^3+3^5x+3^6)/(3^2x^2+3^4x+3^6))^3+
        ((3^3x^2+3^5x)/(3^2x^2+3^4x+3^6))^3]
```

мы сразу получим ответ $x$. Разумеется, к тому же результату приведет и упрощение этого выражения при помощи `Simplify`, `FullSimplify` или `Refine`. Интересно, что хотя это вычисление и может быть проведено человеком — оно и было первые два раза проведено человеком!!! — тем не менее, вряд ли многие математики захотят проводить подобное вычисление по своей воле без какой-то великой цели. Косвенным подтверждением этого является тот факт, что в книге[22], специально посвященной суммам кубов, эта формула приведена с ошибкой (в последнем слагаемом пропущен множитель $3^3$ при $x^2$, а в качестве множителя при $x$ там же напечатано

---

[17] J.F.Ritt, Prime and composite polynomials. — Trans. Amer. Math., Soc., 1922, vol.23, p.51–66.

[18] J.F.Ritt, Permutable rational functions. — Trans. Amer. Math., Soc., 1923, vol.25, p.399–448.

[19] В.О.Бугаенко, Коммутирующие многочлены. — Математическое Просвещение, Сер.3, N.1, с.140–163.

[20] В.В.Прасолов, О.В,Шварцман, Азбука римановых поверхностей. — М., Фазис, 1999, с.1–142; стр.75–85.

[21] L.E.Dickson, History of the theory of numbers, vol.II. — Chelsea, 1952, p.1–802.

[22] Ю.И.Манин, Кубические формы. —М., Наука, 1972, с.1–304.

$3^4$ вместо $3^5$), причем эта ошибка воспроизведена в обзоре[23]. Тем самым, авторы сами эту формулу не проверяли, что, впрочем, неудивительно, если учесть, что при этом вычислении получаются коэффициенты наподобие 387420489, 129140163 или 28697814.

## § 3. Алгебраические уравнения

Alles vergangliches ist nur ein Gleichnis. — Все преходящее есть всего лишь уравнение[24].

Johann Wolfgang von Goethe, Faust I

Решение уравнений и неравенств, а также систем уравнений и неравенств является `sanctum sanctorum` школьной математики. `Mathematica` может успешно справиться с **любой** задачей, с которой могут справиться школьник, школьный учитель математики, преподаватель педвуза и любой репетитор над репетиторами.

• **Алгебраические уравнения от одной неизвестной.** Основной командой для решения алгебраических и сводящихся к ним уравнений в системе `Mathematica` является `Solve`. Эта команда допускает вызов в разных форматах и настройку большого числа опций, а ее реализация занимает около 500 страниц кода, поэтому здесь мы изложим только простейшие примеры ее использования. В случае решения одного уравнения $f(x) = g(x)$ относительно одной неизвестной $x$ команда `Solve` вызывается в следующем формате

```
Solve[f[x]==g[x],x].
```

Вот простейший пример использования этой команды.

```
In[29]:=Solve[Sqrt[x^2+1]==x+2,x]
Out[29]={{x->-3/4}}
```

Как всегда, начнем с обсуждения синтаксиса:

◦ Обратите внимание на то, что при записи уравнения всегда используется только предикат `== Equal`, но ни в коем случае не предикат `=== SameQ` и, тем более, не оператор `= Set`!!! Вычисление

```
In[30]:=Solve[Sqrt[x^2+1]===x+2,x]
```

даст ответ {}. В самом деле, спрашивая `Solve[f[x]==g[x],x]` мы интересуемся, при каких $x$ *значения* функций $f$ и $g$ в точке $x$ совпадают. В то же время спрашивая `Solve[f[x]===g[x],x]` мы интересуемся, при каких $x$ *внутренние представления* функций $f$ и $g$ совпадают в языке системы. Ну и при каких $x$, по Вашему `Sqrt[x^2+1]` и `x+2` могут совпадать

[23] Ю.И.Манин, А.А.Панчишкин, Введение в теорию чисел. — Соврем. Проблема Математики, т.49, М., ВИНИТИ, 1990, с.1–348; стр.191.

[24] В кольце $\mathbb{Z}/m\mathbb{Z}$. *Как всегда*, канонический русский перевод “Все преходящее есть только символ” полностью извращает смысл сказанного. На самом деле `Gleichnis` может быть чем угодно: `сравнением`, `подобием`, `несовершенной тенью`, `иносказанием` — но никак не `символом`. Наоборот, `символ` есть как то, `как бы проекцией` чего является `Gleichnis`.

*как выражения*? Да ни при каких! С другой стороны, попытка вычислить `Solve[Sqrt[x^2+1]=x+2,x]` сразу приведет к сообщению об ошибке, так как здесь мы пытаемся присвоить новое значение внутренней (защищенной!!!) функции `Sqrt`. Итак, уравнение всегда записывается в виде `f[x]==g[x]`.

◦ В школьной математике существует устойчивая традиция обозначать **параметры** буквами $a, b, c, d, \ldots$, а **неизвестные** буквами $x, y, z, u, v, w$. Однако было бы нелепо ожидать, чтобы система `Mathematica` использовала подобное соглашение — символ есть символ, хоть символом его назови, хоть как. Это значит, что если уравнение содержит *единственный* символ, то этот символ и будет с необходимостью истолкован как неизвестная, так что в этом случае команду `Solve` можно вызывать с одним аргументом, в формате `Solve[f[x]==f[y]]`:

```
In[31]:=Solve[x^3+3*x^2-3*x+1==0]
Out[31]={{x->-1-2^(1/3)-2^(2/3)},
         {x->-1+(1-I*Sqrt[3])/2^(1/3)+(1+I*Sqrt[3])/2^(2/3)},
         {x->-1+(1-I*Sqrt[3])/2^(2/3)+(1+I*Sqrt[3])/2^(1/3)}}
```

◦ В то же время, если уравнение содержит несколько символов, то указание на то, который из них рассматривается как переменная, абсолютно необходимо. Если это не сделать, то система будет решать уравнение относительно того символа, который — с ее точки зрения — идет первым в *записи* этого уравнения. Это значит, что (в зависимости от истории сессии и настроек ядра) в качестве неизвестной в уравнении $ax^2+bx+c=0$ обычно будет рассматриваться $a$ или $c$, но отнюдь не $x$! При попытке провести вычисление

```
In[32]:=Solve[a*x^2+b*x+c==0]
```

результатом будет сообщение об возможной ошибке:

```
Solve:  Equations may not give solutions for all
             "solve" variables
```

сообщающее нам, что мы пытаемся найти *четыре* неизвестных из единственного уравнения, и следующий аутпут:

```
Out[32]={{c->-b*x-a*x^2}}
```

Таким образом, чтобы решить уравнение $ax^2+bx+c=0$ относительно $x$, нужно задавать системе вопрос в форме `Solve[a*x^2+b*x+c==0,x]`, где $x$ явно декларируется в качестве неизвестной.

◦ Кроме того, обратите внимание на формат ответа! Найденные корни выражаются в форме **правил подстановки** `x->c`, с тем, чтобы их можно было подставлять в другие выражения, не модифицируя при этом значения самого $x$ (которое остается независимой переменной).

По этому поводу стоит заметить, что в `Mathematica` имеется еще одна команда решения алгебраических уравнений *от одной неизвестной*, а именно `Roots`, которая выражает *набор* корней уравнения как объединение наборов корней более простых уравнений. При этом ответ записывается

как дизъюнкция этих более простых уравнений — и, тем самым, в большинстве случаев, которые могут реально встретиться начинающему, как дизъюнкция линейных уравнений. Использование команды `Roots` по своему синтаксису ничем не отличается от использования команды `Solve` *для одной неизвестной*:

```
Roots[f[x]==g[x],x].
```

Следующий пример иллюстрирует использование команды `Roots` для решения алгебраического уравнения:

```
In[33]:=Roots[x^2+x+1/x+1/x^2==4,x]
Out[33]=x==1/2(-3-Sqrt[5])||x==1/2(-3+Sqrt[5])||x==1||x==1
```

Чтобы перевести этот ответ в форму списка правил подстановки, порождаемую командой `Solve`, необходимо применить к нему форматную команду `ToRules`. Несмотря на более привычную форму ответа, возвращаемого командой `Roots`, при решении уравнений *в вещественных или комплексных числах* в большинстве ситуаций предпочтительно пользоваться более общими командами `Solve`, при помощи которой можно решать уравнения от нескольких неизвестных, и `Reduce`, решающей или упрощающей в том же стиле, что `Roots`, любые уравнения, как алгебраические, так и трансцендентные, а также неравенства, логические суждения и т.д. С другой стороны, команда `Roots` использует чисто алгебраические алгоритмы (те же, что `Factor` и `Decompose`) и оказывается *незаменимой* во многих ситуациях, когда `Solve` бессильна (конечные поля, модулярная арифметика, вычисления в кольцах и т.д.).

Увидев пустой ответ, не забудьте внимательно пересчитать скобки!!! А именно:

◦ Ответ в форме {} означает, что у уравнения нет решений. Например, мы получим такой ответ, предложив системе проделать следующее упражнение: `Solve[1==0,x]`.

◦ Ответ в форме {{}} означает, что *любое* допустимое значение аргумента служит решением уравнения. Мы получим такой ответ предложив системе вычислить `Solve[(x-1)(x+1)==x^2-1,x]`.

• **Уравнения степени $\leq 4$.** Уравнения степени $\leq 4$ команды `Solve` и `Roots` решают в школьном стиле — т.е. в радикалах, возвращая формулы в духе Кардано и Феррари. Вот, скажем, как выглядят корни многочлена Тэйлора $1 + x + x^2/2 + x^3/6$ порядка 3 для экспоненты:

```
In[34]:=Solve[Sum[x^i/i!,{i,0,3}]==0,x]
Out[34]={{x->-1-1/(-1+Sqrt[2])^(1/3)+(-1+Sqrt[2])^(1/3)},
         {x->-1-1/2*(-1+Sqrt[2])^(1/3)*(1-I*Sqrt[3])
             +(1+I*Sqrt[3])/(2*(-1+Sqrt[2])^(1/3))},
         {x->-1+(1-I*Sqrt[3])/(2*(-1+Sqrt[2])^(1/3))
             -1/2*(-1+Sqrt[2])^(1/3)*(1+I*Sqrt[3])}}
```

В тех случаях, когда системе удается решить в таком же стиле уравнения более высоких степеней, она делает это. Посмотрим, как система

борется с уравнением $x^5 + x - c = 0$, которое, вообще говоря, в радикалах не решается. Более того, как хорошо известно, при помощи **преобразования Чирнгаузена** решение любого алгебраического уравнения степени 5 сводится к уравнению такого вида, называемому **уравнением Бринга—Джерарда**[25]. Для того, чтобы чуть упростить вид входящих в решение радикалов, вынести общие множители и пр., поверх команды `Solve` рекомендуется применять команду `Simplify` — но, как будет объяснено ниже, вообще говоря, не команду `FullSimplify`:

```
In[35]:=Simplify[Solve[x^5+x-1==0,x]]
Out[35]={{x->(-1)^(1/3)},{x->-(-1)^(2/3)},
        {x->1/6*(-2+2^(2/3)*(25-3*Sqrt[69])^(1/3)+
            2^(2/3)*(25+3*Sqrt[69])^(1/3)},
        {x->-1/3-1/6*(1+I*Sqrt[3])*(1/2*(25-3*Sqrt[69]))^(1/3)
            +1/6*I*(I+Sqrt[3])*(1/2*(25+3*Sqrt[69]))^(1/3)},
        {x->-1/3+1/6*I*(1+I*Sqrt[3])*(1/2*(25-3*Sqrt[69]))^(1/3)
            -1/6*(I+Sqrt[3])*(1/2*(25+3*Sqrt[69]))^(1/3)}}
```

Заметим, что формулы с радикалами легко отключить, задав в теле команд опции `Cubics->False` или `Quartics->False` (по умолчанию обе эти опции поставлены на `True`).

Еще одной чрезвычайно интересной и полезной опцией, которую допускает команда `Roots` при решении алгебраических уравнений с рациональными коэффициентами, является `Modulus`. Значением модуля может быть любое целое число (по умолчанию `Modulus->0`). Это значит, что включив в тело команды опцию `Modulus->m`, мы ищем решение уравнения в кольце $\mathbb{Z}/m\mathbb{Z}$ классов вычетов по модулю $m$.

```
In[36]:=Roots[3*x^2+5==0,x,Modulus->17]
Out[36]=x==2||x==15
```

Стоит подчеркнуть, что при помощи команды `Solve` без довольно деликатной перенастройки проделать подобное упражнение не просто.

• **Корни алгебраических уравнений.** Почему мы не рекомендуем применять команду `Full Simplify`? Дело в том, что система пытается привести выражение к наиболее простому виду, а никакого *более простого* описания корня $c$ алгебраического уравнения, чем само это — или какое-то другое алгебраическое уравнение, корнем которого является $c$ — в общем случае не существует. Иными словами, утверждение, что $f(c) = 0$ для некоторого многочлена $f$, возможно в сочетании с какими-то утверждениями о локализации корня $c$ и/или изоляции корней, является в подавляющем большинстве случаев *гораздо* лучшим описанием числа $c$ с вычислительной точки зрения, чем любое другое его описание!!! Например, вычислив `Solve[x^5+x-2==0,x]`, даже после упрощения ответа при помощи `Simplify`, Вы натолкнетесь на полторы–две страницы радикалов. Но полностью упростив это выражение, Вы увидите следующее:

[25] В.В.Прасолов, Ю.П.Соловьев, Эллиптические кривые и алгебраические уравнения. — М., Факториал, 1997, с.1–288; стр.222–225.

```
In[37]:=FullSimplify[Solve[x^5+x-2==0,x]]
Out[37]={{x->1},{x->Root[2+#1+#1^2+#1^3+#1^4&,3]},
         {x->Root[2+#1+#1^2+#1^3+#1^4&,4]},
         {x->Root[2+#1+#1^2+#1^3+#1^4&,1]},
         {x->Root[2+#1+#1^2+#1^3+#1^4&,2]},
```

Прокомментируем вначале *форму* ответа. Команда `Root[f,i]` возвращает $i$-й корень *алгебраического* уравнения $f(x) = 0$. Нам нет нужды сейчас *точно* описывать порядок, в котором `Mathematica` учитывает корни. В первом приближении можно считать, что она руководствуется следующими правилами:

◦ Вещественные корни предшествуют комплексным и упорядочиваются естественным образом;

◦ Сопряженные комплексные корни приводятся *парами* и упорядочиваются лексикографически: вначале по вещественной части, а потом по мнимой части того корня из пары, который лежит в верхней полуплоскости.

При этом если все коэффициенты исходного уравнения были числами, полученные при помощи применения команды `Root` объекты тоже рассматриваются как числа, иными словами, к ним можно применять все обычные операции над числами, сравнивать их, вычислять приближенные значения и т.д.

Новым и весьма необычным для начинающих моментом в использовании команды `Root` является то, что многочлен $f$ задается в формате **чистой** или **анонимной функции**. Вызов $i$-го корня многочлена $f = a_n x^n + \ldots + a_1 x + a_0$ может быть произведен в одном из следующих эквивалентных форматов:

◦ `Root[Function[x,a_n*x^n+...+a_1*x+a_0],i]`

◦ `Root[a_n*#^n+...+a_1*#+a_0&,i]`

Первый из этих форматов (функциональный, формат чистой функции) рассматривается как основной способ внутреннего представления выражения, включающего объекты типа `Root`, в языке системы. Мы настоятельно рекомендуем начинающему пользоваться именно этим форматом, несмотря на чуть большую длину получающихся при этом текстов.

Второй формат (операторный, формат анонимной функции) рассматривается системой как сокращение первого. Его назначение состоит только в том, чтобы слегка уменьшить длину ввода за счет использования следующих операторов:

◦ Оператор `#` или, в полной форме `Slot`, обозначает *аргумент* чистой функции, которому мы не хотим присваивать никакого индивидуального имени (отсюда название `анонимная` функция — в этом случае не только сама функция, но и ее аргументы не имеют индивидуальных имен);

◦ Если у анонимной функции несколько аргументов, то они будут вызываться как `#1`, `#2`, `#3`, и так далее, по мере появления;

◦ Оператор `&` есть просто сокращение для `Function`, и обозначает *применение* чистой функции.

Вернемся теперь к *смыслу* последнего результата. Он означает, что система уверена в том, что никакого "более простого" описания корней $c \neq 1$ уравнения $x^5 + x - 1 = 0$, чем как корни уравнения $x^4 + x^3 + x^2 + x + 2 = 0$, *не существует*. Иными словами, она предпочитает использовать именно это описание (а не полустраничное выражение каждого корня в виде комбинации радикалов) в вычислениях и советует нам делать то же самое.

Во многих случаях она с самого начала считает, что у каких-то чисел нет *вообще никакого* более простого описания, чем как корни того уравнения, которое мы пытаемся решить! В этом случае она выражает корни этого уравнения как корни этого уравнения:

```
In[38]:=FullSimplify[Solve[x^5+x-3==0,x]]
Out[38]={{x->Root[-3+#1+#1^5&,1]},{x->Root[-3+#1+#1^5&,2]},
         {x->Root[-3+#1+#1^5&,3]},{x->Root[-3+#1+#1^5&,4]},
         {x->Root[-3+#1+#1^5&,5]}
```

Что, конечно, ничуть не мешает ей знать приближенные *численные значения* этих корней и использовать их в других вычислениях по нашей просьбе:

```
In[39]:=Table[N[Root[-3+#1+#1^5&,i]],{i,1,5}]
Out[39]={1.133,-1.04188-0.82287*I,-1.04188+0.82287*I,
         0.475381-1.1297*I, 0.475381+1.1297*I}
```

С другой стороны, мы можем всегда (когда это возможно!) заставить систему записать формулу в радикалах, применив к объектам типа `Root` команду `ToRadicals`. Более того, система сама делает это для уравнений степени $\leq 2$:

```
In[40]:=f:=Function[x,a*x^2+b*x+c]; {Root[f,1],Root[f,2]}
Out[40]={-b/(2*a)-1/2*Sqrt[(b^2-4*a*c)/a^2],
         -b/(2*a)+1/2*Sqrt[(b^2-4*a*c)/a^2]}
```

Для вычислений с алгебраическими числами чрезвычайно полезна команда `RootReduce`, которая пытается переписать сложное выражение, содержащее корни уравнения, в терминах *единственного* корня этого (или какого-то другого!) уравнения.

• **Что значит решить уравнение?** Тому, кто не задумывался над тем, что значит `решить уравнение`, этот вопрос, скорее всего, покажется чисто схоластическим. Между тем, для каждого, кто *реально* проводит вычисления, этот вопрос становится абсолютно конкретным и неотвратимым.

В самом деле, с младших классов школы нас приучили никуда особенно не вникая писать, что корни уравнения $x^2 - 2 = 0$ равны $x = \pm\sqrt{2}$ и называть это `решением уравнения`. Да, но что такое $\sqrt{2}$? Ведь если вернуться к истокам, то $\sqrt{2}$ как раз и *определяется* как (единственный) положительный корень уравнения $x^2 - 2 = 0$. Да, но где живет этот корень? Коэффициенты нашего исходного уравнения рациональны, но $\sqrt{2}$ не является рациональным числом. Любое фактическое вычисление с $\sqrt{2}$

либо является приближенным, либо сводится к тому, что мы возводим его в квадрат и проводим вычисление с рациональными числами!! Тем самым, любое точное вычисление с $\sqrt{2}$ основано непосредственно на том факте, что $\sqrt{2}^{\,2} - 2 = 0$, т.е. на том, что $\sqrt{2}$ является корнем уравнения $x^2 - 2 = 0$. Никакого более простого описания у числа $\sqrt{2}$ нет. Вся школьная премудрость о решении квадратного уравнения $x^2 - c = 0$ сводится к тому, что мы заявляем, что корнем этого уравнения является корень уравнения $x^2 - c = 0$. Иными словами, под влиянием школьной математики нам *мнится*, что мы умеем решать квадратные уравнения и *не умеем* решать уравнений пятой степени, ну, хотя бы, $x^5 + x - 3 = 0$.

Однако с нашей точки зрения статус уравнений $x^2 - \mathrm{c} = 0$ и $x^5 + x - \mathrm{c} = 0$ абсолютно одинаков и никакой концептуальной разницы между ними нет. Для каждого из них — да и вообще для любого алгебраического уравнения разумной степени — мы можем за доли секунды приближенно найти корни с любой точностью. В то же время в смысле арифметики ни одно из этих уравнений — без введения дополнительных функций — точно не решается. С алгоритмической точки зрения вычисление *значения* квадратного корня основано на разложении в ряд и ничем не отличается от вычисления значения любой другой аналитической функции. Единственное отличие носит чисто психологический характер и состоит в том, что мы ввели специальный значок $\sqrt{c}$ для положительного корня первого из этих уравнений

$$\sqrt{c} = 1 + \frac{1}{2}(c-1) - \frac{1}{8}(c-1)^2 + \frac{1}{16}(c-1)^3 - \frac{5}{128}(c-1)^4 + \dots$$

и долго учились им манипулировать. Если бы мы ввели специальный значок, ну хотя бы $\P(c)$, для **функции Ламберта—Эйзенштейна**, выражающей положительный корень второго из этих уравнений

$$\P(c) = c - c^5 + 10\frac{c^9}{2!} - 15\cdot 14\frac{c^{13}}{3!} + 20\cdot 19\cdot 18\frac{c^{17}}{4!} - \cdots .$$

и в течение такого же времени упражнялись в манипуляциях с этим значком[26,27], а потом с преобразованием Чирнгаузена, позволяющим свести решение любого уравнений пятой степени к последовательности арифметических операций и применения $\sqrt{\ }$, $\sqrt[3]{\ }$ и $\P$, то у нас сформировалось бы отчетливое — и, в целом, вполне обоснованное — убеждение, что мы *умеем* решать уравнения пятой степени.

Тот же вопрос, но, конечно, с еще большей остротой, встает при попытке решения трансцендентных уравнений. Что такое $\pi$? *По определению* это наименьший положительный корень уравнения $\sin(x) = 0$. Пользуясь соотношениями между основными тригонометрическими функциями и теоремами сложения несложно доказать, что тогда $\pi$ будет и корнем

[26] S.J.Patterson, Eisenstein and the quintic equation. — Historia Math, vol.17, 1990, p.132–140.

[27] J.Stillwell, Eisenstein's footnote. — Math. Intelligencer, v.17, N.2, 1995, p.58–62.

уравнений $\cos(x) = -1$, $\cos(x/2) = 0$, $\sin(x/2) = 1$ и тому подобные факты. Но все равно в конце дня (`in the final analysis`, `at the end of the day`) выясняется, что $\pi$ никогда не было ничем, кроме наименьшего положительного корня уравнения $\sin(x) = 0$. В своем знаменитом учебнике[28] Эдмунд Ландау *определяет* $\pi$ как наименьшее положительное число такое, что $\cos(\pi/2) = 0$ (определение 61) и *доказывает*, что тогда $\sin(\pi) = 0$ (теорема 267), но мы же слышали о логической эквивалентности?

Из сказанного выше ясно, что в общем случае трансцендентные уравнения не решаются. У нас просто нет такого количества букв, чтобы присвоить индивидуальное имя каждому корню всех встречающихся в природе трансцендентных уравнений. Более того, в большинстве случаев очень трудно даже связать решения различных трансцендентных уравнений или вообще доказать какое-то индивидуальное суждение об этих решениях. Самый знаменитый пример, когда мы не в состоянии решить простейшее трансцендентное уравнение — это **гипотеза Римана**. При $\mathrm{re}(s) > 1$ определим **дзета-функцию Римана** сходящимся рядом

$$\zeta(s) = \sum_{n=1}^{\infty} \frac{1}{n^s}.$$

Эта функция допускает аналитическое продолжение на всю комплексную плоскость, с полюсом в $s = 1$ (гармонический ряд расходится!) Так вот, многие факты в теории чисел зависят от того, что все нули дзета-функции в критической полосе $0 < \mathrm{re}(s) < 1$ расположены на прямой $\mathrm{re}(s) = 1/2$. Однако на протяжении полутора веков решение этой проблемы — которую Давид Гильберт называл главной задачей не только математики, но и всей жизни! — ускользает от всех усилий специалистов. Иными словами, мы не можем решить уравнение $\zeta(s) = 0$ даже при дополнительных предположениях о локализации корней!

• **Решение трансцендентных уравнений.** Команда `Solve` может служить и для решения трансцендентных уравнений, в тех случаях, когда система может легко сделать это известными ей методами. Однако в действительности правильной командой для решения трансцендентных уравнений в подавляющем большинстве случаев является `Reduce`, а отнюдь не `Solve`!! Разумеется, так как мы *не умеем* решать трансцендентных уравнений, то даже `Reduce` не может в общем случае дать правильного ответа. Однако применение `Reduce` по крайней мере гарантирует от получения неправильного или неполного ответа. Сравним эти команды на конкретных примерах. Скажем, попытка провести вычисление

```
In[41]:=Solve[Exp[2*x]+2*Exp[x]+1==x,x]
```

приводит к следующему сообщению об ошибке:

```
Solve: This system cannot be solved with the methods
```

[28] Э.Ландау, Введение в дифференциальное и интегральное исчисление. — 1948, ИЛ, М., с.1–458.

```
        available to Solve.
```

В результате система просто оставляет наше исходное выражение неэвалюированным. К сожалению, в данном случае и применение `Reduce` не приводит к большому успеху, в ответ на попытку вычислить

```
In[41]:=Reduce[Exp[2*x]+2*Exp[x]+1==x,x]
```

мы получаем такое оптимистическое сообщение:

```
Reduce: This system cannot be solved with the methods
        available to Reduce.
```

Так что не нарисовав графики левой и правой части при помощи команды `Plot`, Вы так и не сможете узнать, имеет это уравнение вещественные корни, или нет. С другой стороны при попытке вычислить

```
In[41]:=Solve[Sqrt[Log[x]]==Log[Sqrt[x]],x]
```

система выдает сообщение о *возможной* ошибке:

```
Solve: Inverse functions are being used by Solve,
        so some solutions may not be found;
        use Reduce for complete solution information.
```

Однако в данном случае, несмотря на свою озабоченность, система возвращает правильный ответ:

```
Out[41]={{x->1},{x->E^4}}
```

Следует отметить, что встречаются примеры — и мы увидим их в Модуле 2, — когда система действительно находит лишь *часть* решений, а не все решения, так что обычно к подобному предупреждению следует относиться с полной серьезностью. Вот еще один похожий пример, в котором мы игнорируем `error message` и в результате получаем лишь частичный ответ:

```
In[42]:=Solve[Sin[x]==Cos[x],x]
Out[42]={{x->-3*Pi/4},{x->Pi/4}}
```

Понятно, что это не совсем то, что мы хотели. В то же время применение здесь команды `Reduce` дает уже абсолютно осмысленный ответ:

```
In[43]:=Reduce[Sin[x]==Cos[x],x]
Out[43]=C[1]∈Integers&&(x==-2*ArcTan[1+Sqrt[2]]+2*Pi*C[1]||
          x==-2*ArcTan[1-Sqrt[2]]+2*Pi*C[1])
```

Дальнейшие примеры трансцендентных уравнений рассмотрены в § 5.

• **Численное решение уравнений.** Основной командой для численного решения алгебраических уравнений является команда `NSolve`. Дело в том, что точное решение алгебраических уравнений с последующим округлением может приводить к чрезвычайно забавным эффектам. В качестве типичного примера можно рассмотреть характеристическое уравнение кубической матрицы золотого сечения $1 - 5x + 6x^2 - x^3 = 0$. Решая это уравнение при помощи формулы Кардано, мы получим точные корни:

```
In[44]:=Solve[1-5*x+6*x^2-x^3==0,x]
```

```
Out[44]={{x->2+7^(2/3)/(3/2*(9+I*Sqrt[3])^(1/3)+
        (7/2*(9+I*Sqrt[3]))^(1/3)/3^(2/3)},
        {x->2-(7/2)^(2/3)*(1+I*Sqrt[3])/(3*(9+I*Sqrt[3]))^(1/3)
        -(1-I*Sqrt[3])*(7/2*(9+I*Sqrt[3]))^(1/3)/2*3^(2/3)},
        {x->2-(7/2)^(2/3)*(1-I*Sqrt[3])/(3*(9+I*Sqrt[3]))^(1/3)
        -(1+I*Sqrt[3])*(7/2*(9+I*Sqrt[3]))^(1/3)/2*3^(2/3)},
```

Как было известно итальянским математикам XVI века, все мнимости в этих формулах сокращаются, так что все три корня вещественные (Кардано назвал это явление `casus irreducibilis`). Однако попытка найти приближенные значения корней приводит к следующему удивительному ответу, еще раз иллюстрирующему полную абсурдность применения численных методов к задачам с точными условиями:

```
In[45]:=Map[N,Solve[1-5*x+6*x^2-x^3==0,x]]
Out[45]={{x->5.04892+2.77556*10^-17*I},
                    {x->0.307979+2.22045*10^-16*I},
                    {x->0.643104-2.22045*10^-16*I}}
```

Понятно, что здесь происходит? Система находит значения $9 + i\sqrt{3}$ и т.д. с машинной точностью (около 16 знаков после запятой), после чего начинает манипулировать с этими приближенными значениями. Но при приближенных вычислениях неизбежно возникают артефакты наподобие 2.77556, 2.22045 и т.д. В подобных случаях изначальная трактовка условий как приближенных приводит к лучшим результатам:

```
In[46]:=NSolve[1-5*x+6*x^2-x^3==0,x,20]
Out[46]={{x->0.3079785283699041304},
        {x->0.6431041321077905561},{x->5.048917339522305314}}
```

## § 4. Системы уравнений и неравенств

> Nowadays we can do computer experiments using `Mathematica`, and even solve a system of 42 equations. This offers another route to knowledge, rather than mere ideas.
>
> John F. Nash, Jr.

Конечно, преимущества системы `Mathematica` становятся полностью понятны только в тот момент, когда нам нужно решить систему 42 уравнений от 47 неизвестных. Для наглядности и из типографских соображений мы проиллюстрируем способности системы на чисто учебных примерах систем от *трех* неизвестных, но она сравнительно легко решает в реальном времени системы от десятка неизвестных, а, если дать ей немного подумать — то и от нескольких десятков неизвестных

В действительности в этом отношении `Mathematica` уступает только самым продвинутым *специализированным* системам полиномиальных вычислений, вроде `Singular`, которые, с другой стороны, не умеют делать абсолютно ничего, *кроме* полиномиальных преобразований и решения систем алгебраических уравнений. Решение систем алгебраических уравнений от нескольких сотен неизвестных сегодня все еще рассматривается как *очень* трудная задача для систем компьютерной алгебры.

• **Решение систем алгебраических уравнений.** Для решения системы алгебраических уравнений

$$f_1(x,y,z) = g_1(x,y,z), \dots , f_n(x,y,z) = g_n(x,y,z)$$

команда `Solve` вызывается в одном из следующих основных форматов:

◦ Как функция *двух* аргументов, первым из которых является *список* уравнений, а вторым — *список* тех неизвестных, относительно которых мы пытаемся решить систему:

```
Solve[{f1[x,y,z]==g1[x,y,z],...,fn[x,y,z]==gn[x,y,z]},{x,y}]
```

остальные неизвестные при этом рассматриваются как **параметры**.

◦ Как функция *двух* аргументов, первым из которых является *конъюнкция* уравнений, а вторым — *список* тех неизвестных, относительно которых мы пытаемся решить систему:

```
Solve[f1[x,y,z]==g1[x,y,z]&&...&&fn[x,y,z]==gn[x,y,z],{x,y}]
```

◦ Как функцию *двух* аргументов, первым из которых является *векторное* уравнение,

$$(f_1(x,y,z), \dots , f_n(x,y,z)) = (g_1(x,y,z), \dots , g_n(x,y,z)),$$

а вторым — *список* тех неизвестных, относительно которых мы пытаемся решить систему:

```
Solve[{f1[x,y,z],...,fn[x,y,z]}=={g1[x,y,z],...,gn[x,y,z]},
                                                {x,y}]
```

◦ Как функция *одного* аргумента, который при этом оформляется либо как список, либо как конъюнкция уравнений, либо, наконец, как векторное уравнение:

```
Solve[{f1[x,y,z]==g1[x,y,z],...,fn[x,y,z]==gn[x,y,z]}]
Solve[f1[x,y,z]==g1[x,y,z]&&...&&fn[x,y,z]==gn[x,y,z]]
Solve[{f1[x,y,z],...,fn[x,y,z]}=={g1[x,y,z],...,gn[x,y,z]}]
```

В этом случае система изо всех сил пытается найти значения **всех** входящих в эти уравнения символов, что, вообще говоря, не всегда получается, если количество уравнений меньше, чем количество неизвестных и параметров.

◦ Как функция *трех* аргументов, первым из которых является система уравнений (представленная в любой из трех описанных выше эквивалентных форм), вторым — *список* тех неизвестных, относительно которых мы пытаемся решить систему, а третьим — та неизвестная или список тех неизвестных, которые мы при хотим при этом полностью **элиминировать** (= исключить) из ответа, как в качестве неизвестных, так и в качестве параметров:

```
Solve[f1[x,y,...]==g1[x,y,...]&&...&&fn[x,y,...]==gn[x,y,...],
                                                {x,y},{u,v}]
```

Поскольку понять (а тем более объяснить!!) не только все детали, но даже азы того, что происходит при вызове команды `Solve` с тремя аргументами, довольно трудно, мы настоятельно рекомендуем начинающему использовать команду `Eliminate` для исключения неизвестных, а потом уже решать получающуюся систему обычным образом.

• **Enough, or too little.** Проиллюстрируем решение систем алгебраических уравнений на простейших примерах. Заметим, что в связи с используемой процедурой элиминации неизвестных система может несколько раз порождать одно и то же решение, поэтому, если нас — в школьном духе — интересует *множество* решений, для сокращения ответа мы обычно применяем поверх команды `Solve` команду `Union`.

Мы уже видели, что уже в случае одного уравнения `Mathematica` не знает, какие из входящих в него символов следует рассматривать как неизвестные, а какие как параметры. Для правильного решения *систем* уравнений эта проблема становится абсолютно критической. Как всегда, `Mathematica` может посчитать некоторые некоторые параметры неизвестными — но это сравнительно мелкая неприятность (или, на компьютерном языке, `small beer`). В действительности, здесь может произойти *значительно* более серьзное несчастье: `Mathematica` может решить, что некоторые неизвестные являются параметрами!!! А это, как правило, приводит к неверному ответу, причем получающиеся ошибки очень трудно отслеживаются (особенно если решение уравнений бесконтрольно вызывается в составе другого вычисления). Например, если Вы постараетесь решить систему

$$x - y = z^2, \qquad x^3 = 1, \qquad y^3 = 1$$

относительно $x$ и $y$ посредством беззастенчивого

```
In[47]:=Solve[{x-y==z^2,x^3==1,y^3==1},{x,y}]
```

то ответом будет оглушительное {}, хотя совершенно ясно, что эта система имеет решение, ну хотя бы $(x, y, z) = (1, 1, 0)$. Ясно, что произошло? Дело в том, что у этой системы нет *общих* решений, в которых $z$ могло бы выступать в качестве параметра. Для любых допустимых значений $x$ и $y$ мы получаем явное уравнение на $z$. А попросив разрешить систему уравнений относительно $x$ и $y$ мы предложили `Mathematica` найти такие решения, в которых $z$ выступает в качестве параметра!!! Это значит, что мы окажемся в гораздо лучшем положении, если попросим больше. И действительно, вычисление

```
In[48]:=Union[Solve[{x-y==z^2,x^3==1,y^3==1},{x,y,z}]]
```

дает все 15 решений системы. Итак, решая систему уравнений, всегда задавайте себе вопрос, что Вас интересует: нахождение *всех* решений системы или нахождение *общих* решений, в которых какие-то неизвестные выступают в качестве параметров?

**Мораль:** Чтобы избежать грубых ошибок, ВСЕГДА ПРОСИТЕ У СИСТЕМЫ БОЛЬШЕ, ЧЕМ ХОЧЕТСЯ.

• **Примеры решения систем.** Приведем несколько чисто учебных примеров решения систем алгебраических уравнений.

◦ В первом примере мы пытаемся решить систему[29]

$$x^2 + y + z = 1, \qquad x + y^2 + z = 1, \qquad x + y + z^2 = 1$$

относительно всех входящих в нее неизвестных:

```
In[49]:=Union[Solve[{x^2+y+z==1,x+y^2+z==1,x+y+z^2==1}]]
Out[49]={{x->0,y->0,z->1},{x->0,y->1,z->0},{x->1,y->0,z->0},
         {x->-1-Sqrt[2],y->-1-Sqrt[2],z->-1-Sqrt[2]},
         {x->-1+Sqrt[2],y->-1+Sqrt[2],z->-1+Sqrt[2]}}
```

◦ Решение уже простейших систем, зависящих от параметров, представляет собой довольно серьезную задачу. Дело в том, что при последовательном исключении неизвестных степень уравнений относительно остающихся неизвестных быстро растет. Например, исключая неизвестные $y$ и $z$ из системы

$$x^2 + y + z = a, \qquad x + y^2 + z = b, \qquad x + y + z^2 = c$$

мы получаем уравнение степени 8 относительно $x$, коэффициенты которого довольно тягостным образом выражаются через $a, b, c$. Это значит, что попытавшись решить систему относительно уравнений $x, y, z$ беззастенчивым

```
In[50]:=Solve[{x^2+y+z==a,x+y^2+z==b,x+y+z^2==c},{x,y,z}]
```

[29] Д.Кокс, Дж.Литтл, Д.О'Ши, Идеалы, многообразия и алгоритмы. — Москва, Мир, 2000, с.1–687; стр.151–152.

Вы *конечно*, увидите на экране решение. Вот только что именно Вы собираетесь делать с таким решением, координаты которого выражены в терминах объектов типа `Root` от многочлена степени 8, каждый из коэффициентов которого в свою очередь является многочленом изрядной степени от $a, b$ и $c$? Такой ответ легко может занять сотни или тысячи строк на экране. Поэтому в тех случаях, когда мы не знаем, что произойдет, мы часто вначале интересуемся не самим ответом, а *длиной* ответа, в данном случае *количеством* решений:

```
In[51]:=Length[Union[Solve[{x^2+y+z==a,x+y^2+z==b,x+y+z^2==c},
                                                    {x,y,z}]]]
```

Ну в данном-то случае у нашей системы 8 решений. А если их несколько десятков или несколько сотен? В этом случае мы обычно смотрим на *вид* решений посредством команды `Short`:

```
In[52]:=Short[Union[Solve[{x^2+y+z==a,x+y^2+z==b,x+y+z^2==c},
                                                    {x,y,z}]],10]
```

Вызванная в формате

```
Short[expression,d]
```

команда `Short` приводит к тому, что на экране отображается не все выражение `expression`, которое может занимать несколько сотен страниц и форматирование которого может требовать весьма значительного времени, а только его часть, *приблизительно* $d$ строчек. `Приблизительно` потому, что система все же пытается вывести осмысленную часть выражения, по которой можно составить хотя бы самое общее впечатление о виде всего остального.

В общем случае (например, если количество уравнений меньше, чем количество неизвестных) решить систему относительно всех неизвестных не удастся, в этом случае следует явно указывать те неизвестные, которые мы хотим выразить через остальные.

◦ В следующей задаче мы просим `Mathematica` выразить в системе

$$x + y + z = 1, \qquad x^2 + y^2 + z^2 = 1,$$

неизвестные $x, y$ через неизвестную $z$, которая в этом случае трактуется как параметр:

```
In[53]:=Solve[{x+y+z==1,x^2+y^2+z^2==1},{x,y}]
Out[53]={{x->1/2*(1-z-Sqrt[1+2*z-3*z^2]),
              y->1/2*(1-z+Sqrt[1+2*z-3*z^2])},
          {x->1/2*(1-z+Sqrt[1+2*z-3*z^2]),
              y->1/2*(1-z-Sqrt[1+2*z-3*z^2])}}
```

Если попытаться решить систему относительно всех трех неизвестных, скажем, так:

```
In[54]:=Solve[{x+y+z==1,x^2+y^2+z^2==1},{x,y,z}]
```

то результатом будет уже знакомое нам сообщение об ошибке:

```
Solve: Equations may not give solutions for all
         "solve" variables.
```

Тем не менее, Mathematica снова вернет те же самые формулы, что и в предыдущем случае. Кстати, как Вы думаете, почему она и в этом случае выражает $x$ и $y$ через $z$? Ну это, как раз, совершенно понятно. Она пытается решить систему в первую очередь относительно тех переменных, которые перечислены в списке первыми. В ответ на

```
In[55]:=Solve[{x+y+z==1,x^2+y^2+z^2==1},{z,y,x}]
```

она выдаст то же сообщение об ошибке и выразит $z$ и $y$ (в таком порядке!) через $x$.

• **Исключение неизвестных.** В тех случаях, когда уравнений недостаточно, чтобы найти все неизвестные, часто полезно просто свести исходную систему уравнений к системе от меньшего количества неизвестных. В случае алгебраических уравнений основной командой для этого в языке Mathematica является Eliminate. Для исключения неизвестной $z$ из системы алгебраических уравнений

$$f_1(x,y,z) = g_1(x,y,z), \ldots, f_n(x,y,z) = g_n(x,y,z)$$

команда Eliminate вызывается в формате

```
Eliminate[{f1[x,y,z]==g1[x,y,z],...,fn[x,y,z]==gn[x,y,z]},z]
```

функции двух аргументов, первым из которых является *список* уравнений, а вторым — исключаемая неизвестная (или *список* неизвестных, если их несколько). Для получения более симметричного ответа, как всегда, рекомендуется применять поверх команды Eliminate команду Simplify. Вот пара простейших примеров:

```
In[56]:=Simplify[Eliminate[{x+y+z==1,x*y*z==1},y]]
Out[56]=x*z*(-1+x+z)==-1
In[57]:=Simplify[Eliminate[{x+y+z==1,x^2+y^2+z^2==1},y]]
Out[57]=x^2+x*z+z^2==x+z
```

Если нам нужно одновременно исключить несколько неизвестных, то эти неизвестные тоже должны задаваться в виде списка. Вот пример, где мы ищем соотношение между суммами степеней, для чего требуется исключить сразу две неизвестных:

```
In[58]:=Simplify[Eliminate[{u==x^5+y^5,v==x^3+y^3,w==x^2+y^2},
                                                        {x,y}]]
Out[58]=2*u^3+30*u*v^2*w^2+5*v*w^6==
                  v^5+15*u^2*v*w+15*v^3*w^3+6*u*w^5
```

Ясно, что уже в подобном примере выполнение исключения вручную требует известного присутствия духа и уверенности в своих технических возможностях.

При применении команды `Eliminate` система пытается известными ей методами исключать и неизвестные из трансцендентных уравнений, конечно, не всегда одинаково успешно.

• **Решение неравенств.** Естественно, `Mathematica` может решать не только уравнения, но и неравенства. В § 1 мы уже видели, что неравенство записывается обычным образом:

◦ Строгое неравенство `x>y Greater` или `x<y Less` ;

◦ Нестрогое неравенство `x>=y GreaterEqual` или `x<=y LessEqual` .

Обычно система считает, что связанные неравенством величины представляют собой *вещественные числа*, но она производит и некоторые упрощения частей неравенства и в том случае, когда они не являются числами, хотя и не может в этом случае приписать неравенству значение истинности `True` или `False`.

Основной командой для решения неравенств и систем неравенств в языке `Mathematica` является `Reduce`. В простейшем варианте команда `Reduce` вызывается в формате

```
Reduce[expression,{x,y}]
```

где $x, y$ это переменные, относительно которых мы хотим разрешить систему уравнений, неравенств и логических высказываний, а `expression` представляет собой произвольную логическую комбинацию следующих вещей:

◦ уравнений `f[x,y,...]==f[x,y,...]`,

◦ неравенств `f[x,y,...]!=f[x,y,...]`,

◦ строгих неравенств `f[x,y,...]<f[x,y,...]`,

◦ нестрогих неравенств `f[x,y,...]<=f[x,y,...]`,

◦ спецификаций доменов `Element[x,domain]`, утверждающих, что $x$ принадлежит домену `domain`.

◦ кванторов всеобщности `ForAll[x,condition[x],test[x]]`, возвращающих значение `True`, если тест `test[x]` принимает значение `True` *для всех* значений $x$, для которых выполняется условие `condition[x]`.

◦ кванторов существования `Exists[x,condition[x],test[x]]`, возвращающих значение `True`, если *существует* значение $x$, удовлетворяющее условию `condition[x]`, для которого тест `test[x]` принимает значение `True`.

По умолчанию команда `Reduce` считает, что все входящие в нее неизвестные принимают *комплексные* значения, причем все неизвестные, которые явным образом входят в строгие и нестрогие неравенства — *вещественные*. Поэтому для решения неравенств в *вещественных* числах никаких спецификаций доменов задавать не нужно. Если мы хотим решить систему неравенств в *целых* числах, то проще всего тоже не задавать спецификацию доменов для каждой переменной по отдельности, а сразу вызывать команду `Reduce` в формате:

```
Reduce[expression,{x,y},Integers]
```

В случае одного неравенства использование команды `Reduce` и вид получающегося при этом ответа ничем не отличается от того, что мы уже видели для команды `Roots` в случае одного уравнения:

```
In[59]:=Reduce[x+(x-1)/(x+1)>0,x]

Out[59]=-1-Sqrt[2]<x<-1||x>-1+Sqrt[2]
```

Разумеется, в общем случае, сводящемся к решению уравнений степени $\geq 3$ решение будет выражено в терминах объектов типа `Root`:

```
In[60]:=Reduce[x^2+(x-1)/(x^2+x+1)>1,x]

Out[60]=x<Root[2+2*#1+2*#1^2+#1^3&,1]||x>1
```

Для неравенств, сводящихся к уравнениям степени $\geq 4$, мы всегда можем, *при желании* выразить ответ в радикалах, установив в теле функции опцию `Cubics->True` и/или `Quartics->True`, в следующем формате:

```
In[61]:=Reduce[x^2+(x-1)/(x^2+x+1)>1,x,Cubics->True]

Out[61]=x<1/3*(-2-2/(-17+3*Sqrt[33])^(1/3)+
                          (-17+3*Sqrt[33])^(1/3)||x>1
```

Разумеется, неравенство совсем не обязано с самого начала иметь полиномиальный вид. С тем же успехом система решает любые неравенства, *сводящиеся* к алгебраическим, скажем, неравенства, содержащие абсолютные величины, квадратные корни и пр. Вот пример, основанный на задаче, фактически предлагавшейся на вступительном экзамене на экономический факультет МГУ:

```
In[62]:=Simplify[Reduce[Abs[y]+x-Sqrt[x^2+y^2-1]>=1,{x,y}]]

Out[62]=Element[y,Reals]&&
        (x==0&&(y==-1||y==1)||
        0<x<1&&(-1<=y<=-Sqrt[1-x^2]||Sqrt[1-x^2]<=y<=1)||
        x==1||
        x>1&&(y<=-1||y>=1))
```

Обратите внимание, что так как $y$ входит в неравенство не в явном виде, а в аргументе функций `Abs` и `Sqrt`, в ответе система считает нужным подчеркнуть, что она считает $y$ *вещественным* числом. При решении неравенств с абсолютными величинами нужно быть чрезвычайно аккуратным. Дело в том, что если все переменные входят в неравенства только под знаком абсолютной величины, то система будет считать их комплексными. Поэтому, скажем, `Reduce[Abs[x]+Abs[y]<1,{x,y}]` вернет гораздо больше решений, чем Вы ожидали! Если вы хотите увидеть только вещественные решения, нужно явно специфицировать домен, вызывая эту команду в следующем формате:

```
In[63]:=Reduce[Abs[x]+Abs[y]<1,{x,y},Reals]

Out[63]=-1<x<=0&&-1-x<y<1+x||0<x<1&&-1+x<y<1-x
```

Разумеется, система может решить и неравенства, в которые входят трансцендентные функции, по крайней мере в тех случаях, когда она может

решить соответствующее уравнение! Более того, даже для трансцендентных *уравнений* заведомо предпочтительно использовать именно команду `Reduce`, так как она всегда пытается определить все множество решений, а не просто *какие-то* решения. Однако во многих случаях при попытке найти *точные* решения неравенства ответом будет сакраментальное `Reduce: This system cannot be solved with the methods available to Reduce.`

Модификации, которые необходимо внести в использование команды `Reduce` для решение систем неравенств относительно нескольких неизвестных, точно такие же, как при использовании команды `Solve` для решения системы уравнений. А именно, в качестве первого аргумента команды `Reduce` в этом случае можно задавать список неравенств, а в качестве второго — список неизвестных, приблизительно в таком формате:

```
Reduce[{f1[x,y]>g1[x,y],f2[x,y]>g2[x,y],...},{x,y}]
```

С другой стороны, эту функцию можно вызывать и в следующем формате:

```
Reduce[f1[x,y]>g1[x,y]&&f2[x,y]>g2[x,y]&&...},{x,y}]
```

Однако следует иметь в виду, что уже решение *простейших* систем неравенств может состоять из нескольких компонент — иногда из огромного числа компонент:

```
In[64]:=Reduce[Abs[x]+Abs[y]>1&&x^2+x*y+y^2<1,{x,y}]
Out[64]=-2/Sqrt[3]<x<=-1&&
              -x/2-1/2*Sqrt[4-3*x^2]<y<-x/2+1/2*Sqrt[4-3*x^2]||
        -1<x<0&&(-x/2-1/2*Sqrt[4-3*x^2]<y<-1-x||
                                  1+x<y<-x/2+1/2*Sqrt[4-3*x^2])||
        0<x<=1&&(-x/2-1/2*Sqrt[4-3*x^2]<y<-1+x||
                                  1-x<y<-x/2+1/2*Sqrt[4-3*x^2])||
        1<x<2/Sqrt[3]&&
              -x/2-1/2*Sqrt[4-3*x^2]<y<-x/2+1/2*Sqrt[4-3*x^2]
```

Вы действительно в состоянии *сразу* увидеть все четыре компоненты решения по этим формулам? Мы уверены, что в подобных случаях — а также для трансцендентных уравнений — гораздо удобнее пользоваться *графическим* представлением ответа при помощи функции `RegionPlot`. Мы совсем коротко описываем использование этой функции в § 6.

## § 5. Элементарные функции

Я стал немного забывать теорию функций. Ну, это восстановится. Врач обещал ... врет, наверно.

Владимир Высоцкий. 'Дельфины и психи (записки сумасшедшего)'

Система `Mathematica` знает определения большого количества элементарных и специальных функций, и громадное количество фактов об их значениях, их поведении, дифференциальные и функциональные уравнения, которым они удовлетворяют, etc., etc., etc. Эти знания в сочетании с мощью ее интеллекта позволяют без труда решать любую задачу об этих функциях, которая может встретиться школьнику, студенту и вообще любому нематематику.

• **Экспоненты и логарифмы.** Как мы уже упоминали, именами всех обычных функций в языке `Mathematica` являются либо их *полные* английские названия, либо *стандартные* сокращения этих названий. Неспециалист должен иметь в виду, что — особенно в области элементарной математики — многие английские сокращения, в частности, имена большинства тригонометрических и гиперболических функций, отличаются от континентальных. В то же время русская педагогическая традиция основана на латинских и французских сокращениях!

В соответствии с этим этим общим принципом экспонента и логарифм называются `Exp` и `Log`:

◦ `Exp[x]` возвращает $e^x$;

◦ `Log[x]` возвращает натуральный логарифм $x$;

◦ `Log[b,x]` возвращает логарифм $x$ по основанию $b$. Обратите внимание на весьма необычный (для языка `Mathematica`!) порядок аргумента и параметра: в записи большинства функций аргументы предшествуют параметрам.

Стоит предупредить, что в языке `Mathematica` есть и функция `Exponent`, но по-английски слово `exponent` означает как собственно экспоненту, т.е. степень, так и показатель степени. Так вот `Exponent[f,x]` обозначает наибольший *показатель* степени, с которым $x$ входит в $f$.

Экспонента и логарифм наряду с круговыми ("тригонометрическими") и гиперболическими функциями являются примерами числовых функций. В Модуле 2 мы достаточно подробно обсуждаем специфику **числовых функций**, поэтому ограничимся пока констатацией того, что в тех случаях, когда их аргументы принимают численные значения, эти функции всегда, когда это возможно, пытаются вернуть *точные* численные значения.

В природных условиях `Mathematica` работает с экспонентой и логарифмом как *комплексными* функциями, считая при этом, что для логарифма разрез произведен по лучу $(0, -\infty)$. Вот, например, как `Mathematica` понимает **формулу Эйлера**:

```
In[65]:=ComplexExpand[Exp[x+I*y]]
Out[65]=E^x*Cos[y]+I*E^x*Sin[y]
```

А вот что такое логарифм отрицательного числа:

```
In[66]:=Refine[Log[x],x<0]
Out[66]=I*Pi+Log[-x]
```

Использованная здесь команда `Refine` является одной из самых полезных команд системы при работе с числовыми функциями. Вызванная в формате

```
Refine[expression,assumptions]
```

она упрощает выражение `expression` так как она сделала бы это при условии, что входящие в него символы заменены *явными числовыми значениями*, удовлетворяющими предположениям `assumptions`. В действительности во многих ситуациях, например, когда мы пытаемся упростить несколько различных функций при одних и тех же предположениях, удобно вызывать `Refine` внутри вспомогательной команды `Assuming`, в следующем формате:

```
Assuming[assumptions,Refine[expression]]
```

Стоит подчеркнуть, что команда `Refine` не конкурирует с командами `Simplify` и `FullSimplify`, а дополняет их. Дело в том, что `Simplify` опирается главным образом на общие полиномиальные алгоритмы, в то время как `Refine` использует несколько сотен *конкретных* типов преобразований известных системе числовых функций. Например, не только `Simplify`, но даже `FullSimplify` не выполняет произведенного выше упрощения:

```
In[67]:=FullSimplify[Log[x],x<0]
Out[67]=Log[x]
```

Это значит, что для упрощения формул, в которые входят значения числовых функций, полезно применить к ним как команду `Simplify`, так и команду `Refine`.

Используемые командой `Refine` предположения могут состоять из произвольных логических комбинаций неравенств, уравнений и спецификаций доменов. Непосредственно в языке ядра описаны всего *семь* доменов:

- $\mathbb{C}$ = `Complexes` — комплексные числа,
- $\mathbb{R}$ = `Reals` — вещественные числа,
- $\overline{\mathbb{Q}}$ = `Algebraics` — алгебраические числа (в документации фирмы `Wolfram Research` этот домен обозначается через $\mathbb{A}$, но большинство алгебраистов использует обозначение $\mathbb{A}$ не для поля алгебраических чисел, а для кольца *целых* алгебраических чисел),
- $\mathbb{Q}$ = `Rationals` — рациональные числа,
- $\mathbb{Z}$ = `Integers` — целые числа,
- $\mathbb{P}$ = `Primes` — простые числа,
- {`True`,`False`} = `Booleans` — значения истинности.

При этом условие принадлежности элемента $x$ домену `Domain` записывается в виде

```
Element[x,Domain]
```

Вот пример упрощения, использующего условие $y \in \mathbb{Z}$:

```
In[68]:=Refine[Exp[x+I*Pi*y],Element[y,Integers]]
Out[68]=(-1)^y*E^x
```

• **Круговые и гиперболические функции.** Естественно, в `Mathematica` имплементированы и все остальные основные элементарные функции, в частности, круговые, гиперболические, обратные круговые и обратные гиперболические. Все они рассматриваются как числовые функции (комплексного аргумента) и к ним относится все, что было сказано выше об экспоненте и логарифме.

Работая с комплексными числами мы уже имели возможность убедиться, что косинус и синус называются `Cos` и `Sin`. Аргумент круговых функций выражается в радианах, причем по умолчанию всегда возвращается *точное* значение функции. Впрочем, как и для любой числовой функции, точное значение всегда можно превратить в приближенное, применяя функцию `N`:

```
In[69]:={Sin[Pi/12],Sin[Pi/8],Sin[1],N[Sin[1]]}
Out[69]={(-1+Sqrt[3])/(2*Sqrt[2]),Sin[Pi/8],Sin[1],0.841471}
```

Однако при желании аргумент круговых функций можно выражать и в градусах, для этого его величину в градусах нужно умножить на константу `Degree`, равную *численному значению* $\pi/180$. Приведем первые 50 разрядов этой константы:

```
In[70]:=N[Degree,50]
Out[70]=0.017453292519943295769236907684886127134428718885417
```

Подчеркнем, что в отличие от `Pi` константа `Degree` имеет формат *приближенного* вещественного числа (хотя и неопределенной разрядности). Это значит, что `Mathematica` не может ответить на вопрос `Pi/180==Degree`. В то же время тест `N[Pi/180,d]==Degree` даст значение `True` при любом количестве разрядов $d$. Тем не менее, *к нашему большому удивлению*, вычисление значений круговых функций от углов, выраженных в градусах, дает *точные* значения:

```
In[71]:={Cos[15*Degree],Sin[15*Degree]}
Out[71]={(1+Sqrt[3])/(2*Sqrt[2]),(-1+Sqrt[3])/(2*Sqrt[2])}
```

Команда `Refine` работает обычным образом:

```
In[72]:={Refine[Cos[x+Pi/2]],Refine[Sin[Pi/2-x]]}
Out[72]={-Sin[x],Cos[x]}
```

В соответствии с английской традицией тангенс и котангенс называются `Tan` и `Cot`:

```
In[73]:={Sin[x]/Cos[x],Cos[x]/Sin[x]}
Out[73]={Tan[x],Cot[x]}
```

Названия основных гиперболических функций получаются из названий соответствующих круговых функций дописыванием буковки `h`. Таким образом, например, гиперболический косинус и гиперболический синус называются `Cosh` и `Sinh`, соответственно. Названия обратных функций получаются из исходных функций приписыванием приставки `Arc`, при этом имя исходной функции не меняется (в частности, продолжает писаться с большой буквы). Таким образом, например, арккосинус и арксинус называются `ArcCos` и `ArcSin`, а гиперболический арккосинус и гиперболический арксинус — `ArcCosh` и `ArcSinh`, соответственно.

• **Значения элементарных функций.** Одной из самых мощных и полезных команд системы `Mathematica` для работы со значениями *числовых функций* является команда `FunctionExpand`, которая пытается привести выражение к пусть и более длинному, но более "элементарному" виду. Таким образом, действие `FunctionExpand` в определенном смысле *противоположно* действию команды `Simplify`, которая пытается представить выражения в самом *коротком* виде, хотя бы как значения высших трансцендентных функций. При этом последовательное применение команд `FunctionExpand` и `Simplify` обычно не возвращает исходное выражение, а преобразует его к какому-то совсем другому виду! Команда `FunctionExpand` *пытается* сделать следующее:

◦ избавиться от явного применения операций анализа (дифференцирование, интегрирование и т.д.);

◦ выразить значения специальных функций в терминах элементарных функций и констант;

◦ выразить значения элементарных функций в терминах известных констант и арифметических операций;

◦ упростить вид аргументов.

Мы уже видели, что `Mathematica` не преобразует $\sin(\pi/8)$ к какому-либо другому виду, так как не считает, что, скажем, выражение этого значения в радикалах *проще*, чем $\sin(\pi/8)$. Однако выражение этого значения в радикалах "элементарнее":

```
In[74]:=FunctionExpand[{Cos[Pi/8],Sin[Pi/8]}]
Out[74]={Sqrt[2+Sqrt[2]]/2,Sqrt[2-Sqrt[2]]/2}
```

Для всех более сложных случаев поверх `FunctionExpand` рекомендуется применять `Simplify`:

```
In[75]:=Simplify[FunctionExpand[Cos[2*Pi/7]]]
Out[75]=-1/6+1/6*(7/2*(1-3*I*Sqrt[3]))^(1/3)+
                (I*(7/2*(1-3*I*Sqrt[3]))^(2/3))/(3*I+9*Sqrt[3])
```

Если ответ все еще представляется Вам сложным, то только потому, что Вы не видели, что `FunctionExpand` возвратило до применения `Simplify`!!

Команда `FunctionExpand` может применяться и к значениям числовых функций в случае символьных аргументов, удовлетворяющих определен-

ным предположениям. В этом случае она вызывается в том же формате, что и команда `Refine`:

```
FunctionExpand[expression,assumptions]
```

Однако она действует совершенно иначе, чем `Refine`. Это очень хорошо видно на следующем примере:

```
In[76]:=Refine[Log[x*y],x>0&&y>0]
Out[76]=Log[x*y]
In[77]:=FunctionExpand[Log[x*y],x>0&&y>0]
Out[77]=Log[x]+Log[y]
```

• **Структурные манипуляции.** В ядре системы описаны основные структурные манипуляции в кольцах $\mathrm{Trig}_{\mathbb{R}}$ и $\mathrm{Trig}_{\mathbb{C}}$ вещественных и комплексных тригонометрических многочленов. Вот некоторые простейшие из них:

◦ `TrigExpand` представляет тригонометрический многочлен как линейную комбинацию одночленов одночленов $\cos(x)^m \sin(x)^n$;

◦ `TrigFactor` раскладывает тригонометрический многочлен на множители;

◦ `TrigReduce` приводит тригонометрическое выражение к наиболее простому с точки зрения системы виду.

Возьмем функцию `Cos[2*x]*Cos[3*x]` и посмотрим, что с ней делают эти преобразования:

```
In[78]:=TrigExpand[Cos[2*x]*Cos[3*x]]
Out[78]=Cos[x]/2+Cos[x]^5/2-5*Cos[x]^3*Sin[x]^2+5/2*Cos[x]*Sin[x]^4
In[79]:=TrigFactor[Cos[2*x]*Cos[3*x]]
Out[79]=2*Cos[x]*(-1+2*Cos[2*x])*Sin[Pi/4-x]*Sin[Pi/4+x]
In[80]:=TrigReduce[Cos[2*x]*Cos[3*x]]
Out[80]=1/2*(Cos[x]+Cos[5*x])
```

При решении уравнений в элементарных функциях предварительная обработка уравнений этими преобразованиями обычно гораздо эффективнее, чем непосредственное применение команды `Solve`. Скажем, непосредственное вычисление

```
In[81]:=Solve[(Sin[x]+Cos[x])/Sqrt[2]+Cos[2*x]/Sqrt[3]==1,x]
```

(пример, фактически предлагавшийся на вступительном экзамене по математике на экономический факультет) дает ответ, но это такой ответ, который сам не посмотришь и другим не покажешь. Конечно, правильная стратегия состоит в проведении следующего вычисления:

```
In[81]:=TrigFactor[(Sin[x]+Cos[x])/Sqrt[2]+Cos[2*x]/Sqrt[3]]
Out[81]=((3*Sqrt[2]+2*Sqrt[6]*Sin[Pi/4-x])*Sin[Pi/4+x])/
                  (3*Sqrt[2])
```

после чего решения уравнения непосредственно очевидны.

• **Тригонометрические преобразования.** К сожалению, довольно часто приходится вручную контролировать, какие именно преобразования проводятся с тригонометрическими выражениями. К этому приходится прибегать для функций больших степеней, сложного вида аргументов, а также в тех случаях, когда Вы хотите увидеть ответ в каком-то определенном виде. Дело в том, что система всегда может породить *какой-то* ответ, но может случиться, что он будет выражен не в той форме, которая Вам нужна. Допустим, мы хотим выразить $\mathrm{tg}(x+y+z)$ в терминах $\mathrm{tg}(x)$, $\mathrm{tg}(y)$ и $\mathrm{tg}(z)$. Непосредственное применение $\mathrm{tg}(x+y+z)$ команды `FunctionExpand` приведет к нескольким строчкам косинусов и синусов $x, y$ и $z$, от которых начинает рябить в глазах. В этом месте полезно вспомнить команду `Together`, приводящую дроби к общему знаменателю:

```
In[82]:=Together[TrigExpand[Tan[x+y+z]]]
Out[82]=(Cos[y]*Cos[z]*Sin[x]+Cos[x]*Cos[z]*Sin[y]+
         Cos[x]*Cos[y]*Sin[z]-Sin[x]*Sin[y]*Sin[z])/
         (Cos[x]*Cos[y]*Cos[z]-Cos[z]*Sin[x]*Sin[y]-
         Cos[y]*Sin[x]*Sin[z]-Cos[x]*Sin[y]*Sin[z])
```

Но это все еще не то, на что мы надеялись. Не приводит к успеху ни применение `FunctionExpand`, `Refine` или `Simplify`. В этом случае нам не остается ничего другого, кроме как явно задавать **правила преобразования**, которые мы хотим применить. Мы подробно описывает йогу замен, правил и подстановок в Модуле 2. Пока ограничимся замечанием, что применение правила `lhs->rhs` к выражению `expression` оформляется в виде:

```
expression /.  lhs->rhs
```

если мы хотим однократно применить это правило ко всем частям выражения и в виде

```
expression //.  lhs->rhs
```

если мы хотим, чтобы система применяла правило не только к исходному выражению, но и к получающимся выражениям `quantum satis`, пока выражение не перестает меняться. Первая из этих форм называется `ReplaceAll`, а вторая — `ReplaceRepeated`.

При этом само правило преобразования `lhs->rhs` задается в формате

```
f[x_,y_]->g[x,y]
```

где $f(x,y)$ — функция, которую мы хотим переписать в виде $g(x,y)$. Обратите особое внимание на **бланки** `_` в левой части!!! Эти бланки делают $x$ и $y$ немыми переменными и сообщают системе, что это правило должно применяться, если подставить сюда вместо $x$ и $y$ любые символы и/или численные значения. Пропуск бланков является грубейшей синтаксической ошибкой. В этом случае система будет знать, что $f(x,y)$ следует заменить на $g(x,y)$, но это ее знание не будет распространяться на $f(a,b)$, $f(z,w)$ и т.д.

Проиллюстрируем задание правил преобразования на примере. Допустим, мы не знаем, что команда `TrigFactor` автоматически переписывает

$\operatorname{ctg}(x) - \operatorname{ctg}(y)$ в виде $-\sin(x-y)/(\sin(x)\sin(y))$ и хотим задать соответствующее правило преобразования. В этом случае мы можем задать его в виде

```
Cot[x_]-Cot[y_]->-Sin[x-y]/(Sin[x]*Sin[y])
```

Вернемся теперь к разложению $\operatorname{tg}(x+y+z)$ и заставим систему преобразовывать $\operatorname{tg}(x+y)$ в $(\operatorname{tg}(x)+\operatorname{tg}(y))/(1-\operatorname{tg}(x)\operatorname{tg}(y))$ столько раз, сколько она видит выражение такого вида:

```
In[83]:=Together[Tan[x+y+z] //.
                 Tan[x_+y_]->(Tan[x]+Tan[y])/(1-Tan[x]*Tan[y])]
Out[83]=(-Tan[x]-Tan[y]-Tan[z]+Tan[x]*Tan[y]*Tan[z])/
        (-1+Tan[x]*Tan[y]+Tan[x]*Tan[z]+Tan[y]*Tan[z])
```

После приобретения некоторого опыта правила преобразования становятся одним из основных инструментов программирования на языке Mathematica, позволяющим полностью контролировать вид ответа.

• **Экспоненциальная и тригонометрическая форма.** Функции $x \mapsto \cos(ax)$ и $x \mapsto \sin(ax)$ порождают то же пространство, что $x \mapsto e^{iax}$ и $x \mapsto e^{-iax}$. Следующие команды осуществляют пересчет из тригонометрического базиса в экспоненциальный и наоборот:

∘ `TrigToExp` — преобразование выражения из тригонометрической формы в экспоненциальную;

∘ `ExpToTrig` — преобразование выражения из экспоненциальной формы в тригонометрическцю.

В следующих вычислениях мы преобразуем тригонометрическое выражение в экспоненциальную форму:

```
In[84]:=FactorTerms[TrigToExp[Cos[2*x]*Cos[3*x]]]
Out[84]=1/4*(E^(-I*x)+E^(I*x)+E^(-5*I*x)+E^(5*I*x))
In[85]:={TrigToExp[Tan[x]],TrigToExp[Cot[x]]}
Out[85]={(I*(E^(-I*x)-E^(I*x)))/(E^(-I*x)+E^(I*x)),
        -(I*(E^(-I*x)+E^(I*x)))/(E^(-I*x)-E^(I*x))}
```

Функция `FactorTerms` применена, чтобы вынести общий численный множитель.

Вот еще один очень интересный пример. По умолчанию команды типа `Solve` записывают первообразные корни из 1 степени 5 в экспоненциальном виде:

```
In[86]:=Solve[x^4+x^3+x^2+x+1==0,x]
Out[86]={{x->-(-1)^(1/5)},{x->(-1)^(2/5)},
        {x->-(-1)^(3/5)},{x->-(-1)^(4/5)}}
```

Однако большинство начинающих предпочтут увидеть их в тригонометрическом виде:

```
In[87]:=Map[ExpToTrig,Solve[x^4+x^3+x^2+x+1==0,x],3]
Out[87]={{x->-1/4-Sqrt[5]/4-1/2*I*Sqrt[1/2*(5-Sqrt[5])]}
```

```
{{x->-1/4+Sqrt[5]/4+1/2*I*Sqrt[1/2*(5+Sqrt[5])]}
{{x->-1/4+Sqrt[5]/4-1/2*I*Sqrt[1/2*(5+Sqrt[5])]}
{{x->-1/4-Sqrt[5]/4+1/2*I*Sqrt[1/2*(5-Sqrt[5])]}
```

Команда `Map`, вызванная в формате

```
Map[f,list,d]
```

применяет функцию $f$ к списку `list` на уровне $d$. В приведенном выше примере `ExpToTrig` применяется на уровне 3. На уровне 3 в выражении, получающемся после применения `Solve`, лежат $x$ и собственно корни из 1. Однако ясно, что с $x$ команда `ExpToTrig` ничего не делает.

• **Экстремумы функции.** При помощи системы `Mathematica` можно провести "исследование функций". Упомянем лишь часто встречающуюся задачу отыскания максимумов и минимумов. Основными командами для этого являются `Maximize` и `Minimize`. Так как их использование абсолютно аналогично, в дальнейшем мы будем говорить только о `Maximize`. Для поиска глобального максимума можно вызывать эту команду в формате

```
Maximize[f[x],x]
```

При этом ответ возвращается в формате `{a,{x->c}}` с указанием максимума $a$ функции $x \mapsto f(x)$ и *какого-то* (обычно наименьшего) значения аргумента $c$, в котором этот максимум достигается.

Разумеется, глобальный максимум не обязан существовать или может быть бесконечным. Например, при попытке вычислить `Maximize[x^2,x]` мы получим сообщение

```
Maximize:  The maximum is not attained at any point
                  satisfying the given constraints.
```

и ответ `{Infinity,{x->-Infinity}}`.

В действительности гораздо чаще приходится производить не глобальную оптимизацию, а оптимизацию с **ограничениями** `constraints`. Для этого функции `Maximize` и `Minimize` вызываются в формате

```
Maximize[{f[x],constraints},x]
```

где `constraints` обозначает ограничения или условия, при которых ищется экстремум. Вот, скажем, как ищется максимум функции $f$ на отрезке $[a,b]$:

```
Maximize[{f[x],a<=x<=b},x]
```

Однако уже для очень простых трансцендентных функций явная оптимизация может приводить к довольно сложным ответам:

```
In[88]:=FullSimplify[Maximize[{Sqrt[x]-Exp[x],0<=x<=2},x]]
Out[88]={(-1+ProductLog[1/2])/Sqrt[2*ProductLog[1/2]],
                  {x->1/2*ProductLog[1/2]}}
```

Встречающаяся здесь функция `ProductLog[x]` представляет собой главное решение уравнения $ye^y = x$, удовлетворяющее дифференциальному уравнению $\dfrac{dy}{dx} = \dfrac{y}{x(1+y)}$. Эта функция очень часто возникает при решении уравнений, в которые входят экспонента и/или логарифм.

В данном случае нам крупно повезло, что нашлась функция, решающая возникающее при оптимизации уравнение. Однако в большинстве случаев, сводящихся к решению трансцендентных уравнений, система будет не в состоянии найти точные максимумы и минимумы. Например, попытка вычислить

```
In[89]:=Maximize[{Log[x]+Sin[x],0<=x<=5},x]
```

не даст никакого вразумительного ответа. В подобных случаях можно применять команды численной оптимизации `NMaximize` и `NMinimize`. Эти команды вызываются точно в таком же формате, как команды `Maximize` и `Minimize` и дают приближенные значение экстремума и той точки, в которой оно достигается:

```
In[90]:=NMaximize[{Log[x]+Sin[x],0<=x<=5},x]
Out[90]={1.60552,{x->2.07393}}
```

## § 6. Графики функций

Die Mathematik ist vielmehr eine Wissenschaft für das Auge als eine für das Ohr. — Математика в гораздо большей степени обращается к глазу, чем к уху.

Carl Friedrich Gauß

Und was man sieht und abzeichnet ist in der Regel immer schöner als was man nur so selbst erfindet. — И то, что срисовываешь с натуры, всегда получается лучше того, что пытаешься высосать из пальца.

Wilhelm Hauff

Из-за искажения масштаба дисплеем компьютера окружность выглядит как эллипс.

Владимир Дьяконов[30]

A conjecture both deep and profound
Is whether the circle is round;
In a paper by Erdös,
written in Kurdish,
A counterexample is found.[31]

В действительности с точки зрения профессионала одной из самых сильных сторон системы `Mathematica`, являются огромные возможности **визуализации** вычислений. С практической точки зрения эти возможности ограничены *только* способностью пользователя призывать их. Дело в том,

[30] В настоящей книге встречаются изредка неправильно истолкованные не цитаты (`misconstrued misquotations`), но именно этот эпиграф самый что ни на есть *подлинный*!!! Тот, кто думает, что мы его снова разыгрываем, может сам взглянуть на рисунок и легенду к нему на странице 323 следующей книги: В.П.Дьяконов, `Mathematica 4`. — Питер, 2001, с.1–654.

[31] Окружность я не рисовал, я с детства рисовал овал. — Поль Эрдеш (перевод с курдского).

что хотя картинки произвольно высокой сложности могут быть порождены чрезвычайно простыми формулами, способность фактически установить полное соответствие между вычислениями и графикой требует, как минимум,

- ◦ хорошего понимания математики, стоящей за этими формулами,
- ◦ свободного владения функциональным программированием,
- ◦ собственно навыка применения графических команд, настройки их опций и т.д.,

которые приобретаются только в результате длительных упражнений. Поэтому в настоящем параграфе мы опишем только применение небольшого числа встроенных команд для построения графиков функций одной и двух переменных — и то только в *простейших* вариантах.

• **Построение графиков.** График функции строится при помощи команды `Plot`, которая порождает объект формата `Graphics`. В простейшем варианте, когда Вы доверяете системе принять вместо Вас **все** эстетические решения, эта команда вызывается в формате:

```
Plot[f[x],{x,a,b}]
```

Эта команда порождает график функции $f(x)$, когда $x$ меняется от $a$ до $b$. Однако в действительности, у команды `Plot` еще несколько десятков факультативных аргументов, называемых **опциями**. Все они вместе с их значениями по умолчанию могут быть получены командой

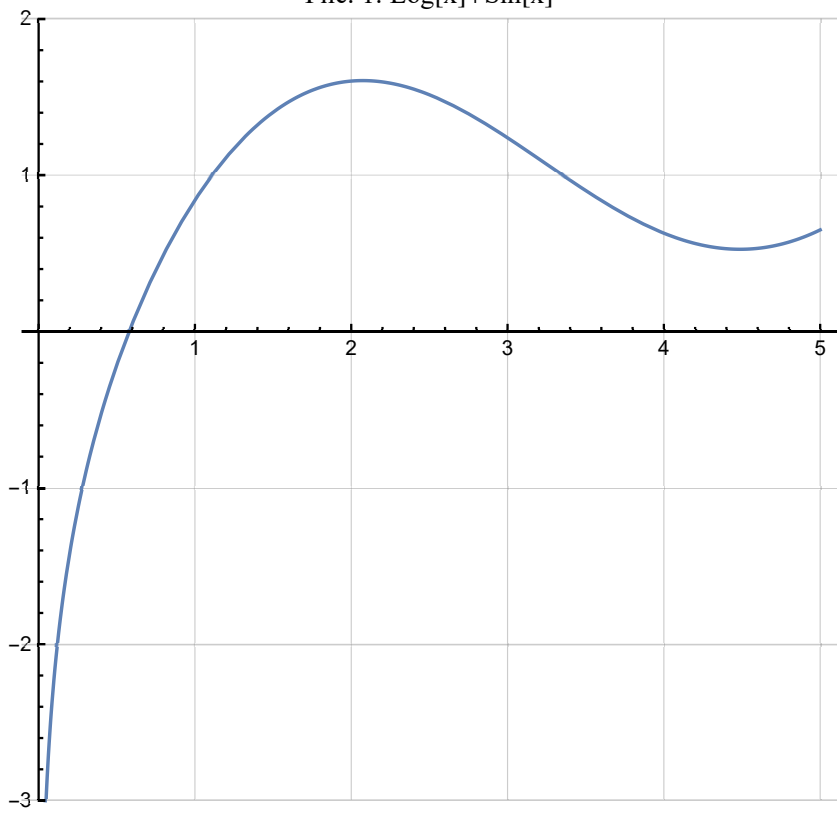
```
In[91]:=Options[Plot]
```

Мы не будем обсуждать использование большинства из этих опций. Но для того, чтобы увидеть то, что Вам хочется, и так, как Вам этого хочется, поучимся настраивать самые важные: *хотя бы* `AspectRatio` и `PlotRange`, может быть, еще `PlotStyle`, `Axes`, `AxesStyle`, `Ticks`, `GridLines`, `PlotLabel` и `TextStyle`. Действительно, без понимания того, как работают `AspectRatio` и `PlotRange` — как наглядно продемонстрировал В.П.Дьяконов — не удастся узнаваемо нарисовать ровным счетом ничего, даже окружность.

Вот как выглядит типичный вызов функции `Plot`. На рисунке 1 мы видим график функции $x \mapsto \ln(x) + \sin(x)$, построенный при помощи следующей команды:

```
In[92]:=Plot[Log[x]+Sin[x],{x,0,5},
                AspectRatio->Automatic,
                PlotRange->{-3,2},
                PlotStyle->AbsoluteThickness[1.5],
                AxesStyle->AbsoluteThickness[1],
                GridLines->Automatic,
                PlotLabel->"Рис.  1: Log[x]+Sin[x]"]
```

Рис. 1: Log[x]+Sin[x]




Разумеется, мы получили бы график той же функции и напечатав просто

In[93]:=`Plot[Log[x]+Sin[x],{x,0,5}]`

без всяких там опций. Но это был бы совсем не такой красивый и информативный график, поэтому поясним использование опций.

○ `AspectRatio` задает **форматное отношение**, т.е. отношение высоты рисунка к его ширине. Например, `AspectRatio->2` означает, что высота в два раза больше ширины, а `AspectRatio->1/2` — что ширина в два раза больше высоты. Форматное отношение по умолчанию равно `1/GoldenRatio`. Это означает, что рисунок имеет те же пропорции, что стандартный лист бумаги формата `A4` в **альбомной ориентации** (`landscape orientation`). В то же время, отношение равное `GoldenRatio` означало, бы, что рисунок имеет пропорции стандартного листа бумаги формата `A4` в **книжной ориентации** (`portrait orientation`). Мы обычно полагаем

⋆ либо `AspectRatio->1` — в этом случае рисунок вписывается в квадрат;

⋆ либо `AspectRatio->ysize/xsize`, где `ysize` равно разности концов отрезка `PlotRange`, а `xsize=b-a` — в этом случае масштаб по осям одинаков;

⋆ либо `AspectRatio->Automatic` — в этом случае система сама решает, что больше соответствует обстановке, и во всех обычных случаях пытается задать одинаковый масштаб по осям. Конечно, это не всегда получается. Например, если функция $f$ очень быстро растет или убывает, то ее значения могут превосходить значения $x$ в тысячи раз.

∘ PlotRange задает **диапазон графика**, т.е. те значения $y$, которые на нем отображаются. Когда функция слишком быстро растет или убывает, система не всегда удачно выбирает значение PlotRange и отсекает наиболее интересные части графика. Поэтому часто приходится задавать диапазон вручную:

⋆ PlotRange->All пытается поместить на график **все** значения функции $f$ при $x$ меняющемся в заданных пределах;

⋆ PlotRange->{c,d} предлагает команде строить только ту часть графика, которая помещается в горизонтальную полосу $c \le y \le d$.

Обратите внимание, что в предшествующем примере мы сознательно выбрали PlotRange так, чтобы ysize равнялось xsize. При этом установка AspectRatio->Automatic приводит к тому, что масштаб по осям одинаков.

∘ PlotStyle задает **стиль графика**, т.е. значения применяемых к нему **графических директив**. Типичными графическими директивами являются задаваемые в типографских точках директивы **ширина** = Thickness и **абсолютная ширина** = AbsoluteThickness, **штриховка** = Dashing и **абсолютная штриховка** = AbsoluteDashing, описывающие плотность и прерывистость кривой, а также такие директивы, как **уровень серого** = GrayLevel, меняющийся от 0 (черный) до 1 (белый) и многочисленные директивы управления цветом: **тон** = Hue, задаваемый списком трех координат HSB (Hue, Saturation, Brightness) и **цвет** = RGBColor, задаваемый списком трех координат RGB (Red, Green, Blue), etc., etc.

В данном случае посредством PlotStyle->AbsoluteThickness[1.5] мы задали ширину кривой, приблизительно отвечающую насыщенности жирного шрифта. По умолчанию ширина соответствует насыщенности обычного шрифта.

∘ AxesStyle задает **стиль осей**, т.е. значения графических директив, применяемых к осям. В этот момент читатель должен догадаться, что напечатав AxesStyle->AbsoluteThickness[1], мы добились того, чтобы оси имели ширину в 1 пункт, чуть меньшую, чем насыщенность полужирного шрифта.

∘ GridLines задает **координатную сетку**, т.е. прямые, параллельные координатным осям. По умолчанию GridLines->None, так что никаких прямых, кроме самих осей не проводится.

⋆ GridLines->Automatic — прямые проводятся через **метки** (Ticks) на осях;

⋆ GridLines->{{x1,...,xm},{y1,...,yn}} — прямые, параллельные оси $y$, проводятся через точки $x_1, \ldots, x_m$ на оси $x$, а прямые, параллельные оси $x$ — через точки $y_1, \ldots, y_n$ на оси $y$.

В данном случае мы задали опцию GridLines->Automatic, что побудило систему проводить прямые через целые точки на осях.

∘ PlotLabel задает **ярлык графика**, т.е. заголовок или название, которое обычно пишется *над* графиком. Если Вам хочется перевести Label

как *метка*, не торопитесь, так как **метка** или **засечка** является стандартным переводом термина `Tick`, который встретится нам далее. Другими близкими опциями являются

⋆ **ярлыки осей** = `AxesLabel` — текст, который пишется на осях,

⋆ **ярлыки рамки** = `FrameLabel` — текст, который пишется на рамке,

⋆ **легенда** = `PlotLegend` — помещенные в рамочку пояснения под графиком.

Обратите внимание, что включенный в задание опции `PlotLabel` текст `"Рис. 1: Log[x]+Sin[x]"` заключается в кавычки. Это превращает этот текст из последовательности символов в **стринг** = `String`, текстовый объект, воспринимаемый и воспроизводимый `verbatim` (буква в букву). Над стрингом не производятся *никакие* обычные вычисления. Иными словами, все специальные символы, которые в иных условиях интрепретировались бы как операторы, в составе стринга представляют собой просто типографские знаки. Следует иметь ввиду, что имеются специальные **текстовые команды**, при помощи которых проводятся преобразования стрингов.

**Комментарий.** Отметим, что те картинки, которые Вы видите в этой книге, порождены при помощи несколько более сложного выбора опций. Дело в том, что при подготовке оригинал-макета книги нам еще нужно было привести в соответствие встречающийся в ярлыках, метках и легендах шрифт в соответствие со шрифтом, использованным в основном тексте. Текст этой книги набран с использованием гарнируры `Times New Roman`, в то время как стандартные настройки `Mathematica` порождают вывод шрифтом `Courier`. Это значит, что в действительности для задания ярлыков нам приходилось печатать что-то в таком духе:

```
PlotLabel->StyleForm["Рис. 1: Log[x]+Sin[x]",
                FontFamily->"Times", FontWeight->"Bold", FontSize->12]
```

С целью изменения шрифта меток пришлось включать в тело команды опцию

```
TextStyle->{FontFamily->"Times",FontSize->12}
```

Обратите внимание на задание опции внутри опции!!! Понятно, что все эти вещи делались исключительно из типографских соображений и при проведении реальных вычислений никому не следует подобными вещами злоупотреблять.

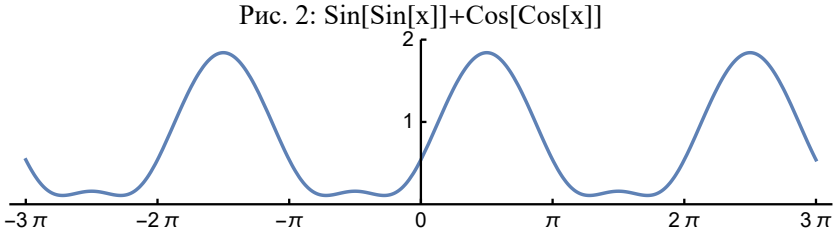
• **Основная формула тригонометрии.** Во многих школьных учебниках упоминается “основная формула тригонометрии”, которая записывается в форме $\cos^2(x) + \sin^2(x) = 1$. Однако каждый может моментально проверить, что в таком виде эта формула БЕЗНАДЕЖНО НЕВЕРНА. В этом легко убедиться, например, заметив, что

$$\cos^2(\pi/2) + \sin^2(\pi/2) = 1 + \sin(1) \neq 1.$$

Для того, чтобы развеять все вредные иллюзии, изобразим на рисунке 2 график функции $x \mapsto \cos^2(x) + \sin^2(x)$, построенный при помощи следующей команды:

```
In[94]:=Plot[Sin[Sin[x]]+Cos[Cos[x]],{x,-3*Pi,3*Pi},
                PlotRange->{0,2},
                AspectRatio->1/5,
```

```
Ticks->{Table[n*Pi,{n,-3,3}],{1,2}},
AxesStyle->AbsoluteThickness[1],
PlotStyle->AbsoluteThickness[1.5],
PlotLabel->"Рис.  2: Sin[Sin[x]]+Cos[Cos[x]]"]
```




Повторимся, что мы могли получить *почти* такой же результат и напечатав просто

In[95]:=`Plot[Sin[Sin[x]]+Cos[Cos[x]],{x,-3*Pi,3*Pi}]`

без всяких опций. Все остальное — это декорации и отделка деталей.

Большая часть этого текста должна быть Вам уже знакома после разбора предыдущего примера. Обратите внимание на то, что порядок задания опций не имеет значения. В предыдущем примере мы вначале определили `AspectRatio`, а потом `PlotRange`, но никто не мешает нам задавать их в другой последовательности. Кроме того, в этом примере встречается новая опция

∘ `Ticks` — это **метки** или **засечки**, т.е. характерные точки, отмечаемые на осях. По умолчанию `Ticks->Automatic` и система сама отмечает такие точки. Однако возможны другие установки:

⋆ `Ticks->None` — метки на осях не ставятся;

⋆ `Ticks->{{x1,...,xm},{y1,...,yn}}` — метки на оси $x$ ставятся в точках $x_1, \dots, x_m$, а на оси $y$ — в точках $y_1, \dots, y_n$.

В данном случае нам показалось, что для этого графика на оси $x$ гораздо естественнее отмечать целые кратные $\pi$, а вовсе не целые числа. При этом, чтобы чуть сократить ввод с клавиатуры и облегчить возможность дальнейших изменений, мы породили множество целых кратных $\pi$ как список посредством команды `Table`, см. § 10 настоящей главы.

То соотношение между основными тригонометрическими функциями, которое действительно имеет место, это не какая-то мифическая “основная формула тригонометрии”, а **теорема Пифагора** $\cos(x)^2 + \sin(x)^2 = 1$. Мы надеемся, что после подобной наглядной демонстрации того, к каким грубым ошибкам неизбежно приводят неряшливые обозначения, читатель будет тщательно различать возведение в квадрат *функции* и возведение в квадрат *ее значения*!

Если к этому моменту Вам надоело печатать при построении каждого графика `AspectRatio->1` или `AspectRatio->Automatic`, то Вы абсолютно правы. В действительности, если Вы на протяжении сессии собираетесь

строить графики нескольких функций, имеет смысл с самого начала изменить опции команды `Plot` и/или других используемых Вами графических команд. Например, для того, чтобы во всех графиках, которые Вы строите на протяжении сессии, форматное отношение стало *по умолчанию* равным 1, а метки на осях не ставились, нужно лишь один раз в начале сессии произвести следующее вычисление:

```
In[96]:=SetOptions[Plot,AspectRatio->1,Ticks->None]
```

Смысл этого вычисления состоит в том, что при этом опциям `AspectRatio` и `Ticks` присваиваются новые значения, отличные от задаваемых по умолчанию, и эти новые значения действуют на протяжении всей сессии, пока не будут снова изменены или переопределены.

• **Несколько функций на одном графике.** Особенно интересна возможность строить графики нескольких функций на одной картинке. Вызваная в формате

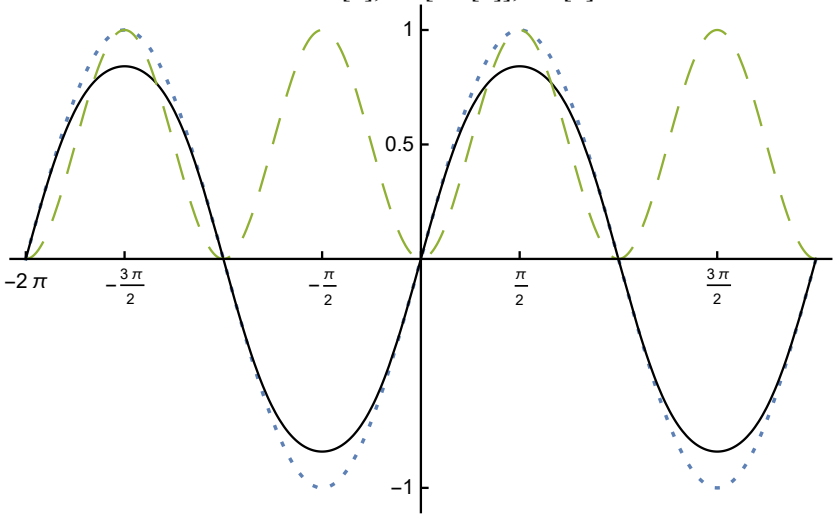
```
Plot[{f1[x],f2[x],...},{x,a,b}]
```

команда `Plot` *одновременно* строит графики функций $f_1(x), f_2(x), \ldots$ при $x$ меняющемся от $a$ до $b$.

Например, очень интересно сравнить графики функций sin, $\sin^2$ и $x \mapsto \sin(x)^2$. Это можно сделать при помощи следующей команды:

```
In[97]:=Plot[{Sin[x],Sin[Sin[x]],Sin[x]^2},{x,-2*Pi,2*Pi},
         PlotStyle->
             {{AbsoluteThickness[1.5], Dashing[{0.002,0.02}]},
             {AbsoluteThickness[1], GrayLevel[0]},
             {AbsoluteThickness[1], Dashing[{0.03}]}},
         AxesStyle->AbsoluteThickness[1],
         PlotLabel->"Рис.  3: Sin[x], Sin[Sin[x]], Sin[x]^2",
         Ticks->{{-2*Pi,-3*Pi/2,-Pi/2,0,Pi/2,3*Pi/2},
                                          {-1,0.5,1}}]
```

Рис. 3: Sin[x], Sin[Sin[x]], Sin[x]^2



Результат этого вычисления воспроизведен на Рисунке 3. Отметим лишь те моменты, которые нам раньше не встречались.

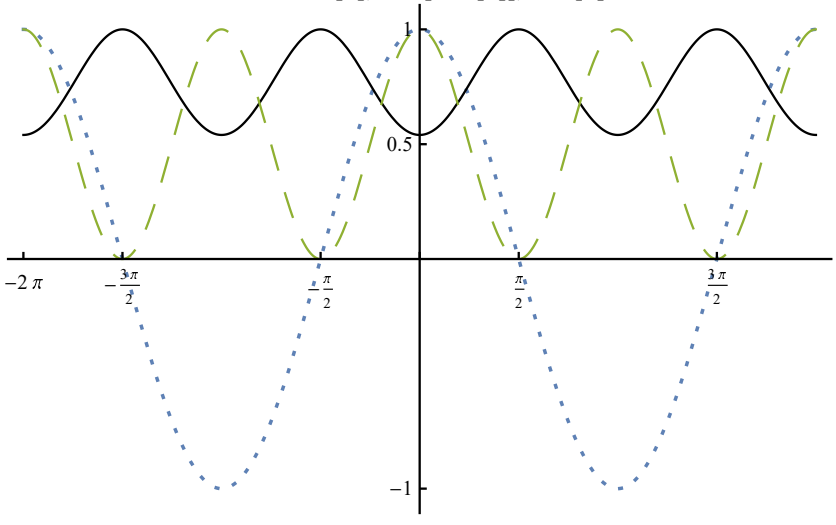
◦ Значения графических директив для списка функций тоже могут задаваться списком, при этом первый элемент этого списка будет применяться к первой функции, второй — ко второй и т.д. Например, в данном случае мы рисуем график первой функции с толщиной линии 1.5, а график второй и третьей — с толщиной 1. Это делается для того, чтобы они производили впечатление одинаковой жирности.

◦ Директива `Dashing` описывает `штриховку`. При этом ее аргумент задается в виде списка, элементы которого определяют длины последовательных сегментов кривой, из которых нечетные закрашиваются, а четные — не закрашиваются. После исчерпания элементов списка длины начинают циклически повторяться. При этом, в отличие от `абсолютной штриховки` = `AbsoluteDashing`, эти длины указываются не в каких-то абсолютных величинах, а в долях от общего размера графика. Таким образом, `Dashing[{0.002,0.02}]` описывает кривую, составленную из точек, расстояния между которыми в десять раз больше размера самих точек, а `Dashing[{0.03}]` — кривую, состоящую из закрашенных и незакрашенных сегментов одинаковой длины.

◦ Наконец, директива `GrayLevel[0]` предписывает рисовать вторую кривую, а именно, график функции $\sin^2$, сплошной черной линией.

**Упражнение.** Нарисуйте картинку, воспроизведенную на Рисунке 4.

Рис. 4: Cos[x], Cos[Cos[x]], Cos[x]^2




**Указание.** Обратите внимание на ярлык и метки!

Стоит подчеркнуть, что штриховка здесь использована исключительно из типографских соображений. В обычных условиях для вывода картинки на экран или цветной принтер мы бы варьировали бы не штриховку, а **цвет** кривых. Иными словами, мы задавали бы опцию `PlotStyle` следующим образом:

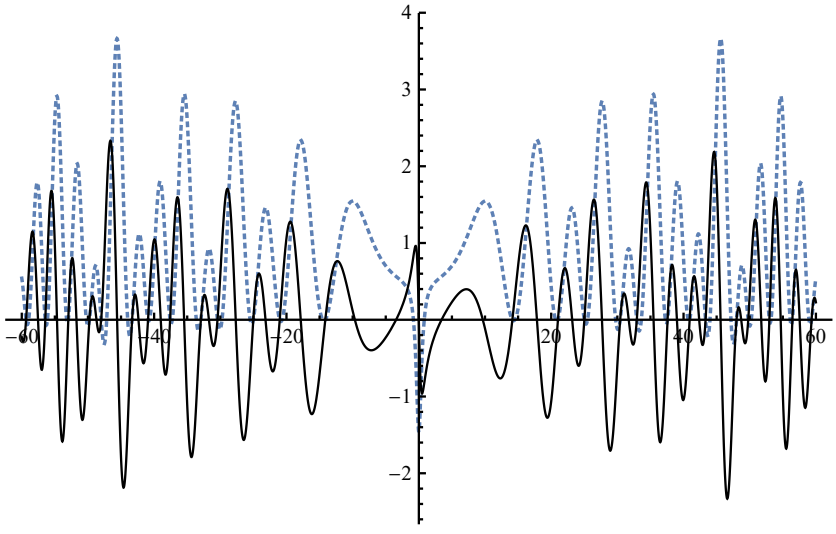
```
PlotStyle->{RGBColor[1,0,0],RGBColor[0,1,0],RGBColor[0,0,1]}
```

При этом первая кривая изображается чистым красным цветом (в самом деле, `RGBColor[1,0,0]` означает *максимальную* насыщенность красного и *нулевую* насыщенность зеленого и синего), вторая — зеленым и третья — синим. Конечно, проще печатать, используя системные названия цветов

```
PlotStyle->{Red,Green,Blue}
```

Еще одна интересная представляющаяся здесь возможность состоит в том, чтобы строить на одном графике вещественную и мнимую часть комплексной функции вещественной переменной. Например, на Рисунке 5 изображены графики вещественной и мнимой части дзета-функции Римана на критической прямой, причем вещественная часть изображена пунктиром, а график мнимой части — сплошной.

Рис 5: Riemann Zeta




Этот рисунок порожден при помощи следующего текста:

```
In[98]:=Plot[{Re[Zeta[1/2+x*I]],Im[Zeta[1/2+x*I]]},{x,-60,60},
         PlotStyle ->
             {{AbsoluteThickness[1.5],AbsoluteDashing[{1,3}]},
             {AbsoluteThickness[1],GrayLevel[0]}},
         AxesStyle->AbsoluteThickness[1],
         PlotLabel->"Рис.  5: Riemann Zeta"
         Ticks -> Automatic]
```

Все в этом тексте уже нам встречалось, за исключением ровно одного момента:

◦ Для задания штриховки мы пользуемся не директивой `Dashing`, а директивой **абсолютная штриховка** = `AbsoluteDashing`, аргумент которой представляет собой список длин последовательных сегментов кривой, задаваемых в типографских пунктах. Таким образом, по замыслу график вещественной части состоит из точек размером 1 типографский пункт, разделенных промежутками в три типографских пункта. Однако из-за того,

что мы задали абсолютную толщину кривой в 1.5 пункта, точки выглядят больше, чем расстояния между ними.

Можно, конечно, различить вещественную и мнимую части не штриховкой, а цветом, например, так:

```
PlotStyle->{RGBColor[1,0,0],RGBColor[0,0,1]}
```

В этом случае на экране компьютера график вещественной части будет красным, а график мнимой части — синим.

Стоит предупредить читателя об одной существенной тонкости. Дело в том, что команда `Plot` необычным образом вычисляет свой первый аргумент. Если быть совсем точным, она его вообще никак не вычисляет: специальный параметр HoldAll предписывает вычислять первый аргумент только после присвоения конкретных числовых значений переменной `x`. Это может приводить к ощутимым затратам времени, в частности, если первый аргумент является списком функций, например, при построении на одном графике 10 первых многочленов Чебышева командой

```
Plot[Table[ChebyshevT[n, x],{n,1,10}],{x,-1.01,1.01}]]
```

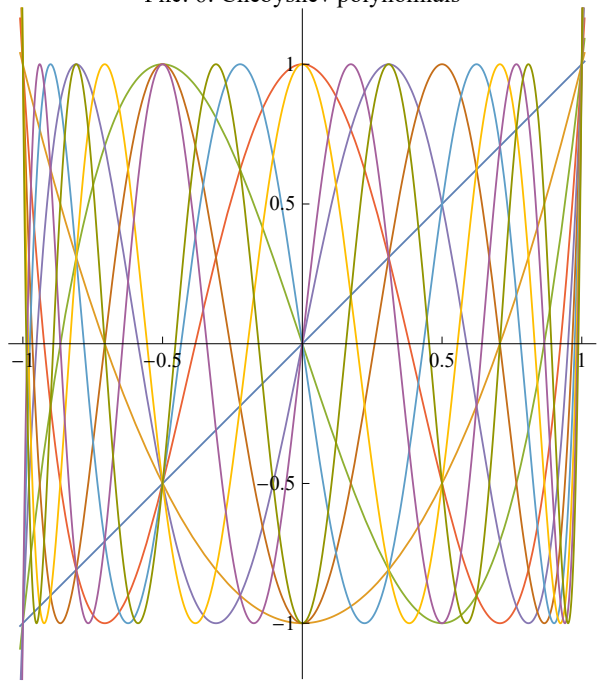
В подобных случаях полезно использовать директиву Evaluate, которая отменяет действие указанного параметра, что сокращает время, затрачиваемое на построение графика. С её использованием построение графика таблицы функций может быть осуществлено следующим образом:

```
Plot[Evaluate[Table[f[i][x],{i,1,n}]],{x,xmin,xmax}]
```

Вот, например, как выполнено построение графика 10 первых многочленов Чебышева, представленное на рисунке 6:

```
In[98]:=Plot[Evaluate[Table[ChebyshevT[n,x],{n,1,10}]],
                                        {x,-1.01,1.01},
        AspectRatio->Automatic,
        PlotStyle->AbsoluteThickness[0.8],
        PlotRange->{-1.7,1.7},
        Ticks->{{-1,-0.5,0.5,1,1.5},{-1,-0.5,0.5,1}},
        PlotLabel->StyleForm["Рис.  6: Chebyshev polynomials"]]
```

Рис. 6: Chebyshev polynomials




Все остальные моменты, кроме только что объясненной необходимости использования команды `Evaluate`, уже встречались нам в предыдущих примерах.

• **Параметрический график.** Еще один важнейший способ построить объект формата `Graphics` состоит в том, чтобы использовать команду `ParametricPlot`. В простейшем варианте эта команда вызывается в следующем формате:

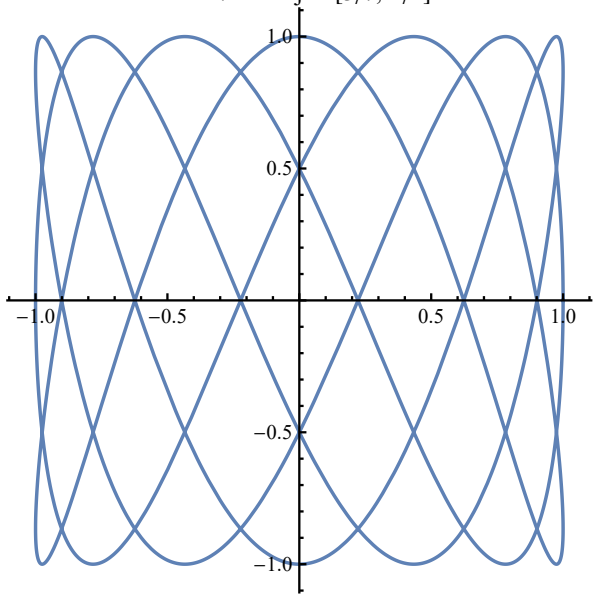
```
ParametricPlot[{f[t],g[t]},{t,a,b}]
```

Эта команда порождает графический объект, изображающий кривую, описываемую точкой с координатами $(f(t), g(t))$, когда $t$ меняется от $a$ до $b$.

Например, во многих областях естествознания возникают **фигуры Лиссажу**, т.е. траектории точки, совершающей гармонические колебания в двух ортогональных направлениях. Конкретный вид этой фигуры определяется периодами, разностью фаз и амплитудами колебаний. Например, на Рисунке 7 изображена фигура Лиссажу, отвечающая отношению периодов $3/7$ и разности фаз $\pi/2$:

```
In[99]:=ParametricPlot[{Cos[3*t],Sin[7*t]},{t,0,2*Pi},
                PlotStyle->AbsoluteThickness[1.5],
                AxesStyle->AbsoluteThickness[1],
                AspectRatio->Automatic,
                PlotLabel->"Рис.  7: Lissajout[3/7,Pi/2]"]
```

Рис. 7: Lissajout[3/7,Pi/2]




**Упражнение.** Постройте кривые Лиссажу, изображенные на рисунках 8–10:

Рис. 8: Lissajout[3/7,Pi/4]

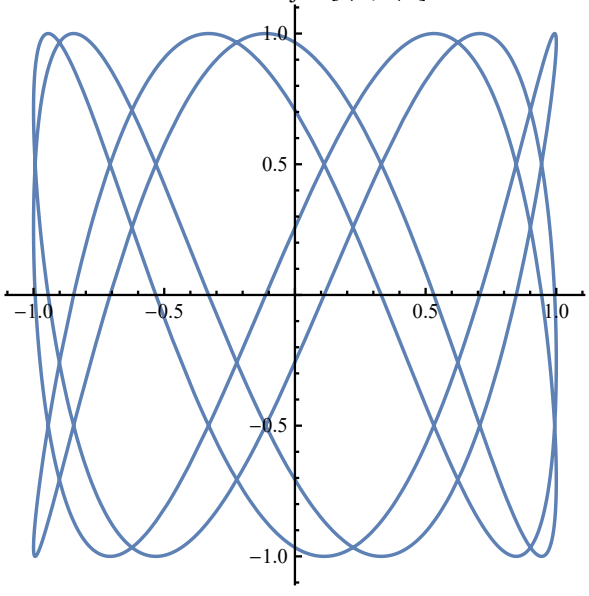



Рис. 9: Lissajout[5/7,Pi/4]

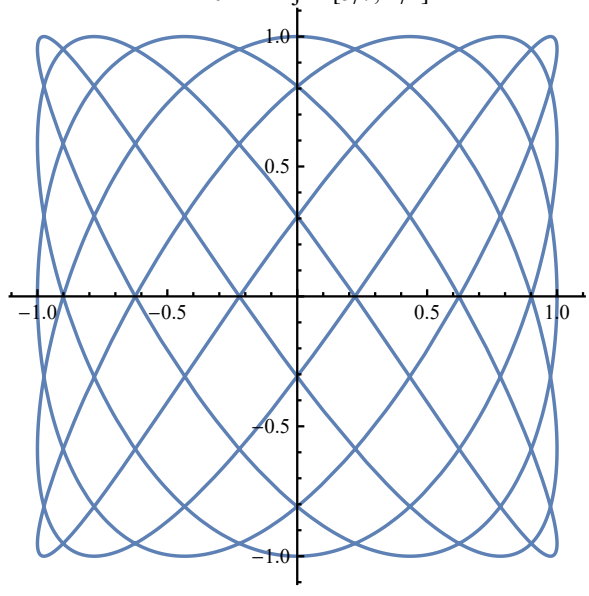
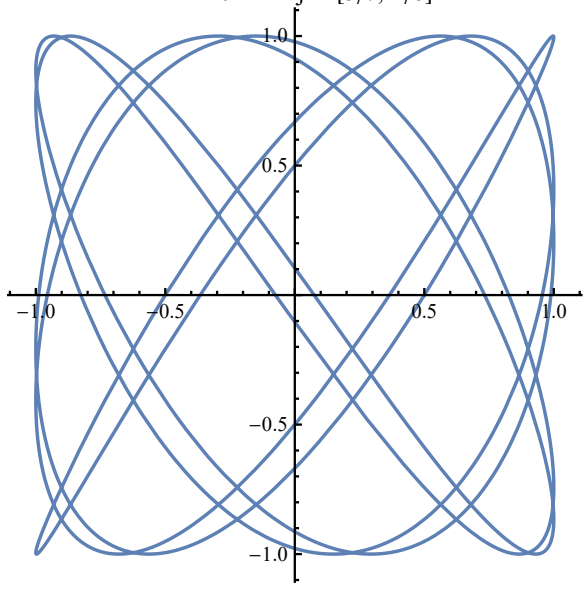


Рис. 10: Lissajout[5/7,Pi/6]




Как и в случае команды `Plot`, мы можем изобразить несколько параметрических кривых на одной картинке. Для этого команду `ParametricPlot` нужно вызывать в следующем формате:

```
ParametricPlot[{{f1[t],g1[t]},{f2[t],g2[t]},...},{t,tmin,tmax}]
```

Команда `ParametricPlot` имеет те же особенности, что и `Plot`. В частности, если Вы задаете ее первый аргумент неявным образом, который требует предварительной обработки до того как можно вычислять его значения, то к этому аргументу необходимо применить команду `Evaluate`.

• **График неявной функции.** Пожалуй еще более замечательной, чем возможность строить параметрические графики, является возможность построения графиков неявных функций. Это делается при помощи команды `ContourPlot`. Работа этой команды основана на способности системы решать системы уравнений (ее имплементация самым существенным образом взывает к команде `Solve` и др.) и, как и многие другие графические функции требует довольно значительного компьютерного ресурса.

Формат вызова команды для построения графика функции, заданной неявно, следующий:

```
ContourPlot[f[x,y]==g[x,y],{x,a,b},{y,c,d}]
```

где $f(x,y) = g(x,y)$ — уравнение, решения которого мы хотим изобразить на графике, $x$ меняется от $a$ до $b$, а $y$ меняется от $c$ до $d$.

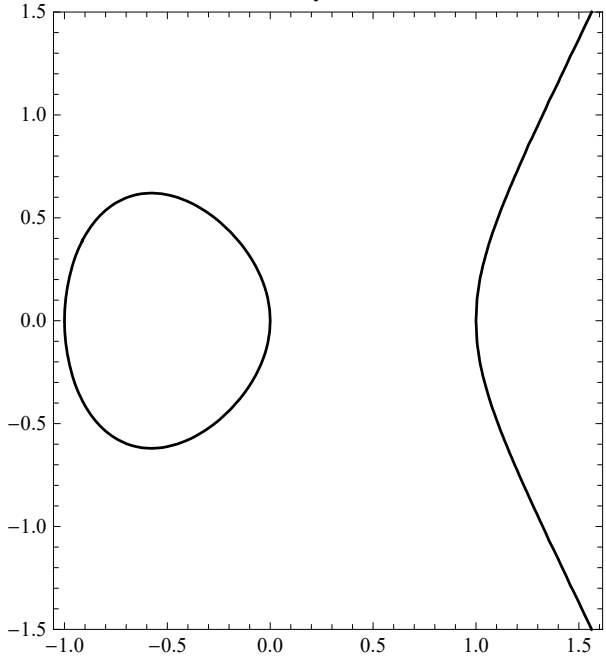
На рисунках 11–14 представлены **эллиптические кривые**. Для непосвященного читателя заметим, что они называются так вовсе не потому, что похожи на эллипс, а потому, что их изучение было первоначально мотивировано теорией **эллиптических интегралов** и **эллиптических функций**, которые, в свою очередь, действительно впервые появились в работе братьев Бернулли при вычислении длины дуги эллипса. В прошлое десятилетие эллиптические кривые стали чрезвычайно знамениты в широких

кругах образованной публики благодаря той роли, которую они сыграли в доказательстве последней теоремы Ферма Эндрю Уайлсом[32].

Вот, например, как строилась первая из этих кривых:

```
In[100]:=ContourPlot[y^2==x^3-x,{x,-1.2,1.6},{y,-1,2},
         AspectRatio->Automatic,
         ContourStyle -> {GrayLevel[0], Thickness[0.005]]},
         AxesStyle->AbsoluteThickness[1],
         PlotRange->{-1.7,1.7},
         Ticks->{{-1,-0.5,0.5,1,1.5},{-1,-0.5,0.5,1}},
         PlotLabel->"Рис.  11: y^2=x^3-x"]
```

Рис. 11: y^2=x^3−x




**Упражнение.** Постройте эллиптические кривые, изображенные на рисунках 12–14, обращая при этом внимание на детали!

[32] A.Wiles, Modular elliptic curves and Fermat's last theorem. — Ann. Math., 1995, vol.141, p.443–551.

Рис. 14: y^2=x^3−x+1

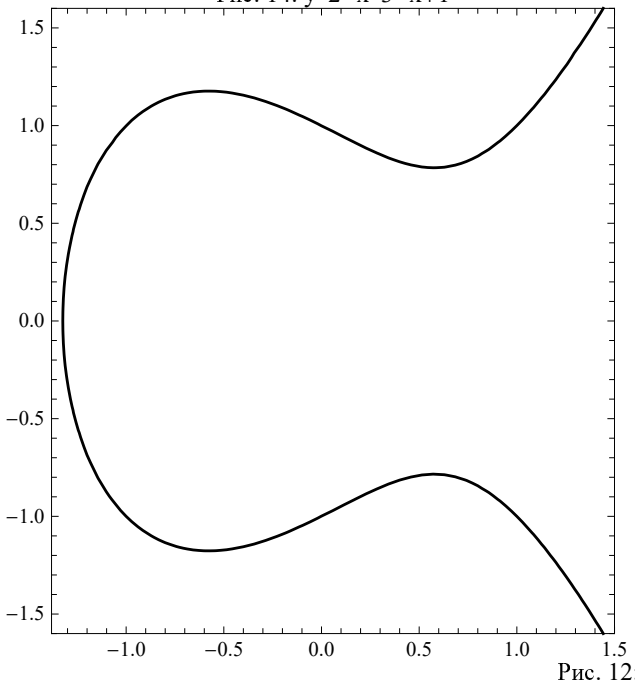



Рис. 13: y^2=x^3+x^2

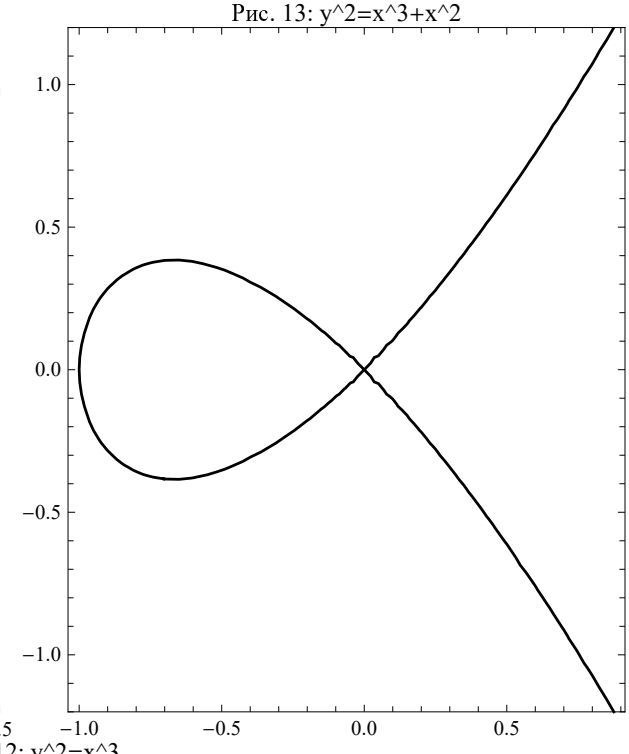
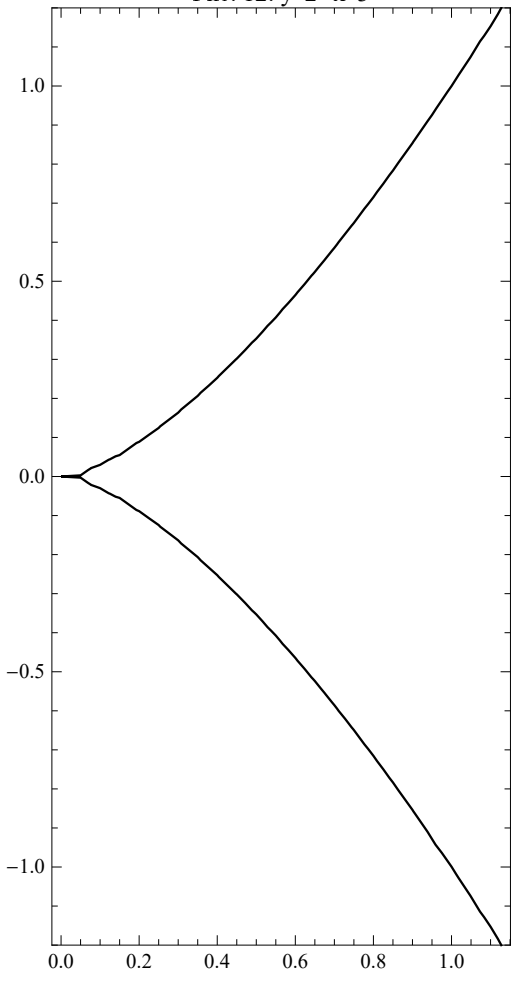



Рис. 12: y^2=x^3



• **Графическое представление неравенств.** Еще одной совершенно изумительной функцией системы, которая за секунды проделывает то, на выполнение чего обычному человеку нужно трудиться полдня, является команда `RegionPlot`. Формат вызова функции:

```
RegionPlot[f1[x,y]<g1[x,y]&&f2[x,y]<g2[x,y]&&...,
                                        {x,a,b},{y,c,d}]
```

В результате изображаются те точки прямоугольника

$$\{(x,y) \in \mathbb{R}^2 \mid a \le x \le b,\ c \le y \le d\},$$

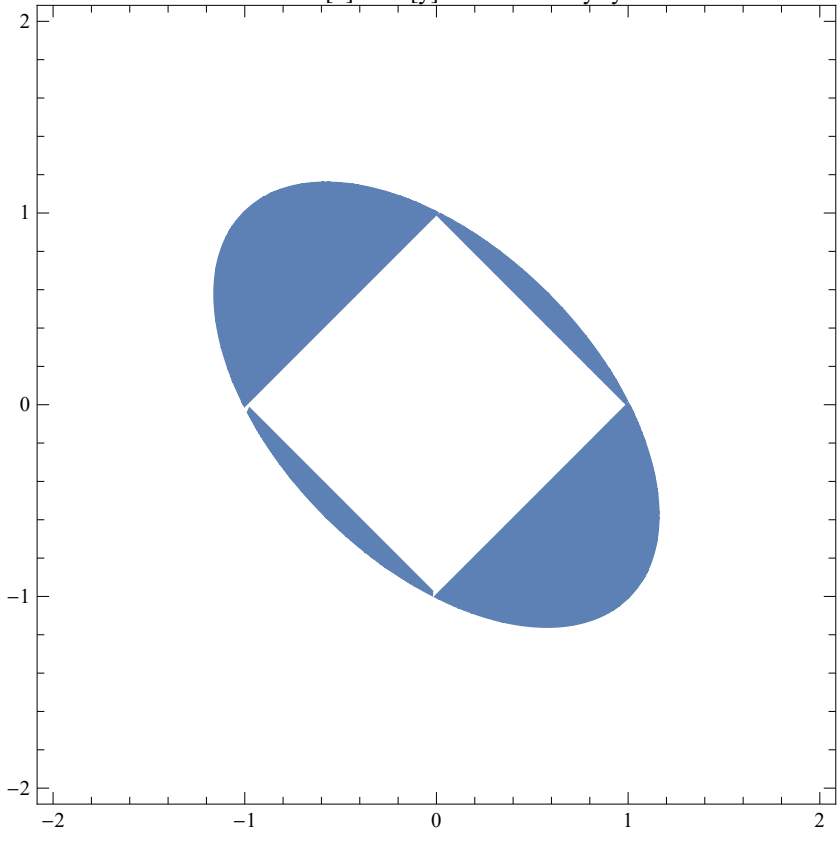
которые удовлетворяют неравенствам

$$f_1(x,y) < g_1(x,y), \qquad f_2(x,y) < g_2(x,y), \qquad \dots$$

Вот, например, как строится рисунок 15, изображающий решения системы неравенств, которая обсуждалась в конце § 4:

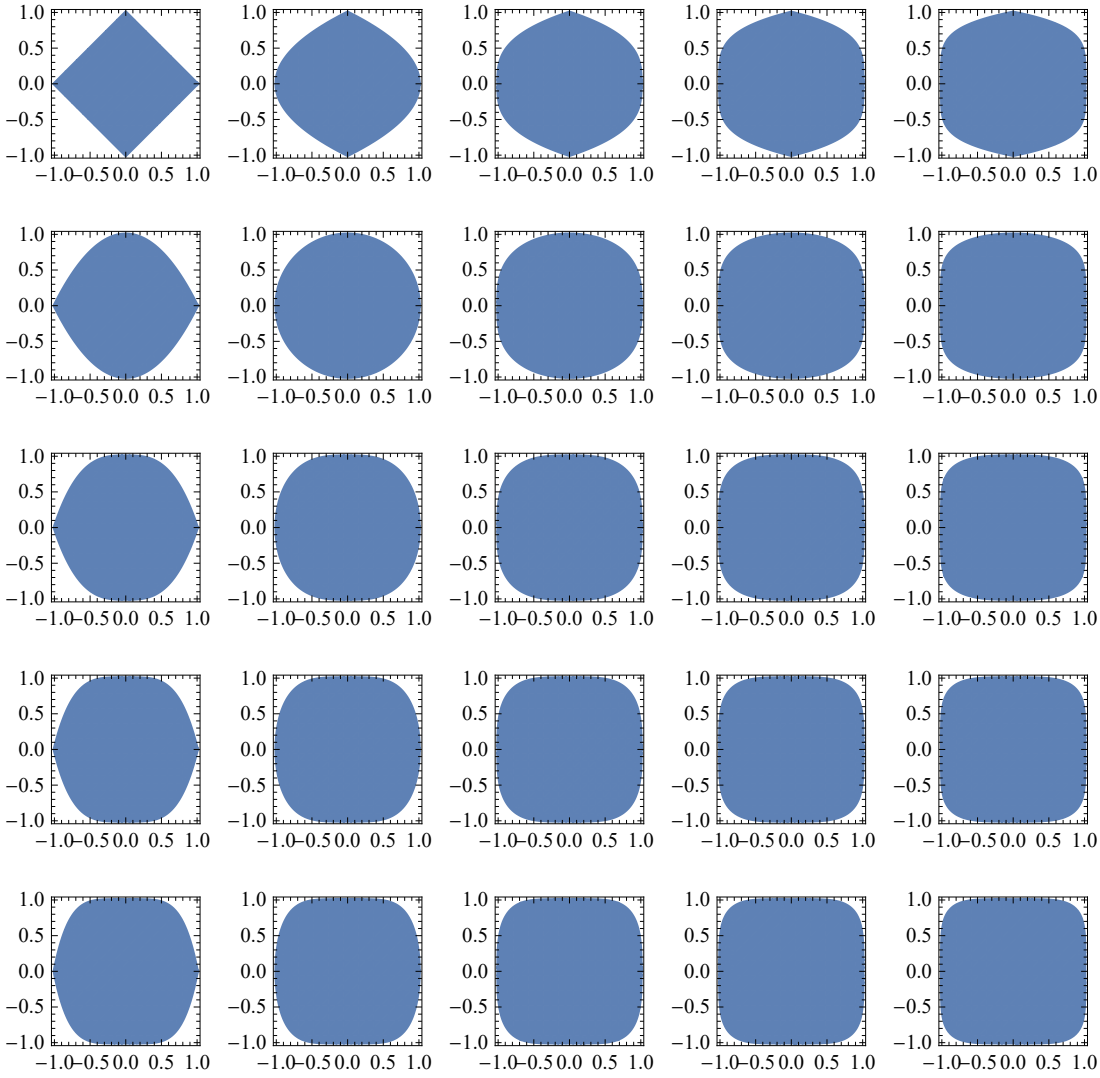
```
In[101]:=RegionPlot[Abs[x]+Abs[y]>1&&x^2+x*y+y^2<1,
                                        {x,-2,2},{y,-2,2},
    AspectRatio->Automatic,
    PlotStyle->AbsoluteThickness[0.5],
    AxesStyle->AbsoluteThickness[1],
    MaxRecursion -> 10,
    PlotLabel->"Рис.  15: Abs[x]+Abs[y]>1&&x^2+x*y+y^2<1"]
```

Рис. 15: Abs[x]+Abs[y]>1&&x^2+x∗y+y^2<1



Вот еще один совершенно замечательный пример, демонстрирующий возможности команды `RegionPlot`, – таблица, состоящая из графиков решения неравенств $|x|^p + |y|^q \le 1$.

Рис.16: Hoelder balls



Поясним, что при $p = q$ решения этого неравенства представляют собой шары по отношению к **метрике Гельдера** или, коротко, **шары Гельдера** = `Hölder balls`. Частными случаями метрики Гельдера являются

- **городская метрика** (при $p = 1$),
- обычная **эвклидова метрика** (при $p = 2$),
- **метрика Чебышева** (при $p = \infty$).

Хорошо видно, что в городской метрике шар является квадратом (как и следовало ожидать!), в эвклидовой метрике — кругом (как и следовало ожидать!), а потом по мере того как $p$ растет, его форма снова приближается к квадрату:

```
In[102]:=GraphicsGrid[Table[RegionPlot[Abs[x]^p+Abs[y]^q<=1,
                                        {x,-1,1},{y,-1,1},
        PlotStyle->AbsoluteThickness[1.5],
        AxesStyle->AbsoluteThickness[1],
        Ticks->None,
        DisplayFunction->Identity],
                    {p,1,5},{q,1,5}],
        PlotLabel->"Рис.16: Hoelder balls"]
```

Здесь используются две не встречавшиеся нам до сих пор команды.

◦ Вызванная в формате

`GraphicsGrid[{u,v,w,...}]`

команда `GraphicsGrid` собирает графические объекты `u,v,w,...` в один объект, состоящий из *одинаковых* прямоугольников, в которые вписаны объекты `u,v,w,...` Вызванная в формате

`GraphicsGrid[{{u,v,...},{w,z,...},...}]`

она делает то же самое, но при этом располагает прямоугольники, содержащие объекты `u,v,w,z,...`, в виде двумерной таблицы. В данном случае мы по отдельности строим 25 шаров, а потом соединяем их в один графический объект $5 \times 5$.

Мы могли бы, например, строить Рисунки 7–10 и 11–14 на одной странице при помощи команды `GraphicsGrid`. Но фактически мы поступали иначе, пользуясь командами двумерной графики, вручную задавая размеры прямоугольников, в которые вписаны отдельные части картинки.

◦ Опция `DisplayFunction->Identity` встроена в тело `RegionPlot` для того, чтобы подавить вывод промежуточных результатов на экран. По умолчанию

`DisplayFunction->$DisplayFunction`,

при этом выводилась бы не только окончательная таблица из 25 шаров, но и каждый из них по отдельности, по мере вычисления соответствующего `RegionPlot`. Обратите внимание, что значение этой опции распространяется на `RegionPlot`, но не на `GraphicsGrid`. Почему?

• **Простейшая двумерная графика.** Мы не будем пытаться сколь-нибудь систематически описывать здесь технику построения двумерных и, тем более, многомерных картинок – эта тема требует отдельного рассмотрения.[33] Ограничимся построением двух знаменитых картинок. При этом мы не стремимся здесь дать наиболее короткую или эффектную программу, а приводим такое описание, где смысл каждой команды должен быть понятен начинающему. В действительности, мы обычно не вводим никаких координат руками, а вычисляем их (как таблицы, решения уравнений или что-нибудь в таком духе), после чего применяем к списку координат

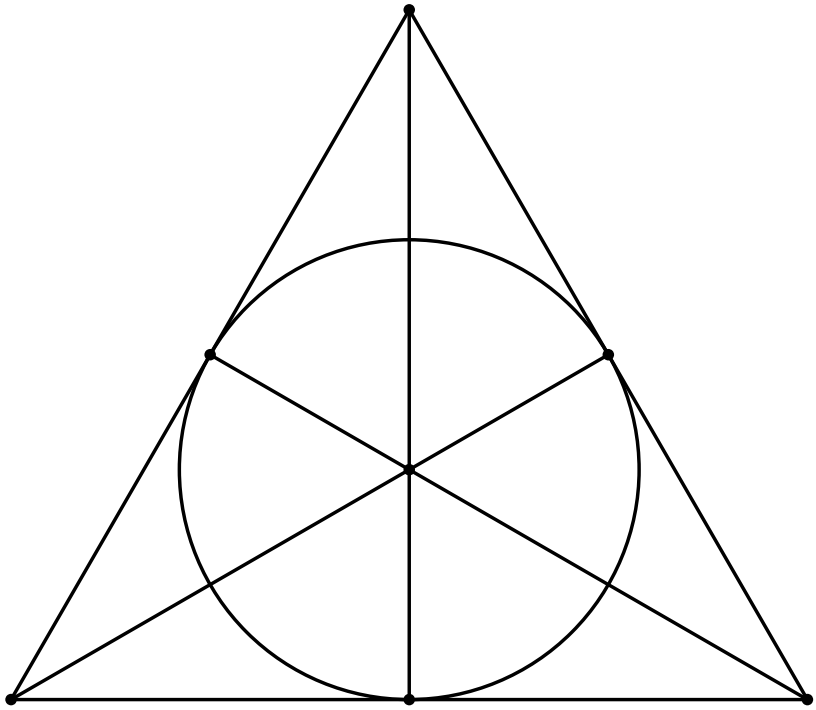
[33] В материалах `Documentation Center` визуазизации данных и построению графиков посвящен специальный раздел `Graph Drawing`

графические примитивы посредством конструкций типа `Map[Point,...]`, `Map[Line,...]` и т.д. Получающиуся текст не намного короче, но имеет более прозрачную с точки зрения опытного пользователя структуру и при этом резко уменьшается вероятность ошибки.

Первый из приведенных ниже текстов порождает изображение **плоскости Фано** — проективной плоскости над полем из двух элементов. Не углубляясь, поясним, что в терминах этой картинки описывается умножение семи мнимых единиц в **алгебре октав Кэли**:

```
In[103]:=rt=Sqrt[3]/2;
        nodes={{0,0},{1,0},{2,0},{1,2*rt},
                        {1,2*rt/3},{1/2,rt},{3/2,rt}};
        Show[Graphics[{
                {AbsolutePointSize[5],
                        Map[Point,nodes]},
                {AbsoluteThickness[1.5],
                        Line[{{0,0},{2,0},{1,2*rt},{0,0}}],
                        Line[{{0,0},{3/2,rt}}],
                        Line[{{2,0},{1/2,rt}}],
                        Line[{{1,2*rt},{1,0}}],
                        Circle[{1,2*rt/3},2*rt/3]}}]]
```

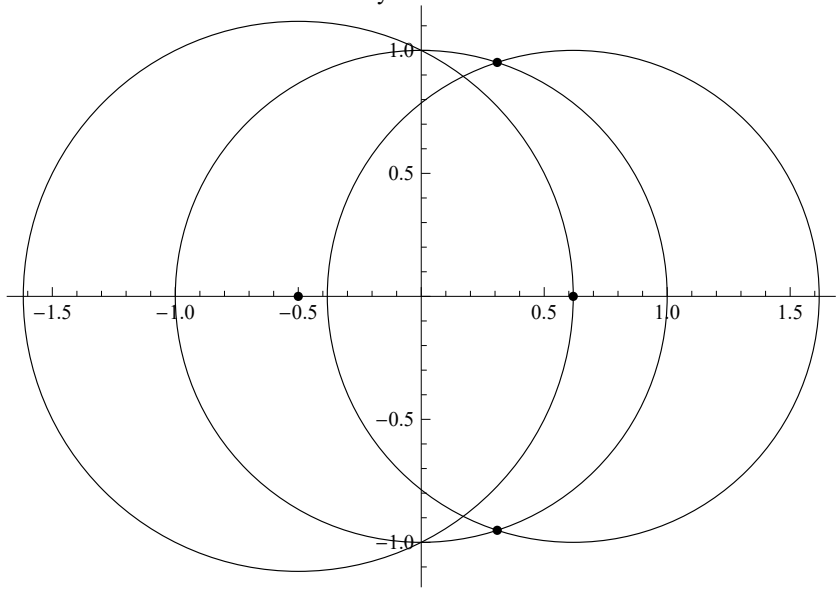
Рис.17: Плоскость Фано



Второй текст порождает рисунок из Альмагеста, изображающий построение правильного пятиугольника при помощи циркуля и линейки:

```
In[104]:=Show[Graphics[{
                {AbsolutePointSize[4],
                        Point[{-1/2, 0}],
                        Point[{-1/2+Sqrt[5]/2,0}],
                        Point[{Cos[2*Pi/5],Sin[2*Pi/5]}],
                        Point[{Cos[2*Pi/5],-Sin[2*Pi/5]}]},
                {AbsoluteThickness[0.5],
                        Circle[{0,0}, 1],
                        Circle[{-1/2,0},Sqrt[5]/2],
                        Circle[{-1/2+Sqrt[5]/2,0}, 1]}},
                Axes->True]]
```

Рис.18: Рисунок из Альмагеста




Мы не будем детально комментировать эти тексты, но большая часть того, что происходит, понятна сама по себе:

∘ Команда `Graphics` соединяет определенные в ее теле графические примитивы в составной двумерный объект формата `Graphics`, который после этого обрабатывается как единое целое.

∘ Команда `Show` отображает этот графический объект на экране. У начинающего в этом месте должен возникнуть вопрос, а что еще можно сделать с графическим объектом, кроме как отобразить его на экране? Ну, с ним можно *много чего* делать: вместо этого мы могли бы, например, сразу записать его в файл, включить его в определение другого графического объекта или мультипликации, преобразовать в другой формат, изменить настройки опций, применить к нему какое-то геометрическое преобразование и т.д.

Следующие три типа объектов называются **графическими примитивами**, при помощи них строятся различные элементы картинки, в данном случае точки, линии и окружности:

◦ Примитив **точка** `Point[{x,y}]` определяет точку $(x, y)$ с координатами $x$ и $y$.

◦ Примитив **линия** `Line[{{u,v},{x,y}}]` определяет отрезок с концами $(u, v)$ и $(x, y)$. Вызванный с более длинным списком координат этот примитив порождает ломаную, состоящую из отрезков, соединяющих последовательные точки. Например, `Line[{{u,v},{x,y},{z,w}}]` состоит из *двух* отрезков, а именно, отрезка с концами $(u, v)$ и $(x, y)$ и второго отрезка с концами $(x, y)$ и $(z, w)$. В случае, когда последняя точка списка совпадает с первой, мы получим замкнутую ломаную. Например,

```
Line[{{u,v},{x,y},{z,w},{u,v}}]
```

изображает треугольник с вершинами $(u, v)$, $(x, y)$ и $(z, w)$.

◦ Примитив **окружность** `Circle[{x,y},r]` определяют окружности радиуса $r$ с центром в $(x, y)$.

Кроме того, в системе имеется много других графических примитивов, например, `Rectangle`, `Disk` и `Polygon` для закрашенных фигур, `Raster`, `PostScript`, `Text` и т.д.

**Графические директивы**, примененные к соответствующим примитивам, задают значения параметров, которые применяются при их построении:

◦ Директива **абсолютный размер точки** `AbsolutePointSize[d]` задает (абсолютный) размер точек в типографских пунктах.

◦ Уже встречавшаяся нам при обсждении графиков директива **абсолютная толщина** `AbsoluteThickness[d]` задает (абсолютную) толщину линий в типографских пунктах.

Обратите внимание на синтаксис!! Графические директивы (в отличие от опций, применяемых к объекту в целом) образуют список вместе с теми графическими примитивами, к которым они относятся. Это сделано для того, чтобы на одном и том же рисунке можно было рисовать точки разных размеров и линии разной толщины. Имеется большое количество других графических директив: `PointSize`, задающая (относительный) размер точек, `Dashing` и `AbsoluteDashing` для создания пунктирных линий, `GrayLevel`, `RGBColor`, `Hue` и т.д.

◦ Опция `Axes->True` предлагает включить в графический объект координатные оси.

• **Построение графиков функций двух переменных.** График функции двух переменных строится при помощи команды `Plot3D`, которая порождает объект формата `SurfaceGraphics`. В простейшем варианте эта команда вызывается в формате:

```
Plot3D[f[x,y],{x,a,b},{y,c,d}]
```

Эта команда порождает трехмерный график функции $f(x, y)$ при $x$ при меняющемся от $a$ до $b$, и $y$ меняющемся от $c$ до $d$.

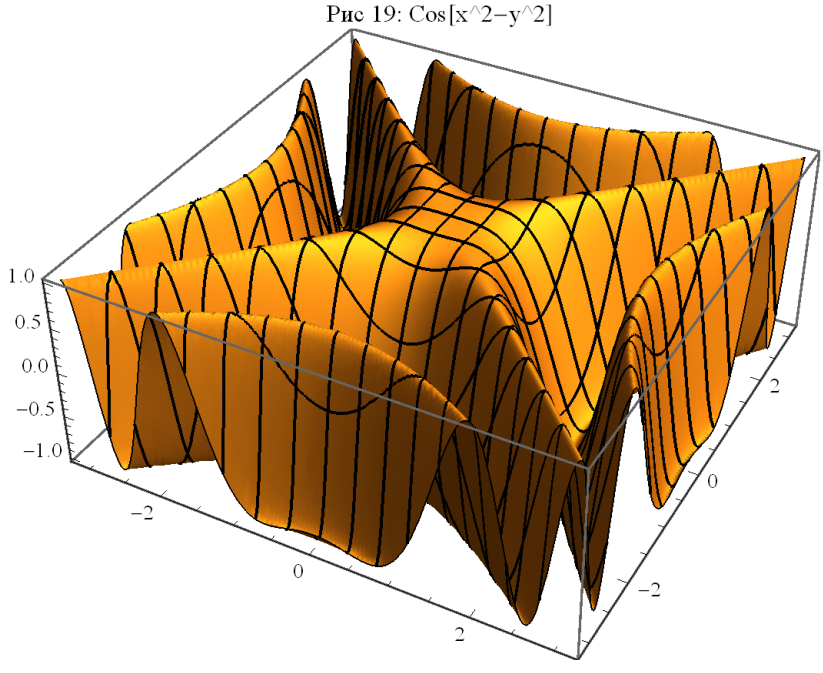
В одном отношении использование команды `Plot3D` радикально отличается от использования команды `Plot`. А именно, `Plot` трактует список функций как предложение построить графики нескольких функций на одной картинке. При этом `Plot` может построить графики сколь угодно большого количества функций, это лимитируется только памятью Вашего компьютера и разрешением устройств вывода. В то же время первый аргумент команды `Plot3D` может быть либо *одной* функцией, либо списком из *двух* функций. Однако вызванная в формате

```
Plot3D[{f[x,y],g[x,y]},{x,a,b},{y,c,d}]
```

команда `Plot3D` будет воспринята отнюдь не как пожелание построить графики функций $f$ и $g$ на одной картинке. Нет, это пожелание построить график функции $f$, для которого **затенение** задается функцией $g$.

Приведем совсем простой пример использования команды `Plot3D`. Воспроизведенный на Рисунке 19 график функции $\cos(x^2 - y^2)$ построен при помощи следующей команды:

```
In[105]:=Plot3D[Cos[x^2-y^2],{x,-Pi,Pi},{y,-Pi,Pi},
                MeshStyle->AbsoluteThickness[1],
                BoxStyle->AbsoluteThickness[0.8],
                PlotPoints->60,
                PlotLabel->"Рис.  19: Cos[x^2-y^2]"]
```




Чтобы понять, что здесь происходит, нужно знать, что графический объект формата `SurfaceGraphics` состоит из большого количества данных, самыми важными из которых являются:

- собственно **поверхность** с координатами $(x, y, f(x, y))$,
- наброшенная на нее **сетка** или **мешок** `Mesh`,

◦ **подсветка** `Lighting`,

◦ **ящик** `Box`.

Кроме того, в полное описание объекта этого типа входят еще **точка зрения** `ViewPoint`, **оси** `Axes`, **решетка** `Grid`, **текст** `Text` и т.д.

◦ Опция `Shading->False` убирает нюанс и введена *исключительно* для получения черно-белой картинки.

В обычных ситуациях мы **никогда** не используем эту опцию — для получения изображения высокого качества на экране компьютера или цветном принтере поступают прямо противоположным образом. А именно, в этом случае мы обычно убираем не нюанс, а сетку `Mesh->False`. При этом получается изображение, приближающееся по своим художественным достоинствам к лучшим холстам и доскам Дени, Дали и Дельво.

Использование следующих двух опций полностью аналогично использованию уже встречавшизся нам опций `PlotStyle` и `AxesStyle`. Они задают графические директивы, которые следует применять к соответствующим частям трехмерного графика:

◦ Опция `MeshStyle` описывает **стиль сеточных линий**, в данном случае абсолютную толщину линий.

◦ Опция `BoxStyle` описывает **стиль охватывающего "ящика"**, в данном случае снова мы меняем только абсолютную толщину линий.

◦ Опция `PlotPoints` показывает, в каком минимальном количестве точек следует вычислять значения функции. Вызванная в формате

```
PlotPoints->n
```

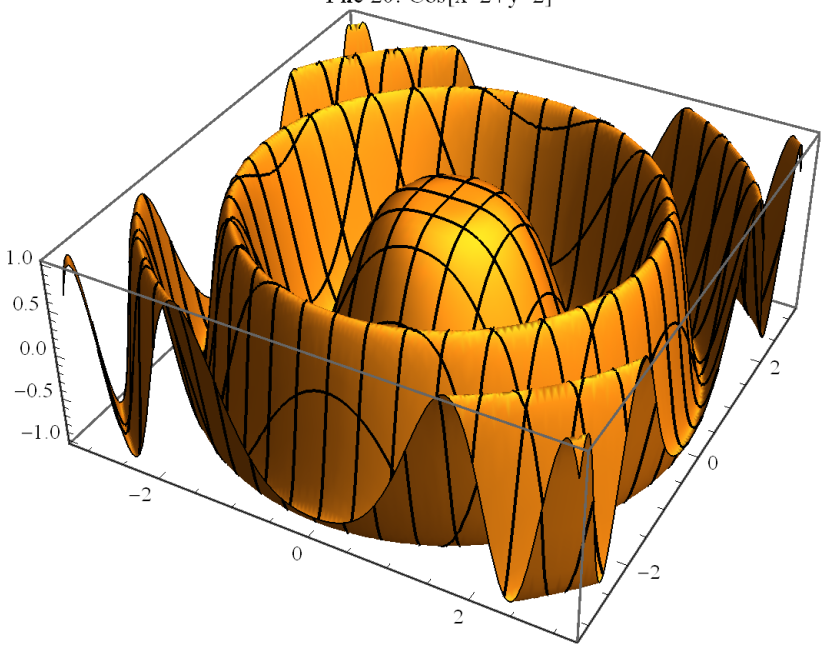
эта опция предписывает команде `Plot` вычислять значение функции *как минимум* в $n$ точках — после чего продолжать вычислять значения в промежуточных точках для быстро осциллирующих функций. По умолчанию `PlotPoints->25`. Для команды `Plot3D` опция `PlotPoints` в формате `PlotPoints->n` предписывает вычислять значения в $n$ точках по каждой из координат. Чтобы вычислять значения в $m$ точках по $x$ и в $n$ точках по $y$, эту опцию следует задавать в формате

```
PlotPoints->{m,n}.
```

Для получения графики высокого качества мы часто задаем `PlotPoints->500` и даже `PlotPoints->1000`, что изначально требует вычисления *миллиона* значений функции и может занять некоторое. Понятно, что это имеет смысл только, если Вы собираетесь выводить эту картинку на такое устройство, которое допускает соответствующее разрешение, и только если одновременно убрать сетку, положив `Mesh->False`.

**Упражнение.** Постройте воспроизведенный на рисунке 20 график функции $\cos(x^2 + y^2)$.

Рис 20: Cos[x^2+y^2]




Стоит подчеркнуть, что в объекте формата `SurfaceGraphics` хранится *гораздо* больше информации, чем в видимом изображении. Например, этот объект сохраняет и те части поверхности, которые при просмотре с данной точки зрения закрыты от зрителя другими частями. Однако эти части можно увидеть, если повернуть поверхность или изменить точку зрения. Более того, опция `HiddenSurface->False` делает поверхность прозрачной и, таким образом, все части поверхности, обычно закрытые другими ее частями, становятся видимыми.

В завершение параграфа хотелось бы высказать надежду, что нам удалось показать некоторые ключевые приемы работы с графикой в системе `Mathematica`, которые позволят читателю строить подобные изображенным картинки – и получать от этого удовльствие.

## § 7. Суммы, произведения, пределы

> Рамануджан говорил, что богиня Намаккал внушала ему формулы во сне. Часто встав с кровати он (не) мог записать результаты и быстро проверить их ... Рамануджан имел устойчивые религиозные взгляды. Он особенно почитал богиню Намаккал.
>
> Сеши Нараяна Айар и Рамачандра Рао, “Жизнь Рамануджана”

> Я меньше всего хочу, чтобы Вы вскидывали руки вверх и восклицали: “Здесь есть что-то необъяснимое, таинственное проявление древней восточной мудрости.” Не верю я в эту древнюю восточную мудрость.
>
> Гарольд Годфри Харди[34]

Системы компьютерной алгебры полностью обесценивают **все** традиционные *навыки*, которым обучают в курсах математического анализа, высшей

[34] Г.Г.Харди, Двенадцать лекций о Рамануджане. — М., ИКИ, 2002, с.1–335; стр.13.

математики, теории дифференциальных уравнений. Mathematica умеет вычислять пределы, ряды, бесконечные произведения, дифференцировать, интегрировать, раскладывать в степенные и тригонометрические ряды, решать дифференциальные уравнения и уравнения в частных производных и т.д. Более того, мы готовы утверждать, что она делает все эти вещи лучше, чем **любой** неспециалист, и все еще лучше, чем подавляющее большинство профессиональных математиков.

• **Суммы.** Одной из простейших и самых полезных команд системы является команда Sum. А именно, Sum[f[i],{i,m,n}] выражает сумму $\sum_{i=m}^{n} f(i)$ значений функции $f$ по $i$ от $m$ до $n$ (с шагом 1). Обратите внимание на форму итератора {i,m,n}, которая используется в большинстве итеративных команд системы. Эта форма взята из языка C, остальные универсальные системы компьютерной алгебры, такие как Axiom, Maple, MuPAD используют i=m..n взятую из Pascal. По умолчанию шаг суммирования равен 1, однако задание итератора в форме {i,m,n,d} говорит, что $i$ должно меняться от $m$ до $n$ с шагом $d$.

Многие алгебраические выражения, многочлены, дроби и тому подобное часто удобно выражать в виде сумм. Вот, скажем, как проще всего ввести многочлен Тэйлора экспоненты:

```
In[106]:= Sum[x^n/n!,{n,0,6}]
Out[106]=1+x+x^2/2+x^3/6+x^4/24+x^5/120+x^6/720
```

В § 1 мы уже обсуждали применение команды Sum для суммирования конкретных чисел. Однако это умеют делать и калькуляторы вроде Mathcad. Гораздо интереснее, что в Mathematica при помощи команды Sum можно производить **все** символьные манипуляции, которые могут встретиться начинающему, причем не только с конечными суммами, но и с бесконечными рядами!

• **Символьное вычисление сумм.** Совершенно замечательно, что суммы могут вычисляться не только в численном, но и в символьном виде, причем один или оба предела суммирования могут быть бесконечными! Что еще интереснее, пределы суммирования сами могут быть символами!!! Именно этим и отличаются интеллигентные системы Maple и Mathematica от бездушных калькуляторов наподобие Matlab.

Следующий совершенно удивительный пример показывает, что Mathematica *хорошо* усвоила[35], что такое многочлены Бернулли:

```
In[107]:= Do[Print[Sum[i^m,{i,1,n}]],{m,1,8}]
Out[107]=1/2*n*(1+n)
         1/6*n*(1+n)*(1+2*n)
```

[35]По причинам исторического и психологического плана мы предпочитаем избегать напрашивающегося выражения понимает, которое было бы использовано в случае успешного применения математической индукции в аналогичном контексте *человеческим* существом.

```
1/4*n^2*(1+n)^2
1/30*n*(1+n)*(1+2*n)*(-1+3*n+3*n^2)
1/12*n^2*(1+n)^2(-1+2*n+2*n^2)
1/42*n*(1+n)*(1+2*n)*(1-3*n+6*n^3+3*n^4)
1/24*n^2*(1+n)^2(2-4*n-n^2+6*n^3+3*n^4)
1/90*n*(1+n)*(1+2*n)*(-3+9*n-n^2-15*n^3+5*n^4+
        15*n^5+5*n^6)
```

`A parte:` еще бы ей этого не знать, если многочлены Бернулли и Эйлера являются основным инструментом суммирования полиномиальных рядов вообще.

Итеративная конструкция с `Do` и `Print` нам уже встречалась при обсуждении круговых многочленов. Первая из представленных здесь сумм, традиционно называется *суммой арифметической прогрессии* и изучается в школе, вторая и третья часто обсуждаются в математических кружках, но вот дальнейшие суммы, открытые в XVII веке Йоганном Фаульхабером и Яковом Бернулли[36], известны главным образом только профессиональным математикам. Поскольку невозможно представить, что все подобные суммы содержатся в системе в табличной форме (ниже мы приводим еще несколько примеров), это значит, что система в какой-то форме владеет идеей математической индукции.

В действительности, следующий пример должен убедить каждого минимально знакомого с предметом в том, что `Mathematica` все же не знает на память большинство сумм, а по-настоящему их вычисляет, причем делает это совершенно иначе, чем это свойственно человеку. Вот сумма, которую каждый будущий профессиональный математик открыл к возрасту пяти лет:

In[108]:= `Sum[i,{i,1,2*n-1,2}]`

Out[108]=`Floor[n]^2`

Еще раз напомним, что вызов итератора в форме `{i,m,n,d}` означает суммирование по $i$ от $m$ до $n$ с шагом $d$. Таким образом, мы предложили системе просуммировать все *нечетные* числа от 1 до $2n-1$. Видно, что здесь представлен не заученный ответ, а плод фактического (притом достаточно сурового) размышления. Поскольку мы изначально подразумевали (хотя пока и не сообщили системе), что число $n$ целое, мы видим, что ценой небанального напряжения `Mathematica` сумела открыть формулу $1+3+\ldots+(2n-1)=n^2$. В действительности в языке системы есть средства, которые позволяют декларировать, что $n$ является целым числом (или, с точки зрения внутреннего языка `Mathematica`, имеет объектный тип `Integer`):

- спецификация доменов,
- спецификация паттернов.

[36] В.В.Прасолов, Многочлены. — М., МЦНМО, 2000, с.1–335; стр.135–137.

Эти средства, относящихся к более высоким сферам программирования, мы здесь лишь упоминаем, но рассматривать не будем. Разумеется, после провозглашения $n$ целым, система сразу вернула бы ответ в форме $n^2$. Однако в § 5 нам уже встречалась команды `Refine` и `Assuming`, которые позволяет упростить ответ при определенных предположениях. Итак, сделаем еще одну попытку:

```
In[109]:= Refine[Simplify[Sum[i,{i,1,2*n-1,2}],
                                    Element[n,Integers]]]
Out[109]=n^2
```

Понятно, что здесь произошло? Мы наконец-то внятно объяснили системе, что $n$ целое. А именно, предположение `Element[n,Integers]` как раз и состоит в том, что $n$ является элементом домена целых чисел `Integers`. разумеется, применив вначале `Refine` с условием `Element[n,Integers]`, а потом `Simplify`, мы тоже получили бы результат $n^2$.

Теперь, когда мы знаем, что `Mathematica` не вытаскивает ответы из каких-то загадочных таблиц, а порождает их силой чистого разума, предложим ей что-нибудь чуть более хитрое:

```
In[110]:= Sum[2^i/i!,{i,1,n}]
Out[110]=-1+E^2*(1+n)*Gamma[1+n,2]/Gamma[2+n]
```

Появляющаяся в числителе функция $\Gamma(x,y)$ — это **неполная гамма-функция**:

$$\Gamma(x,y)=\int_y^\infty t^{x-1}e^{-t}\,dt,$$

в знаменателе стоит обычная гамма функция Эйлера $\Gamma(x)=\Gamma(x,0)$. Вы действительно думаете, что и этот ответ можно извлечь из таблицы? И где то место, где может храниться *такая* Таблица?

• **Кратные суммы.** Кратная сумма вида $\sum\limits_{i=m}^{n}\sum\limits_{j=k}^{l} f(i,j)$ выражается при помощи `Sum[f[i,j],{i,m,n},{j,k,l}` Здесь мы первый раз встречаемся с явлением, понимание которого чрезвычайно существенно для правильного использования итерационных команд вообще и команд формирования списков и работы с списками в особенности! Аналогичное соглашение действует для кратного интегрирования, построения графиков функций нескольких переменных и т.д. А именно, обратите внимание, что внутренние итераторы пишутся последними! Иными словами, `Sum[f[i,j],{j,k,l},{i,m,n}]` значит совершенно не то же самое, что `Sum[f[i,j],{i,m,n},{j,k,l}]`. В самом деле, при этой новой записи вначале происходит суммирование по $i$, а уже потом суммирование по $j$, в то время как в нашей исходной сумме вначале происходило суммирование по $j$ и только потом суммирование по $i$. Разумеется, для *конечных* сумм, в том случае, когда пределы суммирования внутренних сумм сами не зависят от $i$, порядок итераторов можно изменить, значение суммы при этом не меняется — именно этот прием и

называется **изменением порядка суммирования**. Однако для бесконечных сумм, а также в том случае, когда $k$ и $l$ сами являются функциями от $i$, ничего подобного делать нельзя.

Проиллюстрируем это явление на простом но важном примере. В компьютерной математике и анализе алгоритмов очень часто встречаются частичные суммы гармонического ряда

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n} = \sum_{i=1}^{n} \frac{1}{i},$$

которые называются **гармоническими числами**. В `Mathematica` есть специальное обозначение для $H_n$, а именно `HarmonicNumber[n]`, однако в дальнейшем мы иногда будем делать вид, что этого не знаем. Гармонический ряд расходится — иными словами, $H_n$ стремятся к бесконечности, хотя и очень медленно, порядка $\ln(n)$. Вот несколько первых гармонических чисел:

```
In[111]:= Table[HarmonicNumber[n],{n,1,12}]
Out[111]={1,3/2,11/6,25/12,137/60,49/20,363/140,761/280,
                  7129/2520,7381/2520,83711/27720,86021/27720}
```

Так как `Mathematica` знает определение гармонического числа, то вычисление `Sum[1/i,{i,1,n}]` дает `HarmonicNumber[n]`. В прежние годы в качестве упражнения на математическую индукцию мы часто предлагали студентам вычислить сумму первых гармонических чисел, т.е. сумму

$$H_1 + H_2 + \ldots + H_m.$$

Разумеется, это один из тех случаев, когда формулировка задачи содержит подсказку, а именно, предложение *вычислить* сумму недвусмысленно указывает на то, что это *возможно*. А теперь сравните следующие два выражения:

```
Sum[1/i,{n,1,m},{i,1,n}]
Sum[1/i,{i,1,n},{n,1,m}]
```

Понятно, что эти выражения вычисляют? В первой сумме осуществляется суммирование вначале по $i$ от 1 до $n$, а потом по $n$ от 1 до $m$ и это как раз то, что мы хотели. С другой стороны, во второй сумме сначала осуществляется суммирование по $n$ от 1 до $m$, а потом все эти $m$ *одинаковых* сумм (ведь $1/i$ не зависит от $n$) суммируются по $i$ от 1 до $n$.

Таким образом, если Вы не уверены, что отчетливо понимаете, что здесь происходит, то вот Вам добрый совет: оформляйте кратные суммы обычным в математике образом, с повторением команды `Sum`. Например, искомую сумму гармонических чисел можно еще выразить посредством

```
In[112]:= Sum[Sum[1/i,{i,1,n}],{n,1,m}]
Out[112]=-1 - m + HarmonicNumber[1 + m] + m*HarmonicNumber[1 + m]
```

Это чуть длиннее и менее элегантно с программистской точки зрения, зато не оставляет никакого места сомнению. Ясно, что здесь суммирование происходит вначале по $i$ от 1 до $n$ — это дает нам $n$-е гармоническое число $H_n$ — а потом полученные результаты суммируются по $n$ от 1 до $m$.

Уже весьма простые на вид кратные суммы при символьном вычислении часто приводят к достаточно замысловатым ответам. Попытка вычислить первую двойную сумму, которая приходит в голову,

```
In[113]:= Sum[1/(i+j),{i,1,m},{j,1,n}]
Out[113]=-PolyGamma[0,1+n]+PolyGamma[0,1+m+n]
```

сразу приводит нас к функции **дигамма** $\psi(x) = \Gamma'(x)/\Gamma(x)$ (логарифмическая производная гамма-функции Эйлера). Ясно, что получение подобного ответа вручную в реальном времени лежит за пределами возможностей большинства профессиональных математиков, кроме избранных специалистов по комбинаторике, теории чисел, классическому анализу и математической физике.

• **Бесконечные суммы.** `Mathematica` умеет выражать многие бесконечные суммы как значения элементарных или специальных функций:

```
In[114]:= Sum[1/(n!x^n),{n,0,Infinity}]
Out[114]=E^(1/x)
In[115]:= Sum[1/n^3,{n,1,Infinity}]
Out[115]=Zeta[3]
In[116]:= Sum[1/(n!n^3),{n,1,Infinity}]
Out[116]=HypergeometricPFQ[{1,1,1,1},{2,2,2,2},1]
```

В первом случае мы видим экспоненту, во втором — значение $\zeta(3)$ дзета-функции Римана, а фигурирующая в третьем ответе `HypergeometricPFQ` представляет собой обобщенную гипергеометрическую функцию, которая возникает в самых разных вопросах математики, от комбинаторики и теории чисел до математической физики.

• **Произведения.** Формат команды `Product` полностью аналогично формату `Sum`. А именно,

```
Product[f[i],{i,m,n}]
```

выражает произведение $\prod_{i=m}^{n} f(i)$ значений функции $f$ по $i$ от $m$ до $n$ (с шагом 1). Как видим, здесь применяется обычная форма итератора `{i,m,n}`.

Как и в случае сумм, произведения могут использоваться для численных вычислений, такой пример мы уже приводили, и для короткой записи алгебраических выражений. Вот, например, как проще всего ввести **гауссов многочлен**, выражающий количество базисов в пространстве размерности $n$ над полем из $q$ элементов:

```
In[117]:= Product[q^n-q^i,{i,0,n-1}] /.  n->10
Out[117]=(-1+q^10)(-q+q^10)(-q^2+q^10)(-q^3+q^10)(-q^4+q^10)
         (-q^5+q^10)(-q^6+q^10)(-q^7+q^10)(-q^8+q^10)(-q^9+q^10)
```

Используемая в этом тексте замена `/. n->10` состоит из оператора `/.` или, в полной форме, `ReplaceAll` и правила подстановки `n->10`. Примененные вместе они побуждают систему заменить $n$ на 10 всюду, на протяжении одного вычисления. Преимущество такой конструкции перед присваиванием `n=10` состоит в том, что $n$ не приобретает постоянного значения, так что предшествующий текст может быть без всяких изменений включен в другое вычисление, в котором $n$ имеет любое другое значение — или само является своим собственным значением, т.е. рассматривается как независимая переменная. Подробнее все нюансы, связанные с переменными и их значениями, а также использованием присваиваний и подстановок, обсуждаются в Модуле 2.

Разумеется, `Mathematica` не будет автоматически раскрывать скобки в символьном выражении, получающемся в результате образования произведения, чтобы принудительно раскрыть скобки в *конечном* произведении, нужно применить `Expand` или `Distribute`:

```
In[118]:= Product[1-x^i,{i,1,10}]

Out[118]=(1-x)(1-x^2)(1-x^3)(1-x^4)(1-x^5)
         (1-x^6)(1-x^7)(1-x^8)(1-x^9)(1-x^10)

In[119]:= Distribute[Product[1-x^i,{i,1,10}]]

Out[119]=1-x-x^2+x^5+x^7+x^11-x^12-x^13-x^14-2*x^15+x^18+x^19+
         x^20+x^21+3*x^22-x^25-x^26-2*x^27-2*x^28-x^29-x^30+
         3*x^33+ x^34+x^35+x^36+x^37-2*x^40-x^41-x^42-x^43+
         x^44+x^48+x^50-x^53-x^54+x^55
```

Как и в случае сумм, один или оба предела в задании итератора могут быть бесконечными. Скажем, вычисление эйлеровского произведения

```
In[120]:= Product[(1-1/Prime[n]^2)^(-1),{n,1,Infinity}]
```

где `Prime[n]` обозначает $n$-е простое число, дает явное значение $\pi^2/6$.

• **Проверка на порогах.** Вот одно из наших излюбленных произведений, как из уважения к классикам XVIII века, так и конкретно с точки зрения сравнения интеллектуальных возможностей систем компьютерной алгебры и примазывающихся к ним:

$$\prod_{i=1}^{2n} i^{(-1)^{i-1}} = \frac{1\cdot 3\cdot 5\cdot \ldots \cdot (2n-1)}{2\cdot 4\cdot 6\cdot \ldots \cdot 2n}$$

Предложите Вашей любимой системе *вычислить* это произведение. Достоинство этого теста состоит в простоте формулировки и убийственной убедительности ответа. Тот, кто понимает предмет, но не знаком с его историей, должен в этот момент испытывать чувство культурного шока: ожидаем ли мы, что система может *заметить*, что в знаменателе стоит порядок октаэдральной группы $2^n n!$ — $2^n n!$ IS THE ANSWER, WHAT IS THE QUESTION? — а в числителе $(2n-1)!/2^{n-1}(n-1)!$? Иными словами, надеемся ли мы

увидеть что-то в духе

$$\prod_{i=1}^{2n} i^{(-1)^{i-1}} = \frac{(2n-1)!}{2^{2n-1}n!(n-1)!}?$$

Да нет же, мы уже говорили, что проводимые системами компьютерной алгебры вычисления основаны на совершенно других принципах, а реальность превосходит все наши представления о ней. Если система компьютерной алгебры и сможет найти формулу для этого произведения, то это, конечно, будет совсем другая формула! Либо та же формула, но найденная *как-то иначе* и записанная в совершенно другом виде. Абсолютно принципиальный момент, который сразу очень многое объясняет в истории математики, состоит в том, что `Mathematica` скорее найдет совершенно невероятную формулу в духе Эйлера и Рамануджана, чем ту простую формулу, которую написал бы современный математик. Ясно, что никакой мистики здесь нет, она действует тем способом, который *с ее точки зрения* является наиболее экономным, разумным и простым! Это всего лишь значит, что ее представление об экономии мысли отличается от нашего!!!

В любом случае, ценой небанального напряжения мысли (больше секунды раздумий!) `Mathematica` находит общую формулу. Для полного реализма возьмем произведение до $n$, а не до $2n$, чтобы не оставалось уже никаких сомнений, что начиная вычисление система не может знать ответа:

```
In[121]:= Product[i^((-1)^(i-1)),{i,1,n}]

Out[121]=E^((-1/2)*(-Log[2]+(-1)^n*Log[2]+Log[Pi])+
          (-1)^n*(Log[Gamma[(1+n)/2]]-Log[Gamma[(2+n)/2]]))
```

Ну как, все еще из Таблицы? Да, но почему тогда у `Maple` нет доступа к этой Таблице? Взяв теперь произведение до $2n$, мы придем к формуле, которая хотя и эквивалентна написанной выше, но выражена гораздо более экстравагантно:

```
In[122]:= Product[i^((-1)^(i-1)),{i,1,2*n}]

Out[122]=Gamma[1/2+n]/(Sqrt[Pi]*Gamma[1+n])
```

Именно в такой форме, конечно, ее и использовал Рамануджан. Однако в случае компьютера мы *почему-то* твердо уверены, что эта новая формула не внушена богиней Намаккал, а получена в результате честного (и довольно трудного) вычисления — не верю я в эту древнюю восточную мудрость! Если нам трудно представить, что подобное вычисление может быть проведено человеком, то это только потому, что наше собственное математическое воспитание проходило под девизом простоты и неизбежности.

И теперь у нас, конечно, не может быть сомнений, что `Mathematica` сумеет перейти к пределу в предыдущей формуле:

```
In[123]:= Product[i^((-1)^i),{i,1,Infinity}]
```

Out[123]=`Sqrt[Pi/2]`

В действительности, она, конечно, не использует здесь предыдущую формулу, так как вычислить конечное произведение *намного* сложнее.

• **Вычисление пределов.** Основной командой для вычисления пределов в `Mathematica` является `Limit`. А именно, для вычисления $\lim\limits_{x\to c} f(x)$ эта команда вызывается в следующем экзотическом формате:

`Limit[f[x],x->c]`.

Обратите внимание, что в угоду традиционному математическому обозначению значение параметра $c$ задается так, как будто это опция! В действительности, конечно, было бы правильно поступать ровно наоборот, а именно, аналистам следовало бы предел функции $f$ точке $c$ записывать не как $\lim\limits_{x\to c} f(x)$, а просто как $\lim_c(f)$, ведь на самом деле здесь никто никуда не переходит, никто ни к кому не стремится, и никакой буковки $x$ в выражении `предел функции` $f$ `точке` $c$ нет и отродясь не было. Кроме того, запись $f \mapsto \lim_c(f)$ подчеркивала бы аналогию с другими гомоморфизмами кольца функций, в которых $c$ выступает в качестве параметра, скажем, с гомоморфизмом эвалюации $f \mapsto \mathrm{ev}_c(f) = f(c)$. Но чего только не сделаешь ради поддержания самой абсурдной традиции!

Однако подобные глупости не мешают системе правильно вычислять пределы, причем в символьном виде:

In[124]:= `Limit[(Cos[m*x]-Cos[n*x])/(x^2),x->0]`

Out[124]=`1/2*(-m^2+n^2)`

Команда `Limit` ищет предел и в тех случаях, когда он не является единственным:

In[125]:= `Limit[Sin[1/x],x->0]`

Out[125]=`Interval[{-1,1}]`

Разумеется, к точно такому же результату приведет и вычисление

In[126]:= `Limit[Sin[x],x->Infinity]`

Для вычисления левого предела $\lim\limits_{x\to c-} f(x)$ и правого предела $\lim\limits_{x\to c+} f(x)$ команда `Limit` вызывается в следующих форматах:

`Limit[f[x],x->c,Direction->1]`
`Limit[f[x],x->c,Direction->-1]`.

Однако нам встречались случаи, когда система не в состоянии найти предел в символьном виде. В этом случае можно использовать команду `NLimit`, содержащуюся в пакете `NumericalCalculus`.

## § 8. Производные

Don't drink and derive.
Motto of the society: Mathematicians Against Drunk Deriving

В `Mathematica` имеются две команды символьного дифференцирования `D` и `Derivative`, при помощи которых можно явно вычислить все производные, которые могут встретиться любому нематематику.

• **Дифференцирование.** Дифференцирование представляет собой чисто алгоритмическую операцию и производная любой элементарной функции может быть вычислена при помощи чисто механических манипуляций. В системе `Mathematica` имеются две основные команды для вычисления производной, а именно,

◦ функция вычисления производной `D`, которая находит *значение* производной;

◦ оператор дифференцирования `Derivative[1]`, в операторной форме `'`, который вычисляет производную *как функцию*.

В простейшем виде вычисление производной имеет формат `D[f[x],x]`, где `f[x]` — функция, которую мы хотим продифференцировать, а `x` — переменная, по которой производится дифференцирование. Чтобы убедиться в том, что мы правильно понимаем использование команды `D`, продифференцируем семь функций, производные которых мы хорошо знаем, а именно, функцию $x \mapsto x^n$, а также функции exp, ln, cos, sin, arcsin и arccos. Следующее вычисление, как раз, и состоит в нахождении этих семи производных:

```
In[127]:= Map[D[#[x],x]&,{#^n&,Exp,Log,Cos,Sin,ArcSin,ArcCos}]
Out[127]={n*x^(-1+n),E^x,1/x,-Sin[x],Cos[x],
                              1/Sqrt[1-x^2],-1/Sqrt[1-x^2]}
```

Тому, кто раньше не имел дела с системами компьютерной алгебры, формат этого вычисления может показаться несколько странным, поэтому прокомментируем наиболее важные моменты. Во-первых, мы вычисляем *список* производных. Это совершенно необходимо, если мы хотим провести несколько вычислений в одной клетке. Для этого мы приводим список функций, которые хотим продифференцировать, и применяем к *элементам* этого списка дифференцирование `D[#[x],x]&` при помощи функции `Map`. Дифференцирование $f \mapsto f'$ задается здесь в формате **анонимной функции**, `D[#[x],x]&`. Напомним, что при этом `Slot`, в операторной записи `#`, обозначает *аргумент* чистой функции, в данном случае неизвестную функцию от $x$, которую мы хотим продифференцировать, а `Function`, в операторной записи `&`, обозначает *применение* чистой функции, в данном случае функции дифференцирования по $x$. Обратите внимание, что в нашем примере аргумент анонимной функции имеет вид `#[x]`, а не просто `#`, так что эта неизвестная функция рассматривается именно как функция от того самого $x$, по которому производится дифференцирование.

Может быть чуть проще понять использование этой конструкции во втором примере, когда мы задаем в таком формате функцию возведения в $n$-ю

степень. А именно, с точки зрения Mathematica выражение #^n& является *именем* функции $t \mapsto t^n$ в том же самом смысле, в котором sin является *именем* функции $x \mapsto \sin(x)$. Используемый формат имени без явного именования аргументов называется форматом *анонимной функции*. При некотором навыке использование анонимных функций становится одним из основных средств программирования в Mathematica. Однако начинающему, вероятно, гораздо удобнее использовать формат *чистой функции*, Function[t,t^n] или, в полной форме Function[t,Power[t,n]], в котором явно указывается, что куда переходит. В этом формате D[#[x],x]& выглядит как Function[f,D[f[x],x]], где *явно* указано, что функции $f$ сопоставляется ее производная по $x$.

Таким образом, вычисление Function[f,D[f[x],x]][Cos] даст нам -Sin[x], как и просто вычисление D[Cos[x],x]. Чуть более витиеватый формат понадобился нам для того, чтобы вычислить одновременно производные нескольких функций — и то только потому, что мы не знали настоящего имени дифференцирования. В действительности, оператор дифференцирования имеет имя, он называется Derivative[1] или, в операторной записи, '. Таким образом, мы могли бы провести то же вычисление с использованием команды Derivative[1] вместо команды D:

```
In[128]:= Map[#'&,{#^n&,Exp,Log,Cos,Sin,ArcSin,ArcCos}]
Out[128]={n#1^(-1+n)&,E^#1&,1/#1&,-Sin[#1]&,
                    Cos[#1]&,1/Sqrt[1-#1^2]&,-1/Sqrt[1-#1^2]&}
```

Разумеется, первая строчка является просто сокращением строчки

```
In[129]:= Map[Derivative[1],{#^n&,Exp,Log,Cos,Sin,ArcSin,ArcCos}]
```

вычисление которой даст точно такой же результат. Мы видим, что в этом случае вычисляются не *значения* производных, а сами производные, *как функции*. Обратите внимание на появляющееся здесь вместо # обозначение аргумента через #1. Дело в том, что если у анонимной функции несколько аргументов, то они обозначаются #1,#2,#3 и так далее. В данном случае программа не знает, в составе какого более сложного выражения мы собираемся вызвать эти производные, и нумерует их аргументы с тем, чтобы при появлении следующего аргумента дать ему имя #2 и т.д.

• **Kettenregel, Produktregel, usw.** Разумеется, Mathematica знает все основные правила вычисления производных, причем следующее вычисление показывает, что она может применять их в символьном виде к неизвестным функциям. Вот как с ее точки зрения выглядят формулы дифференцирования композиции, произведения и частного двух функций, известные под народными названиями Kettenregel, Produktregel и Quotientenregel, которые дал им фон Лейбниц:

```
In[130]:= {D[f@g[x],x],D[f[x]*g[x],x],D[f[x]/g[x],x],}
Out[130]={f'[g[x]]*g'[x],g[x]*f'[x]+f[x]*g'[x],
                          f'[x]/g[x]-f[x]*g'[x]/f[x]^2}
```

Обратите внимание, что $f$ и $g$ обозначают здесь произвольные функции.

Разумеется, тем более Mathematica сможет применить эти формулы к конкретным функциям.

**Комментарий.** Первая из этих формул — это **формула дифференцирования композиции**, которая по-русски испокон веку называлась **цепным правилом** (Kettenregel, chain rule). К сожалению в последние десятилетия под влиянием украинского языка в русской учебной литературе по так называемой высшей математике укоренился полонизм дифференцирование сложной функции. В польском языке это словообразование вполне оправдано, так как композиция функций называется там сложением (ну а сложение функций, естественно, додаванием). Но в русском языке эта малограмотная калька, впервые появившаяся в украинском переводе учебника Банаха, ничем не подкреплена и ее употребление свидетельствует либо о преступной некомпетентности (если писатель не знает о происхождении этого выражения из польского funkcja złożona), либо, в противном случае, о чудовищном манеризме. Кстати, в старых русских учебниках и сама композиция функций называлась *по-простому* **функцией от функции** (fonction de fonction), что, конечно, тоже гораздо лучше, чем мифические *сложные* функции.

Следующий диалог иллюстрирует разницу между $f^{-1}$ и $1/f$:

```
In[131]:= {D[InverseFunction[f][x],x],D[1/f[x],x]}
Out[131]={1/f'[f^(-1)[x]],-f'[x]/f[x]^2}
```

При большом желании можно даже выяснить, когда $f^{-1} = 1/f$ (*чрезвычайно* редко!)

Вот, кстати, **правило дифференцирования степени** (Potenzregel), которое уже далеко не всегда приводится в элементарных учебниках:

```
In[132]:= D[f[x]^g[x],x]
Out[132]=f[x]^g[x]*(g[x]*f'[x]/f[x]+Log[f[x]]*g'[x])
```

Следующий пример показывает, что система действительно полностью отдает себе отчет в том, как применяется цепное правило:

```
In[133]:= {D[ArcCos[Log[x]],x],D[Log[ArcCos[x]],x]}
Out[133]={1/(x*Sqrt[1-Log[x]^2]),-1/(Sqrt[1-x^2]*ArcCos[x])}
```

После этого ясно, что любое другое подобное вычисление является просто упражнением в терпении. Вот пример из тех, которые преподаватели высшей математики любят давать на зачетах:

```
In[134]:= D[Log[ArcCos[x]]/ArcCos[Log[x]],x]
Out[134]=-1/(Sqrt[1-x&2]*ArcCos[x]*ArcCos[Log[x]])+
              Log[ArcCos[x]]/(x*ArcCos[Log[x]]^2*Sqrt[1-Log[x]^2])
```

• **Высшие производные.** Понятно, что значение второй производной $f''(x)$ функции $f$ по $x$ можно вычислить при помощи D[D[f[x],x],x], однако в языке системы предусмотрены и две более короткие записи этого выражения, а именно D[f[x],x,x] и D[f[x],{x,2}]. В действительности внутренним образом все эти три выражения интерпретируются абсолютно одинаково, а именно как Derivative[2][f][x], т.е. как *значение* Derivative[2][f] в точке $x$. В свою очередь Derivative[2][f] обозначает собственно вторую производную $f''$, рассматриваемую *как функцию*. Ну и, наконец, Derivative[2] обозначает *дифференциальный оператор*, сопоставляющий функции ее вторую производную. Именно для того, чтобы

обращаться по имени к этому оператору Mathematica и интерпретирует Derivative как функцию *одного* аргумента, а именно, *порядка*!!! У функции Derivative[n] тоже один аргумент, а именно *функция*, $n$-ю производную которой мы хотим вычислить. Наконец, у функции Derivative[n][f] снова один аргумент, а именно, *точка*, в которой мы вычисляем значение $f^{(n)}$. В Модуле 2 мы подробнейшим образом обсуждаем, чем f[x][y] отличается от f[x,y] и от f[{x,y}] и почему в разных ситуациях используются разные форматы. Однако уже сейчас читатель должен запомнить, что вызов Derivative в формате Derivative[n,f,x] является грубейшей синтаксической ошибкой.

Вот, скажем, чему равна вторая производная $x^{x^x}$:

```
In[135]:= Simplify[D[x^x^x,x,x]]
Out[135]=x^(-2+x+x^x)(-1+2*x+x^x+x*(3+x+2*x^x)*Log[x]+
                  x*(2*x+2*x^x+x^(1+x))*Log[x]^2+
                  x^2*(1+2*x^x)*Log[x]^3+x^(2+x)*Log[x]^4))
```

Мы применили функцию Simplify, чтобы система привела получающееся после применения всех обычных правил выражение для производной к чуть более простому виду, вынеся общий множитель и т.д. Напомним, что система не применяет дистрибутивность автоматически и ей нужно явно подсказывать сделать это. Кстати, обратите внимание и на то, что по умолчанию для функции Power используется **правая группировка**, так что x^y^z истолковывается как $x^{(y^z)}$, а вовсе не как $(x^y)^z$. Чтобы получить $(x^y)^z$, необходимо *явным образом* ставить скобки!!! Вычислим для сравнения вторую производную $(x^x)^x$:

```
In[136]:= Simplify[D[(x^x)^x,x,x]]
Out[136]=(x^x)^x*(3+2*Log[x]+(x+x*Log[x]+Log[x^x])^2)
```

Теперь уже совершенно ясно, что D[f[x],x,x,x] и D[f[x],{x,3}] выражает значение $f'''(x)$ третьей производной $f'''$ функции $f$ по $x$, и интерпретируется как Derivative[3][f][x]. В свою очередь функция $f'''$ вычисляется как Derivative[3][f]. Смысл Derivative[n][f] и D[f[x],{x,n}] теперь уже совершенно очевиден.

Разумеется, Mathematica умеет применять дифференцирования высших порядков в символьном виде к неизвестным функциям. Вот несколько первых примеров **формулы Фаа ди Бруно**[37] дифференцирования композиции:

```
In[137]:= ColumnForm[NestList[D[#,x]&,f@g[x],4]]
Out[137]=f[g[x]]
         f'[g[x]]*g'[x]
         g'[x]^2*f''[g[x]]+f'[g[x]]*g''[x]
```

[37] Дональд Кнут предложил называть эту формулу **формулой Арбогаста**, чем очень огорчил официальный Ватикан.

```
3*g'[x]*f''[g[x]]*g''[x]+g'[x]^3*f'''[g[x]]+
        f'[g[x]]*g'''[x]
3*f''[g[x]]*g''[x]^2+6*g'[x]^2*g''[x]*f'''[g[x]]+
        4*g'[x]*f''[g[x]]*g'''[x]+g'[x]^4*f^(4)[g[x]]+
        f'[g[x]]*g^(4)[x]
```

Как обычно, $f^{(n)}(x)$ обозначает значение $n$-й производной $\dfrac{d^n f}{dx^n}$.

• **Частные производные.** Точно такой же формат как для функций одной переменной имеет и вычисление частных производных по одной из переменных. А именно, `D[f[x,y,z],{x,n}]` обозначает $\dfrac{\partial^n f}{\partial x^n}$. С другой стороны, `D[f,x1,x2,x3]` обозначает смешанную производную $\dfrac{\partial}{\partial x_1}\dfrac{\partial}{\partial x_2}\dfrac{\partial f}{\partial x_3}$ — обратите внимание на порядок переменных!! Напомним, что в такой форме мы обращаемся к значениям этих производных. Собственно сама частная производная $\dfrac{\partial^{n_1}}{\partial x_1^{n_1}}\dfrac{\partial^{n_2}}{\partial x_2^{n_2}}\dfrac{\partial^{n_3} f}{\partial x_3^{n_3}}$, рассматриваемая как функция, вызывается посредством `Derivative[n1,n2,n3][f]`. Вот простейший пример

```
In[138]:= {D[Exp[x^2+y^2],x,x],D[Exp[x^2+y^2],x,y],
           {D[Exp[x^2+y^2],y,y]}
Out[138]={2*E^(x^2+y^2)+4*E^(x^2+y^2)*x^2,
          4*E^(x^2+y^2)*x*y,
          2*E^(x^2+y^2)+4*E^(x^2+y^2)*y^2}
```

Как всегда, `Mathematica` может считать в символьном виде с неизвестными функциями. Вот, скажем, как она понимает цепное правило для функций двух переменных. Следующие формулы можно найти на первых страницах любого учебника по анализу функций нескольких переменных (`multivariable calculus`):

```
In[139]:= Simplify[D[f[g[x,y],h[x,y]],x]],
Out[139]={f^(0,1)[g[x,y],h[x,y]]*h^(0,1)[x,y]
          g^(0,1)[x,y]*f^(1,0)[g[x,y],h[x,y],
In[139]:= Simplify[D[f[g[x,y],h[x,y]],y]]
Out[139]=f^(1,0)[g[x,y],h[x,y]]*g^(1,0)[x,y]
         f^(0,1)[g[x,y],h[x,y]]*h^(1,0)[x,y]
```

Как обычно, через $f^{(i,j)}(x,y)$ здесь обозначена функция $\dfrac{\partial^i}{\partial x^i}\dfrac{\partial^j f}{\partial x^j}(x,y)$. А вот соответствующее правило для вторых производных большинство читателей уже вряд ли видело в явном виде.

```
In[140]:= Simplify[D[f[g[x,y],h[x,y]],x,x]]
Out[140]=f^(0,2)[g[x,y],h[x,y]]*h^(1,0)[x,y]^2+
         2*g^(1,0)[x,y]*h^(1,0)[x,y]*f^(1,1)[g[x,y],h[x,y]]+
         g^(1,0)[x,y]^2*f^(2,0)[g[x,y],h[x,y]]+
         f^(1,0)[g[x,y],h[x,y]]*g^(2,0)[x,y]+
         f^(0,1)[g[x,y],h[x,y]]*h^(2,0)[x,y]
```

```
In[141]:= Expand[D[f[g[x,y],h[x,y]],x,y]]
Out[141]=h^(0,1)[x,y]*f^(0,2)[g[x,y],h[x,y]]*h^(1,0)[x,y]+
              h^(0,1)[x,y]*g^(1,0)[x,y]*f^(1,1)[g[x,y],h[x,y]]+
              g^(0,1)[x,y]*h^(1,0)[x,y]*f^(1,1)[g[x,y],h[x,y]]+
              f^(1,0)[g[x,y],h[x,y]]*g^(1,1)[x,y]+
              f^(0,1)[g[x,y],h[x,y]]*h^(1,1)[x,y]+
              g^(0,1)[x,y]*g^(1,0)[x,y]*f^(2,0)[g[x,y],h[x,y]]
In[142]:= Simplify[D[f[g[x,y],h[x,y]],y,y]]
Out[142]=h^(0,1)[x,y]^2*f^(0,2)[g[x,y],h[x,y]]+
              f^(0,1)[g[x,y],h[x,y]]*h^(0,2)[x,y]+
              g^(0,2)[x,y]*f^(1,0)[g[x,y],h[x,y]]+
              2*g^(0,1)[x,y]*h^(0,1)[x,y]*f^(1,1)[g[x,y],h[x,y]]+
              g^(0,1)[x,y]^2*f^(2,0)[g[x,y],h[x,y]]
```

Явно написать аналог формулы Фаа ди Бруно для высших производных композиции функций нескольких переменнных, ничего при этом не пропустив, определенно находится за пределами комбинаторных способностей обычного человека. Например, уже в `D[f[g[x,y],h[x,y]]x,x,y,y]` входит 46 слагаемых, каждое из которых является произведением двух, трех, четырех или пяти частных производных.

• **Ряд Тэйлора.** Основной командой для вычисления степенных разложений функции $f$ является `Series`. Вызванная в формате

```
Series[f[x],{x,c,m}]
```

эта команда находит первые $m$ членов ряда Тэйлора функции $f$ от $x$ в окрестности точки $c$. В тех случаях, когда `Mathematica` может фактически вычислить производные, она вычисляет их, в противном случае — оставляет их в символьной форме:

```
In[143]:= Series[f[x],{x,c,4}]
Out[143]=f[c]+f'[c]*(x-c)+1/2*f''[c]*(x-c)^2+
              1/6*f'''[c]*(x-c)^3+1/24*f^(4)[c]*(x-c)^4+O[x-c]^5
```

В тех случаях, когда команда `Series` не может найти ряд Тэйлора (например, если значения каких-то производных в точке $c$ бесконечны), она пишет разложение функции в ряд, в который входят также отрицательные и/или дробные степени $x - c$:

```
In[144]:= Series[Exp[Sqrt[x]],{x,0,3}]
Out[144]=1+x^(1/2)+1/2*x+1/6*x^(3/2)+1/24*x^2+
              1/120*x^(5/2)+1/720*x^3+
                    O[x]^(7/2)
```

## § 9. Интегралы

Are you an Analyst? — I am an Oralist.

George Bergmann

$$\int_1^{\sqrt[3]{3}} z^2\,dz\cos(3\pi/9) = \ln(\sqrt[3]{3})$$

Anonimous Analyst[38]

Одной из самых мощных команд системы является команда `Integrate`, при помощи которой вычисляются как неопределенные, так и определенные интегралы от функций одной и нескольких переменных. Обратим внимание, что имплементация команды `Integrate` включает 600 страниц кода на `C` и еще около 500 страниц высокоуровневого кода!

• **Неопределенный интеграл.** Первообразная функции $f$ вычисляется при помощи команды `Integrate`, вызываемой в том же формате, что и команда дифференцирования `D`, а именно, `Integrate[f[x],x]`. Эта команда возвращает *какую-то* первообразную функции $f$. Таким образом, взятие неопределенного интеграла считается операцией обратной дифференцированию:

```
In[145]:= {D[Integrate[f[x],x],x],Integrate[D[f[x],x],x]}
Out[145]={f[x],f[x]}
```

Вот пара простеньких интегралов, которые легко берутся интегрированием по частям:

```
In[146]:= {Integrate[ArcTan[x],x],Integrate[ArcSin[x],x]}
Out[146]={x*ArcTan[x]-1/2*Log[1+x^2],Sqrt[1-x^2]+x*ArcSin[x]}
```

Конечно, система может легко взять любой стандартный интеграл, для вычисления которого не требуется ничего, кроме многократного применения обычных формул для элементарных функций и стандартных подстановок:

```
In[147]:= Simplify[Integrate[Cos[x]^10,x]]
Out[147]=1/10240*(2520*x+2100*Sin[2*x]+600*Sin[4*x]+
                  150*Sin[6*x]+25*Sin[8*x]+2*Sin[10*x])
In[148]:= Integrate[(x+Sqrt[x^2+1])/(x+1-Sqrt[x^2+1]),x]
Out[148]=1/2*(x+x^2+Sqrt[1+x^2]+x*Sqrt[1+x^2]+
              ArcSinh[x]+2*Log[x]-Log[1+Sqrt[1+x^2])
```

Появление гиперболического арксинуса во втором интеграле совершенно естественно, если вспомнить, что $\displaystyle\int \frac{1}{\sqrt{1+x^2}}\,dx = \operatorname{arcsh}(x) + c$. Тем не менее, выводить вручную даже такие совсем простые формулы не слишком приятно.

---

[38] Для удобства `орали́стов` приведем американское чтение этого лимерика: Integral $z$-squared $dz$/ from 1 to the cube root of 3/ times the cosine/ of three $\pi$ over 9/ equals log of the cube root of $e$.

• **Неберущиеся интегралы.** В отличие от вычисления производных вычисление первообразных представляет собой творческое занятие. Тонкий момент здесь состоит в том, что интеграл от элементарной функции уже совершенно не обязательно является элементарной функцией, но никаких очевидных критериев, которые позволяют сказать, когда подобная неприятность случается, нет. Уже композиция двух основных элементарных функций может не иметь элементарного интеграла. Вот несколько совсем простеньких примеров, которые хорошо иллюстрируют возникающие здесь проблемы.

◦ Интеграл $\int \sin(\ln(x))\,dx$ легко берется интегрированием по частям. Каждый из нас должен был проделать следующее вычисление на первом курсе:

```
In[149]:= Integrate[Sin[Log[x]],x]
Out[149]=(1/2)*x*Cos[Log[x]]+(1/2)*x*Sin[Log[x]]
```

◦ Но вот переставив здесь `Sin` и `Log`, мы получим интеграл $\int \ln(\sin(x))\,dx$, в который входит полилогарифм:

```
In[150]:= Integrate[Log[Sin[x]],x]
Out[150]=(-x)*Log[1-E^(2*I*x)]+x*Log[Sin[x]]+
                  (1/2)*I(x^2+PolyLog[2,E^(2*I*x)])
```

где `PolyLog[n,x]` обозначает **полилогарифм** $\mathrm{Li}_n(x)$, который определяется как

$$\mathrm{Li}_n(x) = \sum_{k=1}^{\infty} \frac{z^k}{k^n}.$$

◦ Также и функция $\ln^2(x) = \ln(\ln(x))$ не интегрируется в элементарных функциях. А именно, интеграл $\int \ln^2(x)\,dx$ равен

```
In[151]:= Integrate[Log[Log[x]],x]
Out[151]=x*Log[Log[x]]-LogIntegral[x]
```

Здесь `LogIntegral[x]` обозначает **интегральный логарифм** $\mathrm{li}(x)$, который можно определить как

$$\mathrm{li}(x) = \int_0^x \frac{dt}{\ln(t)}.$$

◦ А вот $\int \sin^2(x)\,dx$ вообще никак не выражается в терминах более известных функций, так что если Вы предложите `Mathematica` вычислить

```
In[152]:= Integrate[Sin[Sin[x]],x]
```

она после некоторого раздумья просто перепишет это выражение. Это значит, что системе не известно никакой более простой или более явной формы для `Integrate[Sin[Sin[x]],x]`. Мы не сомневаемся, что к этому моменту читатель уже в состоянии отличить неберущийся интеграл $\int \sin^2(x)\,dx$ от интеграла $\int \sin(x)^2\,dx = x/2 - \sin(2x)/4$.

◦ Чтобы завершить тему с возведением в квадрат всего, что в $\sin(x)$ возводится в квадрат, вычислим еще интеграл $\int \sin(x^2)\,dx$. С точностью до нормировки ответ называется **синус интегралом Френеля**:

```
In[153]:= Integrate[Sin[x^2],x]
Out[153]=Sqrt[Pi/2]*FresnelS[Sqrt[2/Pi]*x]
```

Этот интеграл, как и **косинус интеграл Френеля** $\int \cos(x^2)\,dx$ или, после нормировки, $\int \cos(\pi x^2/2)\,dx$, часто встречается в оптике.

Да чего там композиции — уже произведения двух основных элементарных функций совершенно не обязательно интегрируются в элементарных функциях (произведение двух производных вообще не обязано быть производной чего бы то ни было). Вот два совсем простых примера. При попытке вычислить $\displaystyle\int \frac{x}{\sin(x)}\,dx$ встречается уже знакомый нам полилогарифм

```
In[154]:= Integrate[x/Sin[x],x]
Out[154]=x*(Log[1-E^(I*x)]-Log[1+E^(I*x)])+
                    I*(PolyLog[2,-E^(I*x)]-PolyLog[2,E^(I*x)])
```

А вот $\displaystyle\int \frac{\sin(x)}{x}\,dx$ имеет специальное название – **интегральный синус**:

```
In[155]:= Integrate[Sin[x]/x,x]
Out[155]=SinIntegral[x]
```

Через `SinIntegral[x]` и `CosIntegral[x]` в `Mathematica` как раз и обозначаются интегральный синус и **интегральный косинус** соответственно:

$$\mathrm{Si}(x) = \int_0^x \frac{\sin(t)}{t}\,dt, \qquad \mathrm{Ci}(x) = -\int_x^\infty \frac{\cos(t)}{t}\,dt,$$

Каждый может узнать все эти факты секунд за 30, со скоростью нужной, чтобы набрать эти интегралы на клавиатуре. Заметим, что большая часть этих интегралов вообще не упоминается ни в общем курсе математического анализа, ни в большинстве стандартных учебников. Впрочем, и в этом отношении замечательное руководство Владимира Антоновича Зорича[39] оказывается с *огромным* отрывом лучшим из всех учебных текстов на русском языке. А именно, в упражнениях 6 и 7 на стр. 380–381 первого тома не только перечисляются все эти интегралы, но и объясняется их происхождение из **интегральной экспоненты**:

$$\mathrm{Ei}(x) = \int_{-\infty}^x \frac{e^t}{t}\,dt,$$

на языке `Mathematica`, естественно, `ExpIntegralEi[x]`.

Поэтому, если Вы не являетесь специалистом по некоторым разделам анализа или математической физики, и не читали Зорича, то Вы, скорее

[39] В.А.Зорич, Математический анализ. т. I,II. — М., МЦНМО, 2002, с.1–657, с.1–787.

всего, вообще не видели большинства этих интегралов. А теперь скажите, сколько времени Вам понадобилось бы, чтобы извлечь эту информацию из традиционного справочника, ну скажем, из[40]?

• **Определенный интеграл.** Определенный интеграл $\int_a^b f(x)\,dx$ вычисляется при помощи той же самой команды `Integrate`, которая в этом случае вызывается в формате `Integrate[f[x],{x,a,b}]`, где явно указаны пределы интегрирования. `Mathematica` знает, как дифференцировать определенный интеграл по пределам интегрирования:

```
In[156]:= D[Integrate[f[t],{t,g[x],h[x]}],x]
Out[156]=-f[g[x]]*g'[x]+f[h[x]]*h'[x]
```

А вот формула Ньютона—Лейбница:

```
In[157]:= f[x_]:=D[g[x],x]; Integrate[f[x],{x,a,b}]
Out[157]=-g[a]+g[b]
```

Еще раз объясним, что здесь происходит. В строке ввода две команды, разделенные точкой с запятой. Первая из них — это операция отложенного присваивания `:=` при помощи которой мы задаем равенство $f = g'$. Как всегда при этом бланк `_` используется для того, чтобы сообщить системе, что $f(x) = g'(x)$ *для всех* $x$. Вместо этого мы могли при желании произвести *немедленное* присваивание `f=Derivative[1][g]`, но в этом случае нужно задавать саму функцию $f$, а не ее значения! После этого вторая команда предлагает системе проинтегрировать $f$.

Только что сказанное убеждает нас в том, что система сможет взять любой определенный интеграл, если она может взять соответствующий неопределенный интеграл и значения первообразной на концах конечны. Гораздо интереснее, как она будет выкручиваться при возникновении неопределенностей или бесконечных значений! Оказывается, это тоже не представляет проблемы:

```
In[158]:= Integrate[Log[x],{x,0,1}]
Out[158]=-1
```

Мы уже знаем, что интеграл от функции $\sin^2(x) = \sin(\sin(x))$ не берется. Тем не менее, некоторые определенные интегралы от этой функции берутся явно:

```
In[159]:= Integrate[Sin[Sin[x]],{x,0,Pi}]
Out[159]=Pi*StruveH[0,1]
```

Появляющаяся в этой формуле **функция Струве** `StruveH` является специальным решением неоднородного уравнения Бесселя.

Вот еще один не самый банальный интеграл, вычисление которого обычно сводится к многомерным интегралам, либо интегралам, зависящим от параметра:

---

[40] А.П.Прудников, Ю.А.Брычков, О.И.Маричев, Интегралы и ряды. Элементарные функции. — Наука, 1981. с.1—798.

In[160]:= `Integrate[ArcTan[x]/(x*Sqrt[1-x^2]),{x,0,1}]`

Out[160]=`1/2*Pi*ArcSinh[1]`

По форме ответа *кажется*, что `Mathematica` в лоб вычислила соответствующий неопределенный интеграл и использовала формулу Ньютона—Лейбница. Однако это абсолютно не так, поскольку вычислять неопределенный интеграл $\int \frac{\operatorname{arctg}(x)}{x\sqrt{1-x^2}}\,dx$ она не умеет[41]!!! Таким образом, для нас остается загадкой, каким образом `Mathematica` получила правильный ответ в данном случае! Впрочем, даже зная этот ответ, неспециалист, скорее всего, далеко не сразу заметит, что это в точности $\frac{\pi}{2}\ln(1+\sqrt{2})$.

• **Несобственные интегралы.** Одними из самых знаменитых и часто встречающихся в приложениях определенных интегралов являются **интеграл Эйлера—Пуассона**, известный также как **Гауссов интеграл**[42], и **интеграл Дирихле**:

$$\int_{-\infty}^{\infty} e^{-x^2}\,dx = \sqrt{\pi}, \qquad \int_{-\infty}^{\infty} \frac{\sin(x)}{x}\,dx = \pi.$$

Посмотрим, может ли `Mathematica` взять эти интегралы, ведь пределы интегрирования в них бесконечны. Оказывается, и это не представляет никакой проблемы:

In[161]:= `Integrate[E^(-x^2),{x,-Infinity,Infinity}]`

Out[161]=`Sqrt[Pi]`

In[162]:= `Integrate[Sin[x]/x,{x,-Infinity,Infinity}]`

Out[162]=`Pi`

Вот еще один очень красивый определенный интеграл, соответствующий неопределенный интеграл выражается в терминах полилогарифма, но при интегрировании от 1 до $\infty$ все исчезает, остается (ПРОСТРАНСТВО, ЗВЕЗДЫ И ПЕВЕЦ) **константа Каталана** $C$:

In[163]:= `Integrate[Log[x]/(1+x^2),{x,1,Infinity}]`

Out[163]=`Catalan`

Заметим, что $\int_0^1 \frac{\ln(x)}{1+x^2}\,dx = -C$. Если хотите испытать культурный шок, то попробуйте теперь вычислить $\int_0^{\infty} \frac{\ln(x)}{1+x^2}\,dx$.

---

[41] Это один из *немногих* известных нам интегралов, которые `Mathematica` *не* умеет вычислять. Впрочем, в подобных случаях больной все равно должен обращаться к доктору, поскольку на стр. 268 книги "Интегралы и ряды" этого интеграла тоже нет, а есть только более простой интеграл $\int \frac{x\operatorname{arctg}(x)}{\sqrt{1-x^2}}\,dx$. **Упражнение для профессионалов**: зная ответ придумайте подстановку, вычисляющую этот интеграл.

[42] В теории вероятностей интеграл $\operatorname{erf}(x) = \int_0^x e^{-t^2}\,dt$, рассматриваемый как функция от верхнего предела, называется еще **функцией ошибок** или **интегралом ошибок** = `error function`.

Попытка вычислить интеграл `Integrate[1/x,{x,-1,1}]` не приведет к большому успеху, так как система выдаст следующее сообщение об ошибке

```
Integrate:  Integral of 1/x does not converge on {-1,1}.
```

В подобном случае можно попытаться вычислить главное значение интеграла. Вычисление

```
In[164]:= Integrate[1/x,{x,-1,1},PrincipalValue->True]
```

даст значение 0. Правило `PrincipalValue->True` меняет установку одной из **опций** функции `Integrate`. Это типичный пример того, как настройка опций меняет способ работы функций и влияет на получающийся ответ.

• **Кратные интегралы.** Команда `Integrate` служит и для вычисления кратных интегралов. Скажем, при вычислении двойного интеграла она вызывается в формате `Integrate[f[x,y],{x,a,b},{y,c,d}]`.

Обратите внимание, что, как и в командах наподобие `Table` или `Array`, порядок итераторов здесь таков: внутренние итераторы изображаются последними. Таким образом, `Integrate[f[x,y],{x,a,b},{y,c,d}]` это

$$\int_a^b \int_c^d f(x,y)\,dy\,dx, \qquad \text{а вовсе не} \qquad \int_c^d \int_a^b f(x,y)\,dx\,dy.$$

Заметим, что при вычислении кратных интегралов результат в большинстве случаев следует упрощать, так как без этого система возвращает его `as is`, как сумму большого количества слагаемых, в том виде, как они возникают. Вот один из классических двойных интегралов,

```
In[165]:= Simplify[Integrate[1/Sqrt[x^2+y^2],{x,-1,1},{y,-1,1}]]
Out[165]=2*(2*ArcSinh[1]-Log[-1+Sqrt[2]]+Log[1+Sqrt[2]])
```

В этот момент у каждого, кто когда-либо преподавал `Multivariable Calculus`, да и просто у того, кто с самого начала внимательно читал этот параграф, закрадываются подозрения, что это все еще не окончательный ответ. Еще одна попытка заканчивается полной победой:

```
In[166]:= FullSimplify[Integrate[1/Sqrt[x^2+y^2],
                                 {x,-1,1},{y,-1,1}]]
Out[166]=8*ArcSinh[1]
```

Математик — это тот, кто никогда не останавливается на достигнутом, однако такого тройного интеграла, пожалуй, не считали даже наши студенты:

```
In[167]:= FullSimplify[Integrate[1/Sqrt[x^2+y^2+z^2],
                                 {x,-1,1},{y,-1,1},{z,-1,1}]]
Out[167]=-2*(Pi-4*ArcCsch[Sqrt[2]]+Log[-8(-2+Sqrt[3])]-
                                 6*Log[1+Sqrt[3]])
```

Естественно, пределы интегрирования сами могут быть функциями от других аргументов:

```
In[168]:= Integrate[1,{x,-1,1},{y,-Sqrt[1-x^2],Sqrt[1-x^2]},
                    {z,-Sqrt[1-x^2-y^2],Sqrt[1-x^2-y^2]}]
```

Out[168]=`4*Pi/3`

Кстати, что это такое мы тут посчитали?

• **Численное интегрирование.** Мы уже знаем, что в элементарных функциях интеграл $\int \sin(\sin(x))\,dx$ не берется. Но, конечно, для любых конкретных $a < b$ значение определенного интеграла $\int_a^b \sin(\sin(x))\,dx$ можно найти с любой точностью. Для этой цели проще всего пользоваться командой численного интегрирования `NIntegrate`, синтаксис которой не отличается от синтаксиса команды `Integrate` вызываемой для вычисления определенного интеграла. По умолчанию `NIntegrate` проводит вычисления с машинной точностью (чуть меньше 16 знаков после запятой). Вот пример ее использования:

In[169]:= `NIntegrate[Sin[Sin[x]],{x,1,2}]`

Out[169]=`0.81645`

Команда `NIntegrate` не может, конечно, дать такие значения, как $\sqrt{\pi}$. Обычно эта команда работает *чрезвычайно* быстро, так как по умолчанию она вначале компилирует программу вычисления подынтегральной функции и использует небольшую глубину рекурсии. Разумеется, это значит, что в случае часто осциллирующих функций (`highly oscillatory functions`) применение команды `NIntegrate` — как и вообще *любое* применение приближенных вычислений!!! — может приводить к *очень* серьезным погрешностям. В частности, поэтому ПРИБЛИЖЕННЫЕ ВЫЧИСЛЕНИЯ *никогда* НЕ СЛЕДУЕТ ИСПОЛЬЗОВАТЬ ВНУТРИ РЕКУРРЕНТНЫХ ИЛИ ИТЕРАТИВНЫХ ПРОЦЕДУР, где уже за несколько десятков итераций ошибка может достичь десятков процентов. Разумеется, в подобных случаях система обычно предупреждает о возможных проблемах и для получения правильного ответа следует использовать символьные, а не численные вычисления, даже если соответствующие функции работают чуть дольше. Вот пример, когда применение функции `NIntegrate` без настройки опций приводит к ошибке уже в 7-ом знаке:

In[170]:= `NIntegrate[Sin[1/x/10000],{x,0,1}]`

Это вычисление производится очень быстро, но дает, увы, неверный ответ `0.000963519`. Разумеется, система сообщает о возможной ошибке:

```
NIntegrate:
  NIntegrate failed to converge to prescribed accuracy after 9
  recursive bisections in x near {x} = {2.07884 * 10^-8}...
```

С нашей точки зрения ПРАВИЛЬНАЯ ПОЛИТИКА во всех подобных случаях СОСТОИТ В ПРОВЕДЕНИИ БЕЗОШИБОЧНЫХ ВЫЧИСЛЕНИЙ, с вычислением приближенного значения на самом последнем шаге:

In[171]:= `N[Integrate[Sin[1/x/10000],{x,0,1}]]`

Это вычисление требут несколько большего времени, но зато дает правильный ответ `0.000963312`. Разумеется фанатики численных методов могут добиться такой же точности и другими способами, например, просто увеличив глубину рекурсии:

In[172]:= `NIntegrate[Sin[1/x/10000],{x,0,1},MaxRecursion->20]]`

В этом случае система так же вернет правильный ответ `0.000963312` и затратит на его нахождение почти столько же времени, как и в исходном примере.

## § 10. Вектора и матрицы

> Как представитель комбинаторики, я идентифицирую реальность с матрицами из 0 и 1.
>
> Дональд Кнут. Математическая типография

Сейчас мы совсем коротко обсудим основы представления, генерации и изображения векторов и матриц в системе `Mathematica`. При этом мы ограничиваемся лишь *абсолютным минимумом*, необходимым для решения простейших задач линейной алгебры.

• **Списки.** Одной из объединяющих концепций, вокруг которых организован язык `Mathematica`, является понятие **списка** (`List`). С точки зрения языка системы и внутреннего представления данных все на свете — множества, наборы, последовательности, векторы, матрицы, тензоры, и т.д. — трактуется как список. Более того, грамотное программирование на языке `Mathematica` состоит в том, чтобы избегать явного использования циклов, а вместо этого объединять переменные, команды, уравнения, условия, подстановки, ... в списки.

В `Mathematica` список $(x_1, \dots, x_n)$ с компонентами $x_1, \dots, x_n$ обозначается `{x1,...,xn}` или, в полной форме, `List[x1,...,xn]`. Подчеркнем, что с математической точки зрения список длины $n$ представляет собой **упорядоченную $n$-ку** ($n$-`tuple`) или, если мы не хотим явно упоминать ее длину[43], **тупель**. По определению два списка равны, если равны их длины и все их соответствующие компоненты:

$$(x_1, \dots, x_m) = (y_1, \dots, y_n) \qquad \Longleftrightarrow \qquad m = n \quad \text{и} \quad x_i = y_i,\ i = 1, \dots, n$$

В программировании существует устойчивая традиция называть компоненты списка **элементами** этого списка. Как мы только что отметили, при определении равенства списков учитываются кратности и порядок их элементов!

**Комментарий.** В русской учебной литературе нет единства по поводу того, как следует переводить термин `Tupel`, `tuple`. Под влиянием программирования сегодня `tuple` и в математических работах чаще всего переводится как `список`. Некоторые авторы предлагали перевод `кортеж`, но работающие математики никогда им не пользуются. Поэтому мы пропагандируем термин `тупель` и его производные `дупель`, `трипель`, `квадрупель`, `квинтупель`, `секступель`, `септупель` и `октупель` для обозначения упорядоченных пар, троек, четверок, пятерок, шестерок, семерок и восьмерок.

Таким образом, список `{a,b,a,c}` или, в полной форме, `List[a,b,a,c]` представляет именно тупель $(a, b, a, c)$, а вовсе не набор или множество с элементами $a, b, a, c$. Это значит, например, что в ответ на вопрос

[43] Например, при $n = k$.

```
TrueQ[{a,b,c}=={b,a,c}]
```

система возвратит `False` — если, конечно, переменным $a, b$ не были ранее присвоены одинаковые значения!

Применение к списку команды `Sort` расставляет его элементы в некотором *естественном* порядке, связанном с их записью как **выражений** в языке `Mathematica`. Таким образом, эффективно применение `Sort` заставляет систему *игнорировать* порядок элементов списка, делая два списка равными, в том и только том случае, когда совпадают представленные ими **наборы** $[x_1, \dots, x_n]$. Напомним, что при определении равенства наборов учитываются *кратности* входящих в них элементов, но не их порядок:

```
In[173]:= TrueQ[Sort[{a,b,a,c}]==Sort[{b,c,a,a}]]
Out[173]=True
In[174]:= TrueQ[Sort[{a,b,a,c}]==Sort[{b,c,b,a}]]
Out[174]=False
```

С другой стороны, команда `Union` превращает список в **множество**, т.е. заставляет систему игнорировать не только порядок элементов списка, но и их кратности:

```
In[175]:= TrueQ[Union[{a,b,a,c}]==Union[{b,c,b,a}]]
Out[175]=False
```

В системе содержится несколько десятков операций над списками, относящихся к следующим категориям:

∘ теоретико-множественные операции;

∘ манипуляции с частями списка: извлечения, вычеркивания, вставки, замены, выборки, и т.д.

∘ структурные манипуляции: сортировки, перестановки, выравнивания, разбиения, и т.д.

∘ применение функций к спискам и их частям: `Map`, `Apply`, `Thread`, `Inner`, `Outer` и их многочисленные варианты.

Гибкое применение возможностей этих операций составляет йогу системы `Mathematica`, однако овладение этим искусством требует некоторого ментального тюнинга и многомесячной практики. Вопросы функционального и списочного программирования требуют отдельного основательного рассмотрения, поэтому здесь мы не будем обсуждать тонкие аспекты работы со списками, а ограничимся иллюстрацией применения нескольких простейших встроенных функций к решению стандартных задач линейной алгебры.

• **Векторы.** В простейшем варианте вектор $(x_1, ..., x_n)$ с координатами $x_1, \dots, x_n$ представляется списком `{x1,...,xn}`. При этом координаты вектора могут быть любыми выражениями: числами, символами, списками, матрицами и т.д.

При действиях над векторами все обычные операции трактуются как **покомпонентные**:

```
In[176]:= {u,v,w}+{x,y,z}
Out[176]={u+x,v+y,w+z}
In[177]:= {u,v,w}*{x,y,z}
Out[177]={u*x,v*y,w*z}
```

Более того, арифметические операции имеют атрибут `Listable`. Это значит, что при выполнении этих операций скаляр $x$ отождествляется с вектором $(x, \dots, x)$ подходящей длины. Иными словами,

```
In[178]:= {w+{x,y,z},w*{x,y,z}}
Out[178]={{w+x,w+y,w+z},{w*x,w*y,w*z}}
```

Стоит подчеркнуть, что в такой форме записи не делается различия между строками и столбцами, и вычисление `{a,b,c}.{x,y,z}` или, что то же самое, `Dot[{a,b,c},{x,y,z}]`, даст `a*x+b*y+c*z`. Тот же результат получится и при вычислении `Inner[Times,{a,b,c},{x,y,z}]`. Однако, интерпретировать этот результат можно совершенно по разному, в зависимости от того, как мы представляем себе исходные векторы и чем являются их элементы.

- произведение строки $(a, b, c)$ на столбец $(x, y, z)^t$;
- скалярное произведение строки $(a, b, c)$ на строку $(x, y, z)$;
- скалярное произведение столбца $(a, b, c)^t$ на столбец $(x, y, z)^t$;
- линейную комбинацию строк/столбцов $x, y, z$ с коэффициентами $a, b, c$;

и многими другими способами.

**Комментарий.** В действительности, `{x,y,z}` не является ни строкой, ни столбцом. Интересно, что Вольфрам считает это достоинством: Because of the way `Mathematica` uses lists to represent vectors and matrices, you never have to distinguish between "row" and "column" vectors. Как мы сейчас увидим, вектор $(x, y, z)$, рассматриваемый как строка, должен записываться в виде `{{x,y,z}}`, а рассматриваемый как столбец — в виде `{{x},{y},{z}}`. С нашей точки зрения это нарушение симметрии между строками и столбцами является одним из самых серьезных концептуальных дефектов системы. Конечно, оно никак не сказывается на практических вычислениях. С другой стороны, в данном случае `Mathematica` всего лишь следует обычной при элементарном подходе практике, когда не делается никакого различия между правыми и левыми векторными пространствами. В действительности, даже над полем необходимо тщательно различать **правые векторные пространства**, координаты векторов в которых записываются *столбцами*, а базисы, соответственно, строчками (состоящими из столбцов) и **левые векторные пространства**, координаты векторов в которых записываются *строчками*, а базисы, соответственно, столбцами (состоящими из строчек). Стоит только начать различать столбцы и строки и все в линейной алгебре сразу становится на свои места. Например, из четырех матриц $g$, $g^t$, $g^{-1}$ и $g^{-t}$, отвечающих за преобразование координат различных геометрических объектов, остаются всего две: ковариантная $g$ и контравариантная $g^{-1}$.

• **Запись матриц.** Матрица в языке `Mathematica` записывается как *список*, составленный из *строк* этой матрицы. Список, элементы которого сами являются списками, называется **вложенным списком** (`nested list`). Таким образом, матрица

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

изображается как {{a,b},{c,d}}. Именно такая форма используется при *вводе* матриц, внутри себя система переводит это выражение в полную форму List[List[a,b],List[c,d]].

Симметрия между строками и столбцами восстанавливается при помощи команды Transpose, переводящей матрицу в транспонированную матрицу, строки которой совпадают со столбцами исходной:

```
In[179]:= Transpose[{{a,b},{c,d}}]
Out[179]={{a,c},{b,d}}
```

По умолчанию команда Transpose переставляет *два верхних уровня* вложенности списков. Таким образом, например, команда Transpose, примененная к *списку* матриц будет переставлять строки этих матриц между собой, а вовсе не транспонировать сами эти матрицы. Для одновременного транспонирования списка матриц нужно применять Transpose к *элементам* этого списка при помощи команды Map:

```
In[180]:= Transpose[{{{a,b},{c,d}},{{e,f},{g,h}}}]
Out[180]={{{a,b},{e,f}},{{c,d},{g,h}}}
In[181]:= Map[Transpose,{{{a,b},{c,d}},{{e,f},{g,h}}}]
Out[181]={{{a,c},{b,d}},{{e,g},{f,h}}}
```

Из описанного представления матриц и того, что было сказано выше об операциях над векторами следует, что при действиях над матрицами все обычные операции тракутются как **покомпонентные**:

```
In[182]:= {{a,b},{c,d}}+{{e,f},{g,h}}
Out[182]={{a+e,b+f},{c+g,d+h}}
In[183]:= {{a,b},{c,d}}*{{e,f},{g,h}}
Out[183]={{a*e,b*f},{c*g,d*h}}
```

Таким образом, $*$ (Times) это умножение матриц по Адамару, а вовсе не обычное произведение матриц, которое обозначается . (Dot).

Более спорным представляется применение того же правила к скалярам, которые при этом отождествляются не с кратными единичной, а с кратными **пробной матрицы**, состоящей из одних единиц! Изобразим для примера пробную матрицу степени 3:

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

В свете этого соглашения нас не должен удивлять следующий результат:

```
In[184]:= {{{a,b},{c,d}}+x,{{a,b},{c,d}}*x}
Out[184]={{{a+x,b+x},{c+x,d+x}},{{a*x,b*x},{c*x,d*x}}}
```

• **Генерация векторов и матриц.** Основной командой генерации векторов, матриц и других списков в языке Mathematica является команда

`Table`. Вызванная с одним итератором команда `Table[f[i],{i,m,n}]` порождает список значений функции $f$ в точках $i = m, m+1, \ldots, n$. Вот типичный пример использования этой команды:

```
In[185]:= Table[x^i/i!,{i,0,5}]
Out[185]={1,x,x^2/2,x^3/6,x^4/24,x^5/120}
```

А вот забавная вариация на тему этого примера, детально обсуждаемая Анри Пуанкаре в "Новых методах небесной механики":

```
In[186]:= Table[N[10^i/i!],{i,1,30}]
Out[186]={10., 50., 166.667, 416.667, 833.333, 1388.89, 1984.13,
          2480.16, 2755.73, 2755.73, 2505.21, 2087.68, 1605.9,
          1147.07, 764.716, 477.948, 281.146, 156.192, 82.2064,
          41.1032, 19.5729, 8.89679, 3.86817, 1.61174, 0.644695,
          0.24796, 0.0918369, 0.0327989, 0.01131, 0.00376999}
```

Из этой таблицы видно, что отношение $10^i/i!$ вначале очень быстро растет ("расходится в смысле астрономов"), а потом очень быстро убывает ("сходится в смысле математиков").

Вызванная с двумя итераторами команда

```
Table[f[i,j],{i,k,l},{j,m,n}]
```

порождает *вложенный* список значений функции двух аргументов $f$ в парах $(i,j)$, где $i = k, k+1, \ldots, l$, $j = m, m+1, \ldots, n$. Этот список будет организован как матрица, причем итератор $i$ считается **внешним**, а $j$ — **внутренним**, иными словами, $i$ нумерует строки, а $j$ — позиции внутри строк. Таким образом, строки этой матрицы имеют вид $(f_{im}, \ldots, f_{in})$. Еще раз обратите внимание на следующие два ключевых момента в этом определении:

- матрица трактуется как строка, составленная из строк,
- внутренние итераторы пишутся последними.

Иными словами, если мы дадим два определения матрицы `Forward`

```
In[187]:= forwaa[n_]:=Table[If[j==i+1,1,0],{i,1,n},{j,1,n}]
In[188]:= forwbb[n_]:=Table[If[j==i+1,1,0],{j,1,n},{i,1,n}]
```

отличающиеся лишь порядком итераторов, то получившиеся матрицы будут транспонированы друг к другу. Вот как, например, выглядят матрицы `forwaa[4]` и `forwbb[4]` в традиционной математической нотации

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \qquad \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Иными словами, только одна из этих матриц в действительности окажется матрицей `Forward`, вторая будет матрицей `Backward`! Ну а, скажем, упомянутая в предыдущем пункте пробная матрица задается командой `test[n_]:=Table[1,{i,1,n},{j,1,n}]` .

В системе Mathematica существует большое количество других команд генерации списков и мтариц специального вида. Одной из таких команд является DiagonalMatrix[{d1,...,dn}], которая порождает диагональную матрицу с диагональными элементами $d_1, \ldots, d_n$. Однако в тех простых ситуациях, которые мы здесь рассматриваем, вполне достаточно команды Table. Кроме того, как мы обсуждаем ниже, матрицы можно создавать из векторов или, наоборот, более глубоких списков при помощи структурных команд таких, как Partition или Flatten.

• **Части матриц.** Для выделения частей списков в языке Mathematica используется специальный вид скобок — двойные квадратные скобки [[]], являющиеся сокращением команды Part. При этом в соответствии с общими правилами спецификации уровня, детально обсуждаемыми в Модуле 2, x[[i]] — или, что то же самое, Part[x,i] — обозначает $i$-ю часть списка $x$; x[[-i]] — $i$-ю часть с конца; x[[i,j]] — $j$-ю часть его $i$-части и т.д. При этом сами номера позиций тоже могут задаваться списком, а если мы хотим выбрать *все* части на каком-то уровне, то спецификацией All. Приведем несколько примеров применения этих соглашений

◦ x[[i,j]] — элемент $x$ в позиции $(i, j)$;

◦ x,[[{i,j},{k,l}]] — подматрица порядка 2 матрицы $x$, стоящая на пересечении строк с номерами $i, j$ и столбцов с номерами $k, l$;

◦ x[[i]] — $i$-я строка матрицы $x$;

◦ x[[All,j]] — $j$-й столбец матрицы $x$;

◦ Tr[x,List] — главная диагональ матрицы $x$;

Так, например, x[[1]] и x[[-1]] обозначают, соответственно, первую и последнюю строку матрицы $x$, а x[[All,1]] и x[[All,-1]] — ее первый и последний столбец. Еще раз обратите внимание на отсутствие здесь симметрии между строками и столбцами! Столбец матрицы является строкой транспонированной матрицы, поэтому $j$-й столбец матрицы $x$ можно породить также посредством Transpose[x][[j]].

Разумеется, также и главную диагональ матрицы можно определять тысячей других способов, например, напрашивающимся

```
Table[x[[i,i]],{i,1,Length[x]}].
```

Однако встроенная команда Tr[x,List] не только изящнее, но и работает быстрее, хотя разница во времени становится по настоящему заметной только для матриц порядка несколько десятков тысяч.

• **Просмотр матриц.** Чтобы увидеть матрицу не как вложенный список, а в традиционной записи, надо применить к ней одну из команд TableForm или MatrixForm. Скажем, следующая строка определяет одну из самых важных в самой математике и ее приложениях матриц — **матрицу Вандермонда** $V(x_1, \ldots, x_n)$. Собственно говоря, вся теория определителей построена исключительно для анализа этого примера:

```
In[189]:= vandermonde[x_,n_]:=Table[x[j]^i,{i,0,n-1},{j,1,n}]
```

Посмотрим теперь, как выглядит матрица `vandermonde[y,5]`. Это делается при помощи команды `MatrixForm`. Вычислив

In[190]:= `MatrixForm[vandermonde[y,5]]`

мы увидим *примерно* следующий результат:

$$V(y_1,y_2,y_3,y_4,y_5)=\begin{pmatrix}1&1&1&1&1\\ y_1&y_2&y_3&y_4&y_5\\ y_1^2&y_2^2&y_3^2&y_4^2&y_5^2\\ y_1^3&y_2^3&y_3^3&y_4^3&y_5^3\\ y_1^4&y_2^4&y_3^4&y_4^4&y_5^4\end{pmatrix}$$

Разумеется, здесь мы воспроизводим не то, что будет фактически отображено в записной книжке `Mathematica`, а уже облагороженный TEX'овский аутпут типографского качества, получающийся с помощью применения поверх `MatrixForm` команды экспорта `TeXForm`.

Мы сами обычно не печатаем форматные команды `TableForm`, `MatrixForm` и т.д. заранее, а применяем их к уже имеющемуся выражению **в постфиксной форме** (задним числом). Это делается при помощи оператора `//`, примерно так[44]

In[191]:= `vandermonde[y,5] // MatrixForm`

В первом приближении различие между командами `TableForm` и `MatrixForm` такое же, как между `matrix` и `pmatrix` в TEX'е. Иными словами, `TableForm[x]` выводит элементы матрицы $x$ в виде таблицы, а `MatrixForm[x]` — в традиционной математической записи с использованием круглых скобок.

**Комментарий.** В действительности, кроме этого очевидного отличия, между командами `TableForm` и `MatrixForm` имеется большое количество тонких различий в том, как они по умолчанию трактуют горизонтальное (слева, сверху, по центру, по десятичной точке, по фиксированному символу и т.д.) и вертикальное (по верху, по низу, по центру, по базовой линии, по оси) выравнивание строк и столбцов и индивидуальных элементов внутри строки или столбца. Однако для большинства пользователей, которые интересуются *результатом* вычисления, а не получением типографского продукта профессионального или полупрофессионального качества, подобные тонкости не имеют большого значения.

• **Матричные единицы.** Самыми важными с точки зрения профессионального математика матрицами являются **стандартные матричные единицы** $e_{ij}$, у которых в позиции $(i,j)$ стоит 1 и 0 во всех остальных местах. Матрицы $e_{ij}$, $1\le i,j\le n$, образуют базис $M(n,K)$ как векторного пространства над $K$. Более того, это **мультипликативный базис**, в том смысле, что произведение двух базисных элементов либо равно 0, либо снова представляет собой базисный элемент: $e_{ij}e_{hk}=\delta_{jh}e_{ik}$. Таким образом, $e_{ij}e_{hk}=0$, если $j\ne h$, в то время как $e_{ij}e_{jk}=e_{ik}$. Вот одно из возможных определений этих матриц в языке `Mathematica`. Конечно, теперь мы должны явно указывать не только $i,j$, но и порядок $n$:

[44]Тот же вид результата может быть получен при выборе варианта представления `TraditionalForm` подменю `Convert To` - щелчок правой кнопкой мыши в пределах ячейки.

```
In[192]:= e[n_,i_,j_]:=Table[If[h==i,1,0]*If[k==j,1,0],
                                                {h,1,n},{k,1,n}]
```

Еще раз обратите внимание на несколько уже встречавшихся нам принципиальных моментов:

- использование `_ Blank` и `:= SetDelayed` для определения функции;
- использование `== Equal` в уравнениях `i==h` и `j==k`;
- использование условного оператора `If[condition,x,y]`, возвращающего $x$, если `condition==True` и $y$, если `condition==False`.

**Комментарий.** Как мы обсуждаем в дальнейшем, в общем случае намного разумнее вызывать оператор `If` с четырьмя аргументами, в формате `If[condition,x,y,z]`, явным образом предписывая, что следует делать в случае, когда система не может решить, выполняется условие `condition` или нет. Однако здесь мы используем этот оператор в чисто иллюстративных целях и только в простейшей ситуации сравнения небольших целых чисел, когда у системы не должно быть никаких сомнений, равны они или нет. В действительности вместо `If[i==h,1,0]` мы могли бы воспользоваться встроенной функцией `KroneckerDelta`, но это чуть длиннее.

Посмотрим теперь, как выглядят стандартные матричные единицы степени 3. Применив к ним форматную команду `MatrixForm`

```
In[193]:= Map[MatrixForm,Flatten[Table[e[3,i,j],{i,1,3},{j,1,3}],1]]
```

мы увидим следующее:

$$\begin{pmatrix}1&0&0\\0&0&0\\0&0&0\end{pmatrix},\quad \begin{pmatrix}0&1&0\\0&0&0\\0&0&0\end{pmatrix},\quad \begin{pmatrix}0&0&1\\0&0&0\\0&0&0\end{pmatrix},$$

$$\begin{pmatrix}0&0&0\\1&0&0\\0&0&0\end{pmatrix},\quad \begin{pmatrix}0&0&0\\0&1&0\\0&0&0\end{pmatrix},\quad \begin{pmatrix}0&0&0\\0&0&1\\0&0&0\end{pmatrix},$$

$$\begin{pmatrix}0&0&0\\0&0&0\\1&0&0\end{pmatrix},\quad \begin{pmatrix}0&0&0\\0&0&0\\0&1&0\end{pmatrix},\quad \begin{pmatrix}0&0&0\\0&0&0\\0&0&1\end{pmatrix}.$$

Стоит подчеркнуть здесь один момент, не встречавшийся до сих пор. Дело в том, что команда `Table[e[3,i,j],{i,1,3},{j,1,3}]` создает вовсе не список из девяти стандартных матричных единиц, а *вложенный* список `list`, состоящий из *трех* списков в каждом из которых *три* матричные единицы! Таким образом, применение команды `MatrixForm` непосредственно к элементам этого списку даст нам совершенно не то, что хотелось. Команда `Flatten[list,1]` служит как раз для того, чтобы убрать *один* лишний вложенный уровень в `list`, а именно, *верхний*, и создать список из *девяти* матричных единиц (каждая из которых, конечно, сама является матрицей, т.е. вложенным списком!!) Применение команды `Flatten[list]` не достигло бы этой цели, так как убрало бы *все* вложенные уровни, создав список из 81 нуля и единицы.

• **Разреженные матрицы.** Разумеется, уже для матриц степени 100–1000 подобные определения становятся чрезвычайно грубыми и непрактичными. В языке `Mathematica` предусмотрен специальный инструмент для генерации матриц, почти все элементы которых равны между собой (скажем, равны 0 или 1). Это команда `SparseArray`, которая порождает объект специального формата **разреженный массив**, в котором хранятся не все элементы матрицы, а только ее размеры, общий элемент и список позиций и элементов в них, где они отличаются от общего элемента. Разумеется, подобный объект должен обрабатываться специальными алгоритмами, так что большая часть обычных команд для работы с матрицами будут рассматривать его как **сырой объект**. Это значит, что для проведения матричных вычислений при помощи стандартных функций необходимо еще конвертировать его в обычный матричный формат посредством команды `Normal`. Вот как, скажем, выглядит определение стандартной матричной единицы с использованием команды `Sparse Array`:

```
In[194]:= e[n_,h_,k_]:=Normal[SparseArray[{{h,k}->1},{n,n}]]
```

Разумеется, в подобных случаях `SparseArray` используется в чисто иллюстративных целях или для сокращения записи программ. Ясно, что при фактическом проведении матричных вычислений с матрицами порядка нескольких десятков или сотен тысяч, разреженные матрицы *невозможно* перевести в обычный матричный формат, так как просто для хранения нескольких матриц такого размера — не говоря уже про какие-то вычисления с ними! — в плотном формате потребуется вся доступная на бытовом компьютере память.

## § 11. Линейная алгебра

> 1000 вещей, которые можно сделать с Вашей любимой матрицей.
> Из учебника линейной алгебры

В этом параграфе мы опишем как проводить некоторые простейшие матричные вычисления. Как правило, для наглядности мы будем приводить не фактический аутпут системы `Mathematica`, а традиционную форму матриц, которая получается, если применить к получающимся матрицам `MatrixForm`, а потом экспортировать результат в TEX'овском формате посредством `TeXForm`.

• **Решение систем линейных уравнений.** Решение систем линейных уравнений осуществляется при помощи функций `LinearSolve` и `NullSpace`. А именно, функция `LinearSolve` ищет частное решение неоднородной системы линейных уравнений, а функция `NullSpace` — общее решение однородной системы.

Функция `Linear Solve` вызывается в различных форматах, в зависимости от того, нужно ли нам решить *одну* линейную систему $ax = b$ с данной матрицей $a$ или (как весьма часто бывает в приложениях!) несколько систем с одной и той же матрицей и различными правыми частями. В первом случае функция `LinearSolve` вызывается в формате

```
LinearSolve[a,b]
```

в то время как во втором случае — в формате

```
LinearSolve[a][b]
```

Разница состоит в том, что во втором случае система записывает факторизацию матрицы $a$, которая используется для решения системы $ax = b$. Если матрица $a$ не обратима, то при вызове команды `LinearSolve` в таком формате выдается сообщение:

```
LinearSolve:  The matrix bla-bla-bla is singular
                    so a factorization will not be saved.
```

Приведем пример использования функции `LinearSolve`. Следующие команды определяют случайную матрицу порядка $m \times n$ с целыми коэффициентами между $-100$ и 100 и случайный вектор высоты $n$ с целыми компонентами в том же интервале:

```
In[195]:= rama[m_,n_]:=Table[Random[Integer,{-100,100}],
                                        {i,1,m},{j,1,n}]
In[196]:= cora[n_]:=Table[Random[Integer,{-100,100}],{i,1,n}]
```

А именно, вызванная в формате `Random[Integer,{k,l}]`, команда `Random` генерирует случайное целое число $x$, $k \le x \le l$, естественно, каждый раз новое. Кстати, почему мы задаем `cora[n]` отдельно? Почему мы не можем просто положить `cora[n_]:=rama[1,n]` или `cora[n_]:=rama[n,1]`? Тот, кто внимательно прочел предыдущий параграф, знает, почему! В самом деле, `rama[m,n]` — в том числе, конечно, `rama[1,n]` и `rama[n,1]` — представляют собой *вложенные* списки. Для того, чтобы получить `cora[n]` в правильном формате, мы должны были бы убрать один уровень вложенности, например, применив к этим спискам команду `Flatten`.

Следующая команда фактически генерирует случайную $4 \times 4$-матрицу и случайный вектор высоты 4:

```
In[197]:= nnn=4;aaa=rama[nnn,nnn];bbb=cora[nnn];
```

Вот так выглядит типичный получающийся при этом результат:

$$a = \begin{pmatrix} 66 & -83 & 1 & 63 \\ -25 & -10 & -49 & 99 \\ 74 & 74 & -31 & -6 \\ -92 & 62 & 44 & 70 \end{pmatrix}, \qquad b = \begin{pmatrix} 13 \\ 19 \\ -79 \\ -8 \end{pmatrix}.$$

Определитель такой случайной целочисленной матрицы грандиозен (например, в данном случае он равен 123258840). Поэтому применение команды `LinearSolve` дает

```
In[198]:= LinearSolve[aaa][bbb]
Out[198]={-1056071/2054314,-3961079/6162942,
                        -597458/3081471,-202933/2054314}
```

В то же время команда `NullSpace` возвращает какую-то фундаментальную систему решений уравнения $ax = 0$. Вот, к примеру, что получается

при применении этой команды к уже встречавшейся нам в предыдущем параграфе матрице `test[4]`:

```
In[199]:= NullSpace[test[4]]
Out[199]={{-1,0,0,1},{-1,0,1,0},{-1,1,0,0}}
```

Часто хочется увидеть общее решение неоднородной системы, совмещающее ее частное решение и общее решение соответствующей неоднородной системы. Для этого нужно свести вместе результаты работы команд `LinearSolve` и `NullSpace`. Традиционную форму такого общего решения можно породить, например, при помощи следующей конструкции:

```
In[200]:= gensolution[g_,b_]:=StringForm["``+``",
                MatrixForm[LinearSolve[g,b]],
                Table[a[i],{i,1,Length[NullSpace[g]]}].
                        Map[MatrixForm,NullSpace[g]]]
```

Здесь использована одна из важнейших команд форматирования вывода `StringForm`. Мы часто пользуемся этой командой в тех случаях, когда нам нужно включить результат вычисления в текст. Эта команда служит для включения результатов вычисления в текстовый объект и вызывается в следующем формате:

```
StringForm["ccc``ccc``ccc...",expression1,expression2,...]
```

где, как мы уже знаем, заключенный в двойные кавычки `"..."` объект рассматривается как стринг, `ccc` обозначает произвольную комбинацию знаков, а каждая пара аксанов `` `` `` заменяется на *значение* очередного выражения `expression`.

Теперь вычисление

```
In[201]:= gensolution[test[4],{1,1,1,1}]
```

даст нам следующий результат

$$\begin{pmatrix}1\\0\\0\\0\end{pmatrix}+a_1\begin{pmatrix}-1\\0\\0\\1\end{pmatrix}+a_2\begin{pmatrix}-1\\0\\1\\0\end{pmatrix}+a_3\begin{pmatrix}-1\\1\\0\\0\end{pmatrix}.$$

• **Умножение матриц.** В отличие от покомпонентного умножения `*` или, в полной форме `Times`, обычное матричное умножение представляющее композицию линейных отображений, обозначается `.` или, в полной форме `Dot`. Таким образом, `x.y` или `Dot[x,y]` выражает произведение матриц $x$ и $y$. Например,

```
In[202]:= {{a,b},{c,d}}.{{e,f},{g,h}}
Out[202]={{a*e+b*g,a*f+b*h},{c*e+d*g,c*f+d*h}}
```

Чтобы проиллюстрировать матричные вычисления на содержательном примере, свяжем с каждой функцией $f$ **матрицу Тэйлора:**

```
In[203]:= taylor[f_,x_,n_]:=
```

```
Table[If[i<=j,D[f,{x,j-i}]/(j-i)!,0],{i,1,n+1},{j,1,n+1}]
```

Взглянем на матрицу Тэйлора *степени* 5 — с точки зрения анализа, где нумерация производных начинается с нуля, эта матрица является матрицей Тэйлора *порядка* 4:

$$T(f,4)=\begin{pmatrix} f(x) & f'(x) & f''(x)/2 & f'''(x)/6 & f''''(x)/24 \\ 0 & f(x) & f'(x) & f''(x)/2 & f'''(x)/6 \\ 0 & 0 & f(x) & f'(x) & f''(x)/2 \\ 0 & 0 & 0 & f(x) & f'(x) \\ 0 & 0 & 0 & 0 & f(x) \end{pmatrix}$$

Вычислим теперь произведение двух матриц Тэйлора:

In[204]:= `Simplify[taylor[f[x],x,4].taylor[g[x],x,4]]//MatrixForm`

Мы не воспроизводим результат этого вычисления ввиду его громоздкости, но у каждого, кто когда-нибудь видел формулу Лейбница дифференцирования произведения, в этот момент закрадываются сильные подозрения. После следующего вычисления

```
In[205]:= Timing[Simplify[ taylor[f[x],x,50].taylor[g[x],x,50]]==
              taylor[f[x]*g[x],x,50]]
Out[205]={3.034,True}
```

эти подозрения незамедлительно переходят в уверенность. В самом деле, две матрицы степени 51 не могут совпасть по случайной причине! Это значит, что от нас в курсе анализа скрывали нечто весьма существенное, а именно то, что многочлен Тэйлора $n$-го порядка произведения двух функций равен произведению их многочленов Тэйлора того же порядка:

$$T(fg,n)=T(g,n)T(g,n).$$

Разумеется, многочлен Тэйлора $n$-го порядка следует здесь понимать *с точностью до бесконечно малых более высокого порядка* или, как сказал бы алгебраист, **по модулю** $x^{n+1}$**.**

То, что мы сейчас увидели — это как раз типичный пример *грамотного* использования `Mathematica`. А именно, опытный пользователь спрашивает у системы то, что он действительно хочет узнать, в данном случае, равна ли матрица Тэйлора функции $fg$ *произведению* матриц Тэйлора функций $f$ и $g$. Но для этого совершенно не обязательно фактически смотреть на сами эти матрицы или их произведение!! Неумелое использование, которым грешат не только начинающие, но и многие авторы учебных текстов, состоит в том, чтобы показывать то, что с точки зрения окончательной цели является промежуточным результатом. Например, выводить на экран матрицу `taylor[f[x],x,50].taylor[g[x],x,50]`.

• **Обращение матриц.** Обратная к $g$ матрица $g^{-1}$ вычисляется при помощи функции `Inverse`. Прежде всего проверим, что мы правильно понимаем, в каком формате вызывается `Inverse`:

```
In[206]:= Inverse[{{a,b},{c,d}}]
Out[206]={{d/(-b*c+a*d),-b/(-b*c+a*d)},
                                    {-c/(-b*c+a*d),a/(-b*c+a*d)}}
```

Проиллюстрируем теперь использование этой функции на содержательном примере. Следующая трехдиагональная матрица является одной из самых знаменитых и важных в математике. Чистым математикам она известна как **матрица Картана**[45], а прикладным — как **матрица конечных разностей**[46]:

```
In[207]:= cartan[n_]:=Table[Which[i==j,2,Abs[i-j]==1,-1,True,0],
                                    {i,1,n},{j,1,n}]
```

Обратите внимание на использование для задания этой матрицы **условного оператора** `Which`. Встречавшийся нам ранее оператор `If[test,x,y,z]`, выбирает одну из трех возможностей $x, y, z$ в зависимости от трех значений истинности теста `test`, в следующем порядке: `True`, `False`, `Undecided`. В отличие от него оператор `Which` вызывается в формате

```
Which[test1,x1,test2,x2,test3,x3,...]
```

Этот оператор *поочередно* оценивает истинность каждого из тестов `test1`, `test2`, `test3`, ... и возвращает `xi` для *первого* из них, который принимает значение `True`. В нашем примере последний тест *всегда* выдает значение `True`, так что коэффицент матрицы в позиции $(i, j)$ равен 0, если пара $(i, j)$ не проходит ни одного из предыдущих тестов. Примерно так же работает и **переключатель** `Switch`, но, в отличие от условного оператора `Which`, он вызывается в другом формате и проверяет не прохождение теста, а совпадение выражения с одной из предписанных форм или его соответствие предписанному паттерну. Так, скажем,

```
Switch[Mod[i-j,4],0,a,1,b,2,c,3,d]
```

принимает значение $a, b, c$ или $d$ в соответствии с тем, чему равен вычет $i - j$ по модулю 4.

Посмотрим для примера на матрицу `cartan[8]`:

$$\begin{pmatrix} -2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 2 \end{pmatrix}$$

[45] Н.Бурбаки,

[46] Р.Грегори, Е.Кришнамурти, Безошибочные вычисления. Методы и приложения. — М., Мир, 1988, с.1–208.

Обратная к этой матрице хорошо известна и очень часто используется. Изобразим для примера `9*Inverse[cartan[8]]`:

$$\begin{pmatrix} 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 \\ 7 & 14 & 12 & 10 & 8 & 6 & 4 & 2 \\ 6 & 12 & 18 & 15 & 12 & 9 & 6 & 3 \\ 5 & 10 & 15 & 20 & 16 & 12 & 8 & 4 \\ 4 & 8 & 12 & 16 & 20 & 15 & 10 & 5 \\ 3 & 6 & 9 & 12 & 15 & 18 & 12 & 6 \\ 2 & 4 & 6 & 8 & 10 & 12 & 14 & 7 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{pmatrix}$$

• **Определитель матрицы.** Определитель матрицы $g$ вычисляется при помощи функции `Det`. Как всегда, прежде всего стоит проверить на совсем простом примере, правильно ли мы понимаем ее использование:

```
In[208]:= Det[{{a,b},{c,d}}]
Out[208]= -b*c+a*d
```

Обратимся теперь к более интересным примерам. Мы уже говорили, что теория определителей была придумана для анализа *единственного* примера — определителя Вандермонда $\det(V(x_1,\dots,x_n))$. Так вот с него и начнем. Вычисление `Length[Det[vandermonde[y,5]]]` показывает, что `Mathematica` считает, что этот определитель является суммой 120 слагаемых — как в этом мире все тяжело, все горести полно!!![47] Однако применяя к тому же определителю команду `Simplify`, мы получим уже вполне осмысленный результат:

```
In[209]:= Simplify[Det[vandermonde[y,5]]]
Out[209]= (y[1]-y[2])*(y[1]-y[3])*(y[2]-y[3])*(y[1]-y[4])*(y[2]-y[4])*
          (y[3]-y[4])*(y[1]-y[5])*(y[2]-y[5])*(y[3]-y[5])*(y[4]-y[5])
```

Команда `Minors[x]` порождает все **дополнительные миноры** матрицы $x$. Однако упорядочивает она их лексикографически по порядку *включенных* в них строк и столбцов (а вовсе не исключенных, как это делается в элементарных учебниках линейной алгебры!) Следующее вычисление показывает дополнительные миноры в матрице степени 3:

```
In[210]:= fancy1=Partition[CharacterRange["a","i"],3]
Out[210]={{a,b,c},{d,e,f},{g,h,i}}
In[211]:= Minors[fancy1]
Out[211]={{-b*d+a*e,-c*d+a*f,-c*e+b*f},
          {-b*g+a*h,-c*g+a*i,-c*h+b*i},
          {-e*g+d*h,-f*g+d*i,-f*h+e*i}}
```

Обратите внимание на то, как мы порождаем матрицу степени 3:

[47] "Deus," dist li reis, "si penuse est ma vie!!!" — La chanson de Roland.

◦ Команда `CharacterRange["a","i"]` генерирует список из 9 букв, лежащих между `a` и `i`, включительно. Отметим необходимость использования здесь кавычек! Эти кавычки означают, что `a` и `i` должны трактоваться не как символы, а как **стринг**, т.е. *текстовые объекты*, воспринимаемые `verbatim`. Большинство обычных команд трактует стринг как **сырой объект** и не проводит с ним никаких вычислений. Имеются специальные текстовые команды, которые позволяют проводить со стрингами все обычные манипуляции, аналогичные обычным манипуляциям со списками.

◦ Команда `Minors[x,m]` порождает все миноры порядка $m$ матрицы $x$, в лексикографическом порядке.

```
In[212]:= fancy2=Partition[CharacterRange["a","p"],4]
Out[212]={{a,b,c,d},{e,f,g,h},{i,j,k,l},{m,n,o,p}}
```

Теперь вычисление

```
In[213]:= Minors[fancy2,2]
```

дает следующий результат:

$$\begin{pmatrix} -be+af & -ce+ag & -de+ah & -cf+bg & -df+bh & -dg+ch \\ -bi+aj & -ci+ak & -di+al & -cj+bk & -dj+bl & -dk+cl \\ -bm+an & -cm+ao & -dm+ap & -cn+bo & -dn+bp & -do+cp \\ -fi+ej & -gi+ek & -hi+el & -gj+fk & -hj+fl & -hk+gl \\ -fm+en & -gm+eo & -hm+ep & -gn+fo & -hn+fp & -ho+gp \\ -jm+in & -km+io & -lm+ip & -kn+jo & -ln+jp & -lo+kp \end{pmatrix}$$

• **Собственные числа и векторы.** Собственные числа матрицы ищутся при помощи команды `Eigenvalues`, а соотвествующие им собственные векторы — при помощи команды `Eigenvectors`. Их можно найти одновременно при помощи команды `Eigensystem`.

В следующем вычислении мы предлагаем системе вычислить собственные числа и собственные векторы матрицы Фибоначчи $\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$:

```
In[214]:= Eigensystem[{{1,1},{1,0}}]
Out[214]={{1/2*(1+Sqrt[5]),1/2*(1-Sqrt[5])},
                {{1/2*(1+Sqrt[5]),1},{1/2*(1-Sqrt[5]),1}}}
```

Уже нахождение собственных чисел и векторов целочисленных $3 \times 3$-матриц может быть нетривиальной вычислительной проблемой. Одним из интереснейших примеров является матрица кубического золотого сечения[48]

$$\begin{pmatrix} 3 & 2 & 1 \\ 2 & 2 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

Функция `CharacteristicPolynomial` вычисляет характеристический многочлен матрицы:

[48] В.И.Арнольд, Что такое математика? — М., МЦНМО, 2004, с.1–104; с.99.

```
In[215]:= CharacteristicPolynomial[{{3,2,1},{2,2,1},{1,1,1}},x]
Out[215]=1-5*x+6*x^2-x^3
```

Однако, как мы уже видели в § 3, корни этого многочлена выглядят достаточно хитро.

• **Функции от матриц.** Начинающий должен быть особенно осторожен в том, что касается функций, вычисление которых зависит от используемого умножения. Дело в том, что большинство обычных функций по умолчанию используют *покомпонентное* умножение `Times`, называемое в теории матриц умножением по Адамару или умножением по Шуру. В то же время, для правильного вычисления функций от матриц необходимо всюду использовать обычное матричное умножение `Dot`. Классическая теория Кэли—Сильвестра—Фробениуса функций от матриц как раз и основана на том, что для *диагональных* матриц эти два умножения совпадают.

Переводя разговор в практическую плоскость, это значит, например, что для вычисления степени матрицы нужно использовать не функцию `Power`, а функцию `MatrixPower`, для вычисления экспоненты от матрицы — не функцию `Exp`, а функцию `MatrixExp` и т.д.

Вот, например, вычисление экспоненты числа $1+\sqrt{2}$, выраженного целочисленной матрицей $\begin{pmatrix}1&2\\1&1\end{pmatrix}$:

```
In[216]:= MatrixExp[{{1,2},{1,1}}]
Out[216]={{1/2*E^(1-Sqrt[2])*(1+E^(2*Sqrt[2])),
           E^(1-Sqrt[2])*(-1+E^(2*Sqrt[2]))/Sqrt[2]},
           {E^(1-Sqrt[2])*(-1+E^(2*Sqrt[2]))/(2*Sqrt[2]),
           1/2*E^(1-Sqrt[2])*(1+E^(2*Sqrt[2]))}}
```

Интересно, что применение `Simplify` не упрощает это выражение, но вот применение `FullSimplify` дает

```
In[217]:= FullSimplify[MatrixExp[{{1,2},{1,1}}]]
Out[217]={{E*Cosh[Sqrt[2]],Sqrt[2]*E*Sinh[Sqrt[2]]}
           {E*Sinh[Sqrt[2]]/Sqrt[2],E*Cosh[Sqrt[2]]}
```

• **Приближенные вычисления.** Все, что сказано выше об опасности приближенных вычислений, в еще большей степени относится к вычислениям с матрицами. Дело в том, что большинство алгоритмов используемых для этих вычислений по самой природе итеративны, что приводит к возникновению артефактов — а часто и просто чудовищных ошибок — уже для матриц небольшого порядка. В том же, что касается матриц порядка несколько тысяч или несколько десятков тысяч, приближенные вычисления без надежного контроля точности и устойчивости вычислений являются сплошным обманом зрения.

В качестве подтверждения этого процитируем знаменитый пример Форсайта

```
forsythe=forwaa[10]+10^(-10)*e[10,10,1]
```

Матрица Форсайта

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 10^{-10} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

отличается от жордановой клетки $J_{10}(0)$ лишь в одной позиции, при этом значение элемента в этой клетке довольно мало, $10^{-10}$. В то же время собственные числа этой матрицы по модулю равны $10^{-1}$ и, таким образом, уже весьма заметно ненулевые! Жорданова форма матрицы `forsythe` имеет вид

$$\mathrm{diag}(e_1/10, \ldots, e_{10}/10),$$

где $e_1, \ldots, e_{10}$ — корни 10-й степени из 1. Невооруженным глазом видно, насколько эта жорданова форма отличается от жордановой матрицы $J_{10}(0)$. Этот пример служит замечательной иллюстрацией вычислительной неустойчивости канонических форм. Еще эффектнее выглядит вычисление обратной матрицы `Inverse[forsythe]`:

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 10000000000 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Ясно, что при рассмотрении матриц степени 1000 подобных эффектов можно достичь изменением одного элемента на $10^{-1000}$.

Вычислительная линейная алгебра уже более столетия развивает специальные приемы для борьбы с такого рода явлениями: масштабирование, изучение обусловленности, контроль ошибок. Однако для большинства практически встречающихся случаев самым простым и надежным лекарством является полный отказ от приближенных вычислений.

> Слуха нашего достигло, о чем мы и выговорить не можем без стыда, что ты растолковываешь грамматику кое-кому из своих друзей.
>
> Папа Григорий Великий

Описываемая в нашем учебнике система `Mathematica` является системой компьютерной алгебры общего назначения, при помощи которой можно решать **любой** тип задач, в которых в той или иной форме встречается математика. При этом `Mathematica` наряду с `Maple` является **единственной** такой `high-end`[49] системой, которая настолько проста в использовании, что доступна школьникам и студентам младших курсов.

`Mathematica`, как и любая система компьютерной алгебры, может излагаться с точки зрения

- математика,
- программиста,
- пользователя.

Пользователь интересуется этой системой не с точки зрения принципов ее работы, а как законченным программным продуктом для проведения практических вычислений в конкретной предметной области, скажем, в математике, физике, информатике, технических науках, химии, астрономии, геологии, экономике, управлении, проектировании, архитектуре, лингвистике, компьютерной графике, музыке и т.д. При этом он совершенно не обязан понимать ни математических основ работы системы, ни используемых в ней алгоритмов, ни собственно программистских аспектов ее реализации, ее взаимодействия с платформой, операционной системой и другими программами и других подобных вещей. Тем не менее, любой *серьезный* пользователь, который хочет использовать эти системы не просто в качестве редактора формул и большого научного калькулятора, любой *грамотный* пользователь, который хочет избежать хотя бы наиболее очевидных ошибок и эффективно использовать возможности этих систем, должен понимать хотя бы основные принципы компьютерной алгебры, взаимоотношение математической и вычислительной точек зрения и, в первую очередь, принятые в этих системах стандартные соглашения и специфику используемого ими **языка**.

Модуль 2 рассчитан как раз на такого **серьезного** пользователя, которому и адресованы главы, посвященные основам синтаксиса языка `Mathematica`. Подобно другим системам компьютерной алгебры общего назначения `Mathematica` является *в первую очередь* языком программирования сверхвысокого уровня. Отличие этих систем, в первую очередь

[49] Словарь дает следующие переводы компьютерного термина `high-end`: мощный, профессиональный, высококачественный; высокого класса; с широкими функциональными возможностями. Поскольку ни один из этих переводов не отражает всего пафоса и всех коннотаций оригинала, мы оставляем этот термин `as is`.

`Axiom`, `Mathematica` и `Maple` от традиционных языков программирования таких, как `Fortran`, `Lisp`, `C`, `Pascal` или `Java` состоит в том, что они содержат в *десятки* раз больше командных слов и конструкций и по своей гибкости, выразительной силе и расширяемости приближаются к живым языкам. Поэтому изучать их надо так же, как учат иностранный язык, скажем, французский или греческий.

## Глава 4. ОБЪЕКТЫ И ВЫРАЖЕНИЯ

> Лейбниц говорил, что все бедствия людей от неумения ясно выражаться.
>
> Венедикт Ерофеев, Из записных книжек

Для того, чтобы **вычислить** (`evaluate`) какое-то выражение, `Mathematica` должна быть в состоянии каким-то образом его **интерпретировать**. Всегда ли эта интерпретация совпадает с тем, что имел в виду автор, когда писал программу — это большой отдельный вопрос, к которому мы будем снова и снова возвращаться в следующих главах! Однако чтобы подлежать эвалюации выражение должно быть *как минимум* синтаксически корректным, т.е. **полным** и **правильно составленным**. В этой части мы как раз и опишем некоторые простейшие правила построения корректных выражений и основные способы внутреннего представления данных в самой системе. При этом вводятся важнейшие понятия языка `Mathematica` такие, как *тип объекта, имя объекта, домен, символ, число, переменная, значение, присваивание, подстановка* и т.д., а также некоторые наиболее употребительные команды. В следующих главах мы определим дальнейшие ключевые понятия: *функция, аргумент, параметр, опция, атрибут, оператор, предикат, отношение, паттерн, список, последовательность, стринг* и т.д. Однако даже из этих *наиболее фундаментальных* понятий, *простейших* правил и *важнейших* команд для использования `Mathematica` качестве калькулятора необходима лишь небольшая часть. Поэтому тот, кто интересуется системой `Mathematica` главным образом для решения школьных или вузовских задач по математике, может при первом чтении пропускать **все** непонятные или более технические места, возвращаясь к ним потом по мере необходимости!!! Самое главное — двигаться вперед, потому что математика становится понятной только при взгляде назад: `Mathematica` is only learned in hindsight!!!

## § 1. Выражения: `FullForm`, `Head`, `Part`, `Level`, `Length`, `Depth`

> Everything is an expression.
>
> Steven Wolfram

Прежде, чем начинается вычисление, любой ввод, который Вы печатаете, должен быть интерпретирован ядром системы `Mathematica` как **выражение** (`expression`). Каждое выражение имеет жесткую иерархическую структуру, описывающую из каких объектов оно состоит, как и *в каком порядке* они соединяются в **подвыражения** (`subexpressions`) и т.д. Прежде всего, каждое выражение $x$ имеет **заголовок** `Head[x]`, указывающий, к какому классу оно относится, с чего начинается его вычисление и какие алгоритмы при этом применяются. Заголовок выражения может быть

- **функцией**: `Exp`, `Log`, `Cos`, `Sin` и т.д.,
- **командой**: `Expand`, `Factor`, `Plot`, `Play` и т.д.,
- **оператором**: `Plus`, `Times`, `Power`, `Equal` и т.д.,
- **типом объекта**: `Integer`, `Rational`, `Real`, `Complex`, `Symbol`, `String`, `List`, `SparseArray`, `Graphics`, `SurfaceGraphics`, `Graphics3D`, `Sound` и т.д.

Впрочем, приведенные различия являются психологическими и лингвистическими, а не вычислительными. Они относятся скорее к *интерпретации* проделываемых операций, чем к их внутреннему представлению или тому, как система работает с ними. Например, термин *функция* подразумевает, что проделывается нечто с рассматриваемыми математическими *объектами*, в то время как *команда* меняет не сами эти объекты, а форму их *записи* или, в случае если эти объекты сами являются функциями или командами, *режим* (`modality`) их работы или, наконец, *состояние* самой системы. Удивительная особенность системы `Mathematica`, которая вначале кажется странной, но после приобретения некоторого опыта становится *в высшей степени* удобной, состоит в том, что с ее точки зрения тип `List` объекта `List[x_1,...,x_n]` **ничем** не отличается от функции, объединяющей $n$ аргументов $x_1, \dots, x_n$ в список $\{x_1, \dots, x_n\}$, или команды, *предписывающей* это сделать. А различие между функцией и оператором вообще относится *только* к традиционной форме записи и не имеет никакого судьбоносного значения уже в самой математике. Поэтому серьезные пользователи даже *не пытаются* задумываться над тем, является ли встроенный объект `Blabla` *функцией* или *командой* и используют эти слова как синонимы.

В качестве **атомарных объектов**, не имеющих дальнейшей внутренней структуры, `Mathematica` рассматривает

- **число** (целое, рациональное, вещественное или комплексное),
- **символ** (объект вычислений, определяемый своим **именем**),
- **стринг** (текстовый объект, воспринимаемый `verbatim`).

Атомарный объект тоже имеет заголовок, выражающий его **тип**, но не имеет никаких дальнейших частей. Аналогией этой ситуации является **теория**

**множеств с праэлементами**, где каждый объект является либо **множеством** имеющим элементы (= выражение, имеющее части), либо **праэлементом**, не имеющим элементов (= выражение, не имеющее частей). Перечислим заголовки атомарных типов

| | |
|---|---|
| `Integer` | целое число |
| `Rational` | рациональное число |
| `Real` | вещественное число |
| `Complex` | комплексное число |
| `Symbol` | символ |
| `String` | текстовая строка |

Как мы уже упоминали, в ядре `Mathematica` определено много других типов объектов и данных — `List`, `SparseArray`, `Graphics`, `SurfaceGraphics`, `Graphics3D`, `Sound` и т.д., но они имеют дальнейшую *внутреннюю структуру*, (во многих случаях чрезвычайно сложную!!!) Еще больше таких типов введено в пакетах расширения. Кроме того, опытный пользователь может вводить новые типы объектов и правила, по которым производятся операции над ними.

Таким образом, *каждая* часть выражения, в свою очередь имеет заголовок, а те части, которые *не являются* атомарными объектами, *кроме того*, сами имеют части, являющиеся **составляющими** (`constituents`) исходного выражения. Однако, *как и в теории множеств*, часть части не обязательно является частью исходного выражения (= отношение $\in$ нетранзитивно). Части частей исходного выражения вложены в него **на уровне 2** или **на глубине 2**. Нюанс здесь состоит в том, что один и тот же объект может входить в выражение на разных уровнях, например, $x$ входит в множество $\{x, \{x\}\}$ как на уровне 1 (как элемент), так и на уровне 2 (как элемент элемента)! Кроме того, *в отличие* от теории множеств теперь чрезвычайно важно, **сколько раз** и **в каких позициях** каждая составляющая входит в выражение.

В свою очередь части частей тоже являются либо атомарными объектами, либо имеют дальнейшую структуру. В последнем случае про их части говорят, что они вложены в исходное выражение **на уровне 3** или **на глубине 3**. В силу конечности процесс взятия частей должен оборваться, а оборваться он может только на атомарных объектах, это те составляющие, на которых невозможно дальнейшее движение вглубь, они называются **листьями** (`leaves`) — в мире `Mathematica` деревья растут сверху вниз!! Лист является ЧАСТЬЮ ЧАСТИ ... ЧАСТИ и глубина его вхождения определяется тем, сколько раз здесь повторено слово *часть*.

Перечислим несколько команд, позволяющих увидеть, как *на самом деле* устроено выражение, каковы его заголовок, части, уровни и полная форма.

**Философский комментарий.** Выражение *на самом деле* здесь относится к тому уровню сознания, на котором функционирует *язык* системы `Mathematica`. Разумеется *на самом деле* внутреннее представление данных в коде языка `C` и фактическое представление данных в памяти компьютера устроены как-то иначе и, в любом случае, еще

**гораздо** более детально. Но система Mathematica только потому и обладает интеллектом, достаточным для проведения весьма сложных вычислений в режиме диалога, а не процедуры, что *на сознательном уровне* она не имеет доступа к этим более низким уровням мышления. Здесь мы обсуждаем представление данных на *внутреннем* языке Mathematica, не зависящее от платформы и программной реализации, иными словами, **внутренние**, а не **системные** феномены.

Итак, вот основные команды, позволяющие анализировать структуру выражений, в простейшем виде. В дальнейшем мы обсудим дальнейшие спецификации параметров в этих командах, которые позволят видеть больше и/или иначе.

| | |
|---|---|
| `FullForm[x]` | полная форма выражения $x$ |
| `TreeForm[x]` | иерархическая структура выражения $x$ |
| `Head[x]` | заголовок выражения $x$ |
| `Part[x,n]` | $n$-я часть выражения $x$ |
| `Length[x]` | длина выражения $x$, *не считая* заголовка |
| `Depth[x]` | глубина выражения $x$, *считая* заголовок |
| `Level[x,{n}]` | составляющие выражения $x$ на уровне $n$ |
| `Level[x,n]` | составляющие выражения $x$ на уровнях $\leq n$ |
| `AtomQ[x]` | вопрос, является ли $x$ атомарным объектом |

Опишем, что делают эти команды. Использование функций `Head[x]`, `Length[x]` и `Depth[x]` ясно без всяких дальнейших объяснений, нужно только помнить, что по умолчанию (`Heads->False`) команда `Length` *не включает* заголовок, являющийся **нулевой частью** выражения $x$ в подсчет частей, а вот команда `Depth` *включает* **нулевой уровень** (состоящий из заголовка выражения $x$) в его глубину. Вот что делают остальные команды:

• `FullForm[x]` дает **полную форму** выражения $x$ в функциональной записи, где расшифрованы все сокращения, расставлены все *необходимые* скобки, произведены все подстановки, после чего составляющие отсортированы в **стандартном порядке** (обычно численном и/или алфавитном).

• `TreeForm[x]` — практически то же самое, что `FullForm[x]` но, *кроме того*, выражение явным образом *графически* разделено на уровни, показывающие глубину вхождения составляющих.

• `Part[x,n]` или, сокращенно, `x[[n]]` — часть выражения $x$ с номером $n$. Выражение `Part[x,n]` имеет смысл только для `0<=n<=Length[x]`, если Вы попробуете выделить $n$-ю часть выражения $x$ для `n>Length[x]`, система выдаст сообщение об ошибке. По определению `Part[x,0]===Head[x]`. Мы подробно опишем действие команд `Part[x,m,n]`, `Part[x,{m1,...mr}]`, `Part[x,{m1,...mr},{n1,...,ns}]`, а также альтернативный способ извлечения частей `Extract[x,{i,j}]` в § 2 Главы 8.

• `Level[x,{n}]` порождает список составляющих выражения, лежащих на уровне $n$, а `Level[x,n]` — список всех составляющих на уровнях от 1 до $n$. По определению `Level[x,{0}]==={x}` и `Part[x,0]==={}`. В § 2 Главы 8

мы более обстоятельно опишем детализацию уровней (`level specification`) в этой и других подобных командах.

## § 2. Что такое квадратный трехчлен?

В качестве простейшего примера изложенной в предыдущем параграфе теории проанализируем внутреннее представление квадратного трехчлена $f = ax^2 + bx + c$. Вот что *на самом деле* ядро программы `Mathematica` понимает под квадратным трехчленом и вот с чем происходит вычисление:

```
In[1]:=f=a*x^2+b*x+c; FullForm[f]
Out[1]=Plus[c,Times[b,x],Times[a,Power[x,2]]]
```

Посмотрим вначале на ввод: в нем встречаются три специальных символа $+$, $*$ и $\hat{}$, используемых при **сокращенной инфиксной записи** алгебраических операций сложения, умножения и возведения в степень, см. по этому поводу § 6 Главы 6. В палитре `BasicCalculations` можно вводить значки всех этих операций в **традиционной форме**, наподобие того, как это делается в `MathCad`, как знак умножения и верхний индекс, однако с нашей точки зрения это абсолютно бессмысленно, так как ввод с клавиатуры значительно быстрее, чем шуршание мышкой по палитрам. Посмотрим теперь вывод: `Mathematica` считает, что мы предложили ей сложить три вещи: атомарный объект $c$, который не имеет никакой дальнейшем структуры (на момент вычисления является *символом*), произведение атомарного объекта $b$ на атомарный объект $x$ и еще одну вещь, которая тоже устроена как произведение. Однако это второе произведение имеет *дальнейшую структуру*, так как только первый множитель является атомарным объектом, в то время как второй в свою очередь является результатом возведения атомарного объекта $x$ в степень 2.

С другой стороны, команда `TreeForm` порождает вывод наподобие следующего, явным образом показывающий, *на каком уровне* вложены подвыражения:

```
In[2]:=TreeForm[f]
Out[2]=Plus[c   |           , |                   ]
               Times[b,x]   Times[a, |         ]
                                   Power[x,2]
```

Обратите внимание, что мы не повторяли определение $f$. Если это происходит в той же сессии, в которой мы спрашивали про полную форму $f$, мы можем не вводить определение $f$ второй раз, так как система его помнит. Для опытного пользователя, умеющего мгновенно не включая сознание считать скобки до седьмого уровня, эта форма представления выражения не содержит ничего нового по сравнению с полной формой. Однако начинающий должен отметить для себя несколько моментов, например, то, что *на каждом уровне* закрывается такое же количество скобок, сколько открывается. В действительности, это является одной из важнейших характеристик *правильно составленного* выражения. В частности, мы видим, что внутреннее

представление многочлена $ax^2+bx+c$ состоит из **трех** уровней — плюс, конечно, заголовок `Plus`: спросите у системы, верно ли, что `Head[f]===Plus`? Таким образом, общая глубина этого выражения, *включая заголовок*, равна 4, `Depth[f]===4`. Посмотрим теперь еще раз на части представляющего $f$ выражения:

```
In[3]:=Map[Part[f,#]&,Range[0,3]]
Out[3]={Plus,c,b*x,a*x^2}
```

Мы видим, что это выражение состоит из заголовка `Plus` и **трех** частей `c`, `b*x` и `a*x^2`. Это значит, что `Length[f]===3`.

В предыдущем диалоге мы использовали несколько команд, а именно, `Map`, `Function` (в сокращенной записи, использующей специальные знаки # и &) и `Range`, которые нам до сих пор не встречались. Эти команды подробно описаны в Главе 6, но в действительности они использованы здесь только для сокращения. Эта строка предлагает программе подставить в выражение `Part[f,i]` вместо `i` поочередно все четыре элемента множества `Range[0,3]===`{`0,1,2,3`}. Мы могли бы с таким же успехом напечатать более наивный текст, скажем,

```
In[4]:={Part[f,0],Part[f,1],Part[f,2],Part[f,3]}
```

и получили бы точно такой же результат. Фигурные скобки здесь нужны для того, чтобы система показала нам все четыре результата одновременно. Без этих фигурных скобок она выдала бы нам сообщение о синтаксической ошибке, либо, если бы мы придали этой строке синтаксически правильную форму, заменив все запятые на точки с запятой, отобразила бы на экране только результат последнего вычисления, а не всех четырех вычислений!

А вот как работает команда `Level[f,{x}]`. Посмотрим на уровни многочлена $f$ с нулевого по четвертый:

```
In[4]:=Map[Level[f,{#}]&,Range[0,4]]
Out[4]={{c+b*x+a*x^2},{c,b*x,a*x^2},{b,x,a,x^2},{x,2},{}}
```

Как и в предыдущем случае, мы могли бы породить тот же ответ без всяких там `Map` и `Function`, при помощи следующего незамысловатого текста:

```
In[5]:={Level[f,{0}],Level[f,{1}],Level[f,{2}],
                        Level[f,{3}],Level[f,{4}]}
```

А вот как действует `Level[f,n]` с указанием уровня **без** фигурных скобок, при этом каждый следующий список включает все предыдущие:

```
In[5]:=Map[Level[f,#]&,Range[0,4]]
Out[5]={{},{c,b*x,a*x^2},{c,b,x,b*x,a,x^2,a*x^2},
                                {c,b,x,b*x,a,x,2,x^2,a*x^2}
```

**Упражнение.** Проделайте то же самое для многочленов степени 4 или 5, и посмотрите на их реализацию как выражений на внутреннем языке системы `Mathematica`, их части и уровни.

Кстати, это упражнение будет полезно начинающему и как практика во вводе синтаксически правильного текста с клавиатуры!

Переведем этот ответ на более привычный язык. Разумеется, то же самое относится не только к многочленам степени 2, но и вообще ко всем многочленам $f = a_n x^n + \ldots + a_1 x + a_0$.

∘ на нулевом уровне лежит сам многочлен $f$;

∘ на первом уровне — составляющие его одночлены $a_i x^i$, включая свободный член $a_0$;

∘ на втором уровне — коэффициенты $a_i$ — кроме свободного члена, уже посчитанного на первом уровне! — и *нормированные* одночлены, кроме 1, которая автоматически втягивается в свободный член;

∘ на третьем уровне — переменная $x$ и показатели степени $\geq 2$, с которыми она входит в нормированные одночлены (`Power[x,0]` даже если мы задавали бы многочлен в такой форме, автоматически упрощается до 1, а `Power[x,1]` — до `x`, так что они до этого уровня не доживают);

∘ на четвертом уровне (и, тем самым, на более глубоких уровнях!) нет вообще ничего, что, впрочем, не удивительно, так как глубина $f$ равна 4, *считая* нулевой уровень.

Это показывает, насколько язык системы `Mathematica` близок к структурному математическому мышлению! В качестве составляющих многочлена естественно возникли одночлены, нормированные одночлены, коэффициенты, неизвестная и показатели степени, то есть как раз все те понятия, которые используются при *математическом* описании многочленов! Эта ситуация в высшей степени типична!! Нет ничего более полезного для эффективной организации вычислений в любой области, чем понимание математической структуры рассматриваемых понятий, а язык системы `Mathematica` абсолютно сознательно структурирован в том духе, как это принято в современной математике.

## § 3. Выделение уровней

> Ты, Петька, прежде чем о простых вещах говорить, разберись со сложными.
>
> Виктор Пелевин. Чапаев и Пустота

В § 1 мы уже упоминали о различных способах выделения уровня. Сейчас мы расскажем всю правду.

| | |
|---|---|
| `Level[x,levelspec]` | уровни выражения $x$, описываемые `levelspec` |
| `n` | верхние $n$ уровней выражения |
| `-n` | нижние $n$ уровней выражения |
| `{n}` | только $n$-й уровень выражения |
| `{-n}` | только $n$-й уровень выражения снизу |
| `{m,n}` | уровни выражения между $m$ и $n$ |
| `Infinity` | все уровни выражения |

Использование положительных спецификаций уровня уже обсуждалось выше и понятно само по себе. В качестве еще одного примера рассмотрим вложенный список (`nested list`):

```
alist={a,{b,{c,{d,{e,{f}}}}}}.
```

Вот уровни этого выражения с 1-го по 7-й

```
In[6]:=Map[Level[alist,{#}]&,Range[1,7]]
Out[6]={{{a,{b,{c,{d,{e,{f}}}}}}, {b,{c,{d,{e,{f}}}}},
                {c,{d,{e,{f}}}}, {d,{e,{f}}}, {e,{f}}, {f}, {}}
```

Чуть труднее понять использование отрицательных уровней. Отрицательная спецификация уровня `-n` выделяет все составляющие выражения, которые имеют глубину $n$. Напомним, что по умолчанию заголовок выражения включается в его глубину, так что глубина атома равна 1. Кроме того, во избежание недоразумений еще раз подчеркнем, что речь идет НЕ О ТЕХ СОСТАВЛЯЮЩИХ, КОТОРЫЕ ЛЕЖАТ НА ГЛУБИНЕ $n$ В ИСХОДНОМ ВЫРАЖЕНИИ, А О ТЕХ, КОТОРЫЕ САМИ ИМЕЮТ ГЛУБИНУ $n$!!!

Таким образом, например, `Level[x,{-1}]` дает список всех листьев выражения $x$, т.е. всех входящих в $x$ атомов. Точно так же `Level[x,{-2}]` даст список всех составляющих глубины 2, т.е. тех составляющих, которые сами не являются атомами, но вот всех их части уже являются атомами, и т.д. Вот уровни `alist` с $-1$-го по $-7$-й:

```
In[7]:=Map[Level[alist,{-#}]&,Range[1,7]]
Out[7]={{a,b,c,d,e,f}, {{f}}, {{e,{f}}}, {{d,{e,{f}}}},
                {{c,{d,{e,{f}}}}}, {{b,{c,{d,{e,{f}}}}}},
                {{a,{b,{c,{d,{e,{f}}}}}}}}
```

По умолчанию уровни не включают заголовки выражений, иначе говоря, дефолтной установкой опции `Heads` является `False`. Однако эту установку можно перебороть, установив `Heads->True`:

```
In[8]:=Level[alist,{-1},Heads->True]
Out[8]={List,a,List,b,List,c,List,d,List,e,List,f}
```

Спецификация уровня `{m,n}` включает в себя все уровни выражения между $m$-м и $n$-м, **включительно**. Так, например, `Level[x,{2,-2}]` описывает все составляющие, кроме частей и листьев выражения $x$:

```
In[9]:=Level[alist,{2,-2}]
Out[9]={{f}, {e,{f}}, {d,{e,{f}}}, {c,{d,{e,{f}}}}}
```

Кроме функции `Level` многие другие важнейшие функции, в особенности функции работы со списками, функционального программирования и некоторые отношения, используют спецификацию уровня. К этим функциям относится, *в частности*, `Apply, Cases, Count, FreeQ, Map, MapIndexed, MapThread, Position, Replace, Scan`. Для того, чтобы грамотно использовать эти и другие подобные им функции, исключительно важно помнить, что по умолчанию они используют различные спецификации уровня!!!

Рассмотрим в качестве примера две важнейшие команды применения функции к спискам, `Apply` и `Map`. Каждая из них может применяться к выражению на любом уровне. В качестве примера применим к `alist` функцию $g$ вначале при помощи команды `Apply` на уровнях с 1-го по 6-й, а потом при помощи команды `Map` на уровнях с 1-го по 7-й:

```
In[10]:=Map[Apply[g,alist,{#}]&,Range[0,6]]

Out[10]={g[a,{b,{c,{d,{e,{f}}}}}], {a,g[b,{c,{d,{e,{f}}}}]},
          {a,{b,g[c,{d,{e,{f}}}]}},{a,{b,{c,g[d,{e,{f}}]}}},
                {a,{b,{c,{d,g[e,{f}]}}}}, {a,{b,{c,{d,{e,g[f]}}}}},
                      {a,{b,{c,{d,{e,{f}}}}}}

In[11]:=Map[Map[g,alist,{#}]&,Range[0,7]]

Out[11]={g[{a,{b,{c,{d,{e,{f}}}}}}], {g[a],g[{b,{c,{d,{e,{f}}}}}]},
          {a,{g[b],g[{c,{d,{e,{f}}}}]}}, {a,{b,{g[c],g[{d,{e,{f}}}]}}},
                {a,{b,{c,{g[d],g[{e,{f}}]}}}},{a,{b,{c,{d,{g[e],g[{f}]}}}}},
                      {a,{b,{c,{d,{e,{g[f]}}}}}}, {a,{b,{c,{d,{e,{f}}}}}}
```

Применение $g$ на более глубоких уровнях как при помощи `Apply`, так и при помощи `Map` бессмысленно, так как мы видим, что начиная с уровня 6 для `Apply`, и начиная с уровня 7 для `Map` выражение перестает меняться.

Однако если применить $g$ к `alist` при помощи `Apply` и `Map` **без указания уровня**, то *по умолчанию* `Apply` применяется к выражению в целом, т.е. на уровне 0, а `Map` — к частям выражения, т.е. на уровне 1. Таким образом, вычисление `Apply[g,alist]` даст `g[a,{b,{c,{d,{e,{f}}}}}]`, иными словами, *заголовок* `List` выражения `alist` заменяется на $g$. С другой стороны, вычисление `Map[g,alist]` даст `{g[a],g[{b,{c,{d,{e,{f}}}}}]`, т.е. $g$ применяется к обеим частям выражения `alist`.

## § 4. Имена объектов

Как вы яхту назовете, так она и поплывет.

Христофор Бонифатьевич Врунгель

С точки зрения подавляющего большинства пользователей в `Mathematica` встречаются имена *трех* форматов:

- `Aaaa` — имена **внутренних** (`Internal`) объектов,
- `aaaa` — имена объектов, **определенных пользователем** (`user-defined`),
- `$Aaaa` — имена **системных** (`System`) объектов.

Внутренние и системные объекты являются **встроенными** (`BuiltIn`). При этом внутренние объекты являются фактами *языка*, относящимися к вычислению *как таковому* (как в математических, так и в алгоритмических аспектах), в то время как системные объекты описывают или модифицируют *экстралингвистические* факты: свойства компьютера и операционной системы, на которых *фактически* производится вычисление, взаимодействие ядра `Mathematica` с твердыми и мягкими *изделиями* (`hardware and software`) и тому подобные вещи, внешние по отношению к собственно вычислению. В некоторых случаях нам хочется, чтобы способ выражения влиял на способ действия или, наоборот, возникает искушение включить в нашу прозу непосредственное обращение к действительности (время суток, тип компьютера, широту и долготу и т.д.). Это приводит к появлению

дуплетов типа `MachinePrecision` и `$MachinePrecision`. Впрочем, в большинстве случаев *никакой* прямой связи между внутренними объектами и системными объектами с похожими именами нет: основной графический элемент `Line` не имеет ничего общего со спецификацией ввода `$Line`. На начальном этапе изучения языка следует полностью сосредоточиться на структурных вопросах и игнорировать существование внешнего мира. В настоящей книге мы будем описывать *только* вычисления, которые могут быть реализованы в терминах ВНУТРЕННИХ объектов языка `Mathematica`. Резюмируем основные правила образования имен:

• В отличие от многих языков программирования заглавные (`UpperCase`) и строчные (`LowerCase`) буквы рассматриваются как различные.

• Имя любого ВНУТРЕННЕГО объекта начинается с заглавной буквы. Имя любого СИСТЕМНОГО объекта начинается с доллара $ (`DollarSign`), за которым следует заглавная буква.

• В случае, когда название внутреннего или системного объекта состоит из нескольких слов, КАЖДОЕ из них пишется с заглавной буквы. Никаких пробелов, дефисов и пр. внутри имени не ставится.

В большинстве ситуаций пробел интерпретируется как знак умножения $*$. Таким образом `x y` обозначает **произведение** объектов `x` и `y`, а вовсе не новый объект с именем `xy`. Единственный специальный знак, который начинающий по аналогии с некоторыми другими языками программирования может интерпретировать как часть имени, это _ в составе такой конструкции как blabla_pattern но в действительности этот знак вводит паттерны, т.е. указывает к какому классу принадлежит объект `blabla`

• Имена внутренних и системных объектов являются либо полными английскими словами, либо СТАНДАРТНЫМИ математическими сокращениями, принятыми в англоязычной литературе.

• Индивидуальные объекты, названные в честь своего изобретателя, *как правило* имеют имя формы `PersonSymbol`. Например, `дельта Кронекера` будет называться `KroneckerDelta`, $n$-е число Каталана — `CatalanNumber[n]`. Однако в некоторых случаях это соглашение не соблюдается. Скажем, константа Каталана называется просто `Catalan`.

• Имя объекта, определенного пользователем, может начинаться с любой строчной буквы и содержать в своем составе ЗАГЛАВНЫЕ И СТРОЧНЫЕ БУКВЫ И ЦИФРЫ.

При желании пользователь может определять и имена, начинающиеся с прописной буквы, но в этом случае нет никакой уверенности в том, что они не вступят в конфликт с какими-то именами объектов, встречающимися в пакетах.

• Ни одно имя НЕ МОЖЕТ НАЧИНАТЬСЯ С ЦИФРЫ. Появляющееся в начале выражения число интерпретируется как коэффициент, а не часть имени. Например, $2x$ представляет удвоенное $x$, а вот $x1, x2, x3$ это одна из форм (далеко не самая удобная!!!) ввести переменные $x_1, x_2, x_3$.

• Имя объекта не может содержать НИКАКИХ СПЕЦИАЛЬНЫХ ЗНАКОВ таких как @, %, ^, &, #, / и т.д. ВСЕ эти знаки используются системой для специальных целей: сокращенной (“операторной”) записи функций, управления и форматирования ввода.

**Вся правда 1.** То, что мы сказали об именах объектов есть правда, но не вся правда. Блоки, модули и некоторые другие специальные команды порождают имена некоторых других форматов, скажем локальные и уникальные имена формата `aaa$nnn`. Однако знакомство с этими форматами предполагает некоторое понимание того, что *реально* происходит при работе программы и может понадобиться Вам *только* в тот момент, когда Вы реально начнете писать программы, исполнение которых требует много часов (или дней!) Любой юзер, работающий в диалоговом режиме, может прекрасно обходиться без этих вещей придумывая уникальные имена для своих переменных.

**Вся правда 2.** С точки зрения профессионала (например, разработчика пакетов) имя объекта в языке `Mathematica` устроено примерно так же, как имя файла. В большинстве операционных систем допустимы файлы с одинаковыми именами, *если* они находятся в различных директориях. Иными словами, `name.ext` называемое в просторечии *именем файла*, является только частью его полного имени, включающего в себя **путь** от корневой директории:

```
dir1\dir2\...\dir17\name.ext
```

Точно так же `Name`, которое до сих пор называлось именем объекта `Mathematica`, в действительности является только частью его **полного имени**. Настоящее имя объекта имеет форму `Context‘Name`, где `Context‘` представляет собой **контекст**. В свою очередь контекст может быть частью другого более широкого контекста и т.д.!!! Тем самым, *в действительности* полное имя объекта `Name` имеет вид

```
Context1‘Context2‘...Context17‘Name
```

Это значит, что даже на протяжении одной сессии можно без всяких конфликтов использовать одно и то же имя для *десятков* различных объектов, если это имя *встречается в различных контекстах*.

Переход в другой контекст осуществляется командой `Begin["Context‘"]`, а возвращение в начальный контекст командой `End[]`. Однако по умолчанию начинающий видит (в действительности, не видит!!!) только два контекста, `System‘`, для встроенных объектов, и `Global‘`, для тех объектов, которые он определяет во время сессии. Использование контекстов становится *по-настоящему* важным только в тот момент, когда Вы либо профессионально занимаетесь написанием пакетов, которыми пользуются тысячи юзеров, либо решаете задачи в многопроцессорном режиме, для чего Вам приходится организовывать трубки для обмена данными между несколькими процессами, скажем, между несколькими работающими одновременно сессиями `Mathematica`. Однако и в том и в другом случае Вы вряд ли читаете эту книгу. Во всех остальных случаях Вам не представит труда придумать несколько сотен уникальных имен для определяемых Вами объектов. Даже если среди этих имен есть повторяющиеся, обычно легко избежать их использования на протяжении одной сессии.

## § 5. Группировка и скобки в математике

> In the force if Yoda's so strong, construct a sentence with words in the proper order then why can't he?
>
> Darth Vader

**1. Группировка.** В математике используется много видов скобок, двойных скобок, и т.д., которые следует тщательно различать между собой, в частности

- фигурные скобки $\{,\}$ (`braces`, `curly brackets`),
- круглые скобки $(,)$ (`parenthesis`),

- квадратные скобки $[,]$ (`brackets`),
- ломаные скобки $\langle,\rangle$ (`angle brackets`),

и несколько других видов скобок (`brackets`, `delimiters`). Эти скобки используются в нескольких совершенно различных смыслах, определяемых из контекста.

Круглые скобки используются в первую очередь для **группировки**, т.е. для обозначения последовательности выполнения алгебраических операций: $(x+y)+z = x+(y+z)$. В отличие от школьной алгебры, профессиональные математики **никогда** не используют для группировки квадратные и фигурные скобки. Вместо этого ля лучшей читаемости формул используются круглые скобки разных кеглей: $(,)$, $\big(,\big)$, $\Big(,\Big)$, $\ldots$.

**2. Множества, наборы, тупели.** Напомним, что в математике через $\{a,b,c\}$ обозначается **множество** с элементами $a,b,c$. Еще одной важнейшей функцией круглых скобок является обозначение **тупелей** (`tuple`, `Tupel`). Тупелем называется упорядоченная $n$-ка с неопределенным $n$. В различных учебных и программистских книгах тупели называются также **списками**, **кортежами**, **массивами**, **векторами** и т.д. Так, например, $(a,b,c)$ обозначает упорядоченную тройку с компонентами $a,b,c$. Напомним, что, в отличие от множеств, порядок и кратность компонент тупеля *существенны*! Так, $(a,b,c) \neq (c,a,b)$ и $(a,a,b) \neq (a,b)$, в то время, как $\{a,b,c\} = \{c,a,b\}$ и $\{a,a,b\} = \{a,b\}$. Кроме того, в алгебре широко используется понятие **набора**, промежуточное между множеством и тупелем: в наборе существенны *кратности* компонент, но не их порядок. Специалисты по комбинаторике обычно называют наборы **мультимножествами**. Наборы обычно обозначаются квадратными скобками, таким образом $[a,a,b] \neq [a,b,b]$, но $[a,a,b] = [b,a,a]$.

**3. Другие функции скобок.** Кроме того, **круглые скобки** используются для выражения многих других понятий. Отметим лишь некоторые простейшие функции скобок. Среди прочего они используются для обозначения

- аргумента/ов отображения: $f(x)$, $f(x,y)$,
- скалярного произведения: $(u,v)$,
- интервала частично упорядоченного множества:

$$(x,y) = \{z \in X \mid x < z < y\},$$

- наибольшего общего делителя $(m,n)$.

Впрочем, последнее обозначение следует признать полностью устаревшим, в настоящее время используется исключительно обозначение $\gcd(m,n)$.

**Квадратные скобки** передают, в частности,

- отрезок частично упорядоченного множества:

$$[x,y] = \{z \in X \mid x \leq z \leq y\}$$

,

- наименьшего общего кратного $[m, n]$
- целую часть числа $[x]$

Последние два обозначения также явдяются полностью устаревшими, так как в современных текстах для наименьшего общего кратного используется обозначение $\mathrm{lcm}(m, n)$, а для целой части — $\lfloor x \rfloor$.

Кроме того, в различных разделах математики скобки используются для выражения *десятков* других понятий.

## § 6. Группировка и скобки в Mathematica

> In general, it can never hurt to include extra parentheses, but it can cause a great deal of trouble if you leave parentheses out, and Mathematica interprets your input in a way you do not expect.
>
> Steven Wolfram

> Only idiots' programs contain more parenthesis than actual code.
>
> Real programmer

В тексте, адресованном человеку, предполагается, что он *до некоторой степени* понимает смысл прочитанного и может различить три разные функции круглых скобок в выражении

$$((f(x), f(y)) + (g(x), g(y))) + (h(x), h(y)).$$

Трудно, однако, ожидать подобную степень понимания от компьютера *в любой ситуации*. Это значит, что мы должны каким-то образом дифференцировать эти функции (эта фраза не была *задумана* как каламбур, как это часто бывает, мы ее вначале написали и только потом прочли). В языке Mathematica тот же текст будет записан следующим образом:

```
({f[x],f[y]}+{g[x],g[y]})+{h[x],h[y]}.
```

К сожалению, стандартная компьютерная клавиатура имеет всего три разных пары скобок: `(,)`, `[,]` и `{,}` и все эти три вида здесь использованы!!! Это значит, что для всех остальных контекстов, когда математическая традиция использует скобки, нам нужно будет изобретать новые обозначения. Итак, вот четыре основных вида скобок в языке Mathematica.

| | | |
|---|---|---|
| `()` | `Parenthesis` | группировка |
| `[]` | `Brackets` | аргумент функций |
| `{}` | `Braces` | формирование списков |
| `[[]]` | `DoubleBrackets` | выделение частей списка |

Резюмируем функцию скобок в языке Mathematica. Использование круглых скобок не отличается от их использования в школьной математике, а использование квадратных и фигурных скобок более детально обсуждается в следующих параграфах.

• **Круглые скобки** `(,)` используются исключительно для группировки, т.е. для обозначения порядка выполнения операций. Уже для *алгебраических* операций порядок их выполнения совершенно не очевиден. Употребление круглых скобок совершенно необходимо в следующих случаях:

◦ операция неассоциативна и для нее не задан способ группировки;

◦ операции имеют одинаковый приоритет (в этом случае по умолчанию производится левая группировка);

◦ не существует общепризнанной иерархии операций и/или применяемая в `Mathematica` иерархия отличается от общепринятой (например, для логических операций).

Так, `(x^y)^z` значит совсем не то же самое, что `x^(y^z)`. По умолчанию формула `x^y^z` истолковывается как `x^(y^z)`. Мораль сказанного состоит в том, чтобы в сомнительных случаях, таких как этот, явным образом ставить скобки, указывающие на порядок выполнения операций. В дальнейшем мы снова и снова возвращаемся к этому основополагающему принципу.

• **Квадратные скобки** `[,]` используются для обозначения аргумента — или аргументов — функции. Например, $\sin(x)$ будет записано как `Sin[x]`. Условный оператор `If x Then y Else z` будет записан как `If[x,y,z]`.

• **Фигурные скобки** `{,}` используются для обозначения **всех видов** списков. Множества, наборы и тупели *обозначаются* в `Mathematica` одинаково!! Точнее, множества и наборы представляются *списками* своих элементов. Однако для работы с ними используются разные функции. Обшее понятие списка в `Mathematica` эквивалентно математическому понятию тупеля. Иными словами, два списка

$$\{\texttt{a[[1]],...,a[[m]]}\} \text{ и } \{\texttt{b[[1]],...,b[[n]]}\}$$

тогда и только тогда считаются равными, когда `m=n` и `a[[i]]=b[[i]]` для всех $i = 1, \ldots, n$. В части 3 мы обсудим функции `Sort` и `Union`, которые превращают тупель в набор или множество.

• **Двойные квадратные скобки** `[[,]]` используются для выделения элементов и частей списка (или выражения, трактуемого как список!!), как сокращенная запись функции `Part`. Для списка `x` через `x[[i]]` обозначается его $i$-я компонента. Через `x[[{i,j,...,k}]]` обозначается список

$$\{\texttt{x[[i]],x[[j]],...,x[[k]]}\},$$

состоящий из компонент списка $x$ в позициях $i, j, \ldots, k$. Запись `x[[i,j]]` используется как сокращенная форма `x[[i]][[j]]` для обозначения $j$-й компоненты $i$-й компоненты списка $x$ — если `x[[i]]` сама является списком или интерпретируется как список.

**Комментарий.** Интересно, что в `Maple` делается совершенно другой выбор в плане использования скобок. А именно, круглые скобки используются там в двух совершенно различных смыслах: для обозначения группировки и для обозначения аргументов

функции. Квадратные скобки используются для формирования списков, а фигурные скобки — для формирования множеств. Это значит, что фактически круглые скобки могут использоваться для группировки *только* в сочетании со знаками операций, опускать которые в `Maple` нельзя. Например, в `Maple` выражение `f(x+y)` истолковывается как `f[x+y]`, а вовсе не как `f*(x+y)`.

Одним из важнейших признаков правильно составленного выражения является его сбалансированность: на каждом уровне выражения количество скобок (левых скобок) равно количеству антискобок (правых скобок) того же типа, причем ни в какой момент количество антискобок не может превосходить количество скобок. Скобки могут рождаться (и уничтожаться!) только парами {скобка,антискобка}. Впрочем `FrontEnd` вряд ли позволит Вам забыть об этом, скобки и антискобки, у которых нет пары, будут выделены другим цветом (по умолчанию красным).

## § 7. Числовые домены

Язык `Mathematica` различает множества и домены, которые отвечают обычным в математике способам задания множества, посредством перечисления всех его элементов и посредством характеристического свойства.

• **Множество** задается *явным списком* своих элементов, в обычных обозначениях $\{x_1, \ldots, x_n\}$. Этот список *per forza* конечен!

• **Домен** задается признаком принадлежности, в обычных (некорректных!!!) обозначениях $\{x \mid P(x)\}$ и, вообще говоря, бесконечен.

Математики уже около 100 лет назад пришли к выводу о **недопустимости** использования обозначения $\{x \mid P(x)\}$. Дело в том, что без каких-то дополнительных предположений относительно свойства $P$ нельзя гарантировать существование множества **вообще всех** элементов $x$, обладающих этим свойством. Такие свойства $P$, для которых это множество все же существует, называются **коллективизирующими**. Для всех остальных свойств запись $\{x \mid P(x)\}$ является чисто словесным артефактом, за которым не стоит **никакой** реальности, т.е. который ВООБЩЕ НИЧЕГО НЕ ОБОЗНАЧАЕТ ни в этом, ни в каком-то другом из миров. Все *так называемые* ‘противоречия’, ‘парадоксы’ или ‘антиномии’ теории множеств основаны на предположении о реальности этих артефактов. Что действительно существует *для любого* свойства $P$, так это множество $\{x \in X \mid P(x)\}$ элементов множества $X$, обладающих свойством $P$ (аксиома Цермело **ZF6**). Однако, конечно, в `Mathematica` домены образуются только для коллективизирующих свойств, и, кроме того, все их элементы принадлежат хотя и бесконечному, но очень небольшому множеству всех выражений.

Вот основные домены, определенные в ядре системы `Mathematica`. Дополнительные домены определены в пакетах и, кроме того, пользователь может, конечно, определять собственные домены.

| | | |
|---|---|---|
| `Booleans` | {`True`,`False`} | значения истинности |
| `Integers` | $\mathbb{Z}$ | целые числа |
| `Primes` | $\mathbb{P}$ | простые числа |
| `Rationals` | $\mathbb{Q}$ | рациональные числа |
| `Algebraics` | $\overline{\mathbb{Q}}$ | алгебраические числа |
| `Reals` | $\mathbb{R}$ | вещественные числа |
| `Complexes` | $\mathbb{C}$ | комплексные числа |

Отличие множеств от доменов становится зримым, когда мы хотим выяснить, принадлежит ли какой-то элемент множеству или домену. Проверка принадлежности элемента *множеству* производится посредством вопроса `MemberQ`. В то же время проверка принадлежности элемента *домену* производится посредством отношения `Element`:

| | |
|---|---|
| `Element[x,domain]` | $x$ принадлежит домену `domain` |
| `MemberQ[list,x]` | $x$ встречается в списке `list` |
| `FreeQ[list,x]` | $x$ не встречается в списке `list` |

Отношение `Element` действует совершенно предсказуемым образом:

```
In[12]:={Element[1,Booleans], Element[1,Integers],
                              Element[1,Primes]}
Out[12]={False, True, False}
In[13]:={Element[Pi,Algebraics], Element[Pi,Reals],
                              Element[Pi,Complexes]}
Out[13]={False, True, True}
```

С другой стороны, вычисление `MemberQ[Integers,1]` и `MemberQ[Reals,Pi]` дает значение `False` — в самом деле с точки зрения системы, как `Integers`, так и `Reals` являются *доменами*, а не множествами и не задаются явным списком своих элементов. Более того, со структурной точки зрения они вообще не имеют частей!!!

Несколько особняком стоит интервальная арифметика. **Интервалом** в `Mathematica` называется то, что большая часть специалистов по анализу называет **отрезком** или **сегментом**, т.е. **замкнутый** интервал

$$[a,b] = \{x \in \mathbb{R} \mid a \le x \le b\}.$$

С одной стороны интервал в общем случае заведомо не является списком. С другой, использование интервалов серьезно отличается от использованиея доменов.

| | |
|---|---|
| `Interval[{min,max}]` | интервал между `min` и `max` |
| `IntervalMemberQ[interval,x]` | $x$ лежит в интервале `interval` |

Обратите внимание на синтаксис команды `Interval`!!! Интервал $[a,b]$ задается в формате функции одного аргумента `Interval[{a,b}]`, а вовсе не

как `Interval[a,b]`. Иными словами, аргумент имеет вид списка {`a,b`}. В действительности, функцию `Interval` можно вызывать с любым количеством аргументов, но в этом случае значение функции интерпретируется как объединение нескольких интервалов.

## § 8. Универсальные инструменты: `Simplify`, `FullSimplify`, `Refine`

Едва ли не самыми полезными командами всей системы `Mathematica` являются команды упрощения такие, как `Simplify`, `FullSimplify`, `Refine` и `Assuming`. Типичными условиями в этих командах как раз и является принадлежность определенным доменам каких-то входящих в рассматриваемые выражения переменных или параметров. В приводимых ниже описаниях этих команд под `assuming` как правило как раз и скрывается такого рода условие — или какое-то другое аналогичное условие, например, $x \geq 0$, $x \leq 0$ и т.д.

| | |
|---|---|
| `Simplify[x,assumption]` | упрощение выражения |
| `FullSimplify[x,assumption]` | полное упрощение выражения |
| `FunctionExpand[x,assumption]` | упрощение аргумента функций |
| `Refine[x,assumption]` | упрощение числовой функции |
| `Assuming[assumption,x]` | вычисление выражения $x$, assuming |

Все эти команды, кроме `Assuming`, могут вызываться и без предположений. В этом случае они используют дефолтные системные предположения `$Assumptions`.

Основное различие между `Simplify` и `FullSimplify` состоит в том времени, которое они готовы потратить на выполнение определенного преобразования. Для команды `Simplify` по умолчанию `Time Constraint->300` (пять минут!!!). В то же время для `FullSimplify` по умолчанию `Time Constraint->Infinity`, кроме того, она использует более широкий список табличных преобразований и не накладывает ограничений на длину промежуточных результатов. Тем не менее, для *числовых* функций команда `Refine` заведомо работает лучше:

```
In[14]:=FullSimplify[Cos[x+n*Pi],Element[n,Integers]]
Out[14]=Cos[n*Pi+x]
In[15]:=Simplify[Cos[x+2*n*Pi],Element[n,Integers]]
Out[15]=Cos[x]
In[16]:=Simplify[Cos[x+(2*n+1)*Pi],Element[n,Integers]]
Out[16]=-Cos[x]
In[17]:=Refine[Cos[x+n*Pi],Element[n,Integers]]
Out[17]=(-1)^nCos[x]
```

А вот еще более замечательный пример, из которого видно, что `Mathematica` умеет использовать малую теорему Ферма:

```
Simplify[Mod[n^p,p], Element[n,Integers]&&Element[p,Primes]]
```

упрощается до `Mod[n,p]`. Поясним, что `&&` в правой части обозначает конъюнкцию, так что условие состоит в том, что $n$ целое, а $p$ простое. Конечно, **без** этих предположений никакого подобного упрощения не происходит.

Команда `Assuming` добавляет предположения к дефолтному списук системных предположений `$Assumptions`. Например, `Simplify[Sqrt[x^2]]` сама по себе не сможет произвести вообще никаких упрощений потому, что относительно $x$ ничего не предполагается. Сейчас мы посмотрим, упрощается ли выражение `Sqrt[x^2]` *в предположениях* $x$, что $x$ вещественно, $x \geq 0$ или $x \leq 0$. Команда `Assuming` удобна, если Вы хотите добавить предположения сразу к нескольким упрощениям или, наоборот, попробовать одно и то же упрощение с разными предположениями. Вот как, примерно, она используется.

```
In[18]:=Map[Assuming[#,Simplify[Sqrt[x^2]]]&,
                          {Element[x,Reals],x>=0,x<=0}]
Out[18]={Abs[x],x,-x}
```

Таким образом, мы видим, что для *вещественных* $x$ выражение `Sqrt[x^2]` упростилось до `Abs[x]`, для *неотрицательных* — до $x$ и, наконец, для *неположительных* — до $-x$, as expected. Впрочем, можно подставлять список условий и непосредственно в команды `Simplify`, `FullSimplify`, `Refine`:

```
In[19]:=Map[Refine[Abs[x]+Abs[1-x],#]&, {x<=0,0<=x<=1,1<=x}]
Out[19]={1-2*x,1,-1+2*x}
```

## § 9. Целые и рациональные числа

God is real, unless declared integer.

Real Programmer

Одной из основ системы `Mathematica` является арифметика бесконечной точности. В главе 3 мы детально обсудим использование арифметические операций, пока же остановимся только на задании целых и рациональных чисел. Целое число задается обычным образом и может быть любой длины. Вот основные функции для работы с цифрами целых чисел.

| | |
|---|---|
| `IntegerDigits[n]` | список цифр числа $n$ |
| `FromDigits[x]` | целое число с цифрами $x_n, \ldots, x_0$ |
| `DigitCount[n]` | подсчет количества цифр $0, 1, \ldots, 9$ в числе $n$ |
| `Total[DigitCount[n]]` | общее количество цифр в числе $n$ |

Обсудим использование этих функций.

• Функция `IntegerDigits[n]` игнорирует знак числа $n$, в то время как `IntegerDigits[0]` равен 0.

• Функция `IntegerDigits` может вызываться с одним, двумя или тремя аргументами. Вычисление `IntegerDigits[n,b]` даст список цифр десятичного числа $n$ в базе $b$.

• Вычисление `IntegerDigits[n,b,l]` представляет собой прокрустово ложе, которое *всегда* выдает список длины $l$. Если количество цифр числа $n$ в базе $b$ меньше $l$, то этот список раздувается до нужной длины нулями слева. Если количество цифр числа $n$ больше $l$, то берутся $l$ младших цифр.

• Функция `FromDigits` является обратной к `IntegerDigits` на множестве натуральных чисел. Таким образом, функция `FromDigits[{xn,...,x0}]` перерабатывает список цифр `{xn,...,x0}` в десятичное число

$$x_n 10^n + \ldots + x_1 10 + x_0.$$

Тем самым, `IntegerDigits[FromDigits[list]]` возвращает `list`. С другой стороны, так как `IntegerDigits` игнорирует знак, то вычисление композиции в обратном порядке `FromDigits[IntegerDigits[n]]` возвращает *абсолютную величину* числа $n$, а не само это число.

• Функция `FromDigits` может вызываться также с двумя аргументами, в формате `FromDigits[list,b]`. В этом случае она дает десятичную запись числа, цифры которого в базе $b$ приведены в списке `list`, по умолчанию $b = 10$.

• При вызове функции `FromDigits` ни элементы списка цифр, ни база совершенно не обязаны быть цифрами!! Они могут быть любыми символами и даже выражениями. Например, вычисление `FromDigits[{a,b,c,d}]` даст `10*(10*(10*a+b)+c)+d`, а вычисление `Expand[FromDigits[{a,b,c,d},x]]` даст `d+c*x+b*x^2+a*x^3`. Таким образом, функция `FromDigits` дает еще один способ породить многочлен его списком коэффициентов!

• Функция `FromDigits` может использоваться, например, для того, чтобы задавать числа, цифры которых подчиняются простым закономерностям. В этом случае вместо того, чтобы вводить такое число с клавиатуры, обычно гораздо проще создать список его цифр, а потом переработать этот список в число посредством `FromDigits`. Например,

```
FromDigits[Flatten[Table[{1,2,3},{10}]]]
```

даст тридцатизначное число 123123123123123123123123123123, в котором 10 раз повторяется группа цифр `{1,2,3}`. Функция выравнивания списков `Flatten` использована здесь для того, чтобы убрать лишние скобки в получающемся при исполнении `Table` выражении `{{1,2,3},{1,2,3},...}`. Вот использование той же самой конструкции в более реалистическом контексте. Следующая программа пытается найти простые числа среди первых трех тысяч натуральных чисел, десятичная запись которых состоит из одних единиц:

```
onelist=Table[{n,FromDigits[Table[1,{n}]]},{n,1,3000}];
Select[onelist,PrimeQ[#[[2]]]&]
```

Мы знаем, что первым таким числом является 11. Интересно, есть ли другие? Оказывается, есть!! Вот часть ответа, получающегося при работе предыдущей программы:

```
{{2,11},{19,1111111111111111}, {23,11111111111111111111111}},
```

и, кроме того, среди таких чисел, у которых не больше 3000 знаков, имеется еще ровно два простых числа, в запись которых входит 317 и 1031 единиц, соответственно!!!

• Точно так же `DigitCount` может вызываться с одним, двумя или тремя аргументами. `DigitCount[n,b]` даст *список* количества цифр $1, 2, \ldots, b-1, 0$ в представлении *десятичного* числа в базе $b$, по умолчанию $b = 10$. Например, `DigitCount[100,2]` даст {3,4}. В самом деле, вычисление двоичного представления посредством `BaseForm[100,2]` показывает, что $100 =$ $1100100_2$.

• `DigitCount[n,b,d]` дает количество появлений цифры $d$ в выражении десятичного числа $n$ в базе $b$. Таким образом, если мы хотим увидеть количество появлений *десятичной* цифры $d$ в обычной десятичной записи числа $n$, нужно спросить `DigitCount[n,10,d]`.

• Вызываемая с одним аргументом команда `DigitCount[n]` эквивалентна `DigitCount[n,10,Mod[Range[10],10]]`.

В частности, играя в предыдущих командах с $b$ мы можем перевести число из одной базы в другую. Но для этого есть и специальные команды.

| | |
|---|---|
| `base^^x` | перевод числа из базы `base` в базу 10 |
| `BaseForm[x,base]` | перевод числа из базы 10 в базу `base` |

Использование этих команд таково. Вычисление `x^^b` переводит число из базы $b$ в десятичную форму. Например, `16^^100` дает `256`. При этом

• База $b$ может быть любым *десятичным* числом между 2 и 36.

• Для базы $b$ между 2 и 10 используются обычные десятичные цифры. Для базы $b$ между 11 и 36 в качестве дополнительных цифр используются буквы `a-z` — или, что *в этом констексте* то же самое, `A-Z`.

• Та же конструкция обслуживает и вещественные числа, достаточно ввести в последовательность цифр десятичную точку.

Рациональное число задается как отношение $m/n$ двух целых чисел. При этом автоматически производятся все возможные сокращения, а знак переносится в числитель. Следующие две команды выделяют числитель и знаменатель рационального числа. Рациональные числа считаются точными и все вычисления с ними производятся с *бесконечной* точностью. Вот команды для выделения числителя и знаменателя. В принципе их можно применять к любым выражениям, но для выражений, не являющихся дробями, они могут работать довольно необычным образом.

| | |
|---|---|
| `Numerator[x]` | числитель $x$ |
| `Denominator[x]` | знаменатель $x$ |

В соответствии с тем, что сказано выше, в дроби $-4/6$ будут произведены сокращения, так что `Numerator[-4/6]` равен `-2`, а `Denominator[-4/6]` равен `3`.

## § 10. Запись вещественного числа

Вещественные и комплексные числа в `Mathematica` могут быть как точными, так и приближенными. Рациональное число *всегда* точное, а точные иррациональные числа бывают алгебраическими и трансцендентными. Вычисления с алгебраическими числами производятся по специальным довольно сложным алгоритмам. С другой стороны, трансцендентные числа, между которыми нет очевидных зависимостей (ну как, скажем, между $e$ и $e^2$), рассматриваются как независимые полиномиальные переменные.

Приближенное вещественное число обычно характеризуется явным наличием **десятичной точки** (по-русски, запятой!) Добавление десятичной точки к целому числу, а также к числителю и/или знаменателю рационального числа превращает их в *приближенные* вещественные числа.

| | |
|---|---|
| `N[x]` | приближенное значение $x$ с машинной точностью |
| `N[x,m]` | приближенное значение $x$ с точностью $m$ цифр |
| `RealDigits[x]` | список цифр вещественного числа $x$ |
| `FromDigits[{list,m}]` | восстановление числа по списку цифр |

Команда `N` осуществляет переход от точного к приближенному вещественному или комплексному числу. Чтобы увидеть десятичное приближение к иррациональному числу $x$ с точностью до $m$ знаков, нужно вычислить `N[x,m]`, по умолчанию $m = 6$. В следующем параграфе мы посмотрим на примеры применения этой команды,

• Команда `RealDigits[x]` возвращает *список*, состоящий из двух частей: *списка* цифр и *целого числа* $m$, указывающего на сколько разрядов число $|x|$ нужно сдвинуть *вправо*, чтобы оно попало в отрезок $[0, 1]$, причем первая цифра после запятой стала отлична от 0. В случае положительного $m$, это в точности количество разрядов слева от десятичной точки. Таким образом, `RealDigits[N[Pi^5]]` вернет нам

```
{{3,0,6,0,1,9,6,8,4,7,8,5,2,8,1,4},3}
```

что следует понимать в том смысле, что первые *три* цифры в списке стоят слева от десятичной точки, так что целая часть числа $\pi^5$ равна 306. С другой стороны, если $m$ отрицательно, это значит, что первые $|m|$ цифр после десятичной точки равны 0.

• Как и `IntegerDigits`, команда `RealDigits[x]` игнорирует знак $x$.

• Функция `RealDigits` может вызываться с одним, двумя, тремя или четырбмя аргументами. Полный формат этой команды `RealDigits[x,b,l,n]` возвращает $l$ цифр числа $x$ по основанию $b$, начиная с позиции $b^n$. При этом база $b$ вовсе не обязана быть целым числом.

• По умолчанию `RealDigits[x]` возвращает список цифр, длина которого равна `Precision[x]`. Однако если мы вызываем `RealDigits` с тремя аргументами, в формате `RealDigits[x,b,l]`, причем $l$ больше, чем точность

`Log[10,b]Precision[x]`, то недостающие цифры заполняются значением `Indeterminate`.

• Для того, чтобы к *вещественному* числу можно было применить функцию `RealDigits`, оно должно быть *приближенным* числом. Так, скажем, попытка вычислить `RealDigits[Pi]` приведет к сообщению об ошибке, связанной с тем, что команда не может определить требуемую точность:

```
RealDigits: The number of digits to return
                    cannot be determined.
```

Правильный вопрос `RealDigits[Pi,10,50]` даст нам первые 50 цифр числа $\pi$. С другой стороны, того же эффекта можно добиться и при помощи `RealDigits[N[Pi]]`. Без указания количества цифр это даст нам цифры числа $\pi$ с машинной точностью:

```
{{3,1,4,1,5,9,2,6,5,3,5,8,9,7,9,3},1}
```

• С другой стороны для целых чисел и рациональных чисел с *конечным* десятичным разложением `RealDigits[x]` даст обычный список цифр. Для рационального числа $x$ с *бесконечным* десятичным разложением ответ будет представлен в виде `{{a1,...,ak,{b1,...,bl}},m}`, где $a_1, \dots, a_k$ — начальные цифры разложения $x$, $b_1, \dots, b_l$ — период, а $m$ имеет обычный смысл.

Обратимся теперь к основным формам записи вещественного числа.

| | |
|---|---|
| `MantissaExponent[x]` | мантисса и порядок $x$ |
| `NumberForm[x]` | математическая запись $x$ |
| `ScientificForm[x]` | научная запись числа $x$ |
| `EngeneeringForm[x]` | инженерная запись числа $x$ |

Вот как работают эти команды.

• Команда `MantissaExponent` создает список $(y, n)$, состоящий из мантиссы числа $x$ и его экспоненты При этом по возможности

$$0.1 \le y < 1, \text{ либо } -1 < y \le -0.1 \text{ и } x = 10^n y.$$

Например, вычисление `MantissaExponent[N[Sqrt[7]]]` даст `{0.264575,1}`. Обратите внимание, что команда `MantissaExponent` работает не только с приближенными, но и с точными вещественными числами!!

Все остальные команды переводят вещественное число в формат принятый, соответственно, в математике (`NumberForm`); в физике, химии и астрономии (`ScientificForm`) и, наконец, в технике (`EngeneeringForm`). В математическом формате число задается явным указанием позиции десятичной точки, без попытки отделить экспоненту 10. В научном формате положительное вещественное число $x$ представляется в виде $x = 10^n y$, где $1 \le y < 10$. В инженерном формате показатель степени всегда кратен 3. Иными словами, число $x$ представляется в виде $x = 10^n y$, где $1 \le y < 1000$. Вот, примерно, как это работает на практике:

```
In[20]:=numberforms={NumberForm,ScientificForm,EngineeringForm};
        Through[numberforms[N[Pi^10]]]
Out[20]={93648.04748,9.364804748×10^4,93.64804748×10^3}
```

Команда `Through` здесь осуществляет применение списка функций к одному и тому же аргументу.

Описанные выше команды представления вещественных чисел и родственные им команды (скажем, принятые в бухгалтерии учетная форма `AccountingForm` и раздутая форма `PaddedForm`) допускают большое количество опций, описывающих особенности выравнивания, разбиения на блоки, задания формы знака и экспоненты, разделителей и пр. Вот названия нескольких типичных опций: `DigitBlock`, `NumberSeparator`, `NumberPoint`, `NumberMultiplier`, `NumberSigns`, `NumberPadding`, `SignPadding`, `NumberFormat`, `ExponentFunction`. Все они служат для того, чтобы выводить большие объемы числовых данных в наглядном виде, согласованном с различными типографскими традициями, и начинающему не обязательно разбираться в том, что они делают.

Однако первые две опции настолько полезны, что мы все же укажем, как их следует изменить для лучшей читаемости ответа. Дело в том, что по умолчанию никаких разбиений цифр на блоки не производится, а если производится, то слева от *запятой* — на самом деле, по-английски, конечно, **точки** — блоки разделяются запятыми, а справа — пробелами. Иными словами, по умолчанию `DegitBlock->Infinity`, а `NumberSeparator->{",","` `"}`. Однако этим опциям можно, конечно, придавать другие значения. Мы обычно разбиваем ответ на блоки из пяти цифр, разделенные пробелами как слева, так и справа от запятой:

```
In[21]:=NumberForm[2^150,DigitBlock->5,NumberSeparator->" "]
Out[21]=1 42724 76927 05959 88105 82859 69449 49513 63827 46624
```

## § 11. Константы

> In any formula, constants (especially those obtained from handbooks) are to be treated as variables.
>
> The Tao of Real Programming

> The primary purpose of the DATA statement is to give names to constants; instead of referring to $\pi$ as 3.141592653589793 at every appearance, the variable PI can be given that value with a DATA statement and used instead of the longer form of the constant. This also simplifies modifying the program, should the value of $\pi$ change.
>
> FORTRAN manual

Вот некоторые наиболее важные константы встроенные в ядро системы `Mathematica`:

| | | |
|---|---|---|
| `Pi` | $\pi$ | |
| `E` | $e$ | основание натурального логарифма |
| `Degree` | $1^0$ | градус |
| `GoldenRatio` | $\phi = \dfrac{1+\sqrt{5}}{2}$ | золотое сечение |

В действительности в ядре системы и расширениях содержится значительное количество других констант, однако с математической точки зрения они не могут интересовать начинающего, а с технической точки зрения их использование не содержит *никаких* новых моментов, по сравнению с использованием `E` или `Pi`. Для того же, кто интересуется асимптотикой, интегральными формулами, статистикой или чем-нибудь в таком духе, обращение к другим встроенным в ядро или описанным в пакетах константам, скажем к `EulerGamma`, `Catalan` или `Khinchin`, не представит никакого труда.

Пожалуй, из перечисленных констант самой интересной является градус. Дело в том, что из школьной программы многие вообще не выносят впечатления, что градус является вещественным числом. По этому поводу в учебниках произносятся загадочные и абсолютно невразумительные *заклинания* на тему о мере угла и пр. В действительности $1^\circ$ никогда не был ничем иным, кроме обычного вещественного числа, равного $\pi/180$. Вот его первые 50 значащих цифр:

```
In[22]:=N[Degree,50]
Out[22]=0.01745329251994329576923690768488612713442871888541 7
```

Еще одна константа, играющая огромную роль в дальнейшем — это **золотое сечение**. Так называется больший из корней уравнения $x - \dfrac{1}{x} = 1$, равный $\phi = \dfrac{1+\sqrt{5}}{2}$. Легко видеть, что $\phi = 2\cos\left(\dfrac{\pi}{5}\right)$ — это в точности диагональ правильного пятиугольника со стороной 1, вот численное значение $\phi$:

```
In[23]:=N[GoldenRatio,50]
Out[23]=1.6180339887498948482045868343656381177203091798058
```

В отличие от принятого в континентальной Европе немецкого индустриального стандарта `DIN` система `Mathematica` по умолчанию считает, что высота и ширина стандартной **книжной страницы** (`portrait orientation`) соотносятся как $\phi : 1$, а стандартной **альбомной страницы** (`landscape orientation`) — соответственно, как $1 : \phi$.

**Комментарий.** В математической литературе число $\phi$ обычно обозначается через $\tau$ и называется **отношением крайнего и среднего** (extreme and mean ratio), **божественной пропорцией** (divina proportione) или **числом Фидия**. Золотое сечение часто использовалось греческими скульпторами и архитекторами и явно описывается в "Элементах" Эвклида, который рассматривает пропорцию $\frac{y}{z} = \frac{y+z}{y}$. Ясно, что в терминах $x = \frac{y}{z}$ эта пропорция переписывается в виде $x = 1 + \frac{1}{x}$, а как мы знаем $x = \tau$ как раз и является положительным корнем этого уравнения. Самое знаменитое классическое произведение, специально посвященное золотому сечению — это книга Луки Пачоли "De divina proportione", в иллюстрациях к которой использованы модели и 59 таблиц, изготовленные его близким другом Леонардо да Винчи. Третья часть книги Пачоли представляет собой итальянский перевод книги Пьеро делла Франческа "De quinque corporibus regularibus". С математической точки зрения совершенно исключительная роль золотого сечения связана с тем, что это базисный элемент кольца целых поля $\mathbb{Q}[\sqrt{5}]$, при помощи которого строится конечная группа кватернионов порядка 120, которая в свою очередь ответственна за существование правильного додекаэдра и правильного икосаэдра.

Отношение `Mathematica` к константам заслуживает отдельного обсуждения. Дело в том, что разные части системы трактуют их по разному, причем по умолчанию они понимаются как *символы*, а не как числа:

```
In[24]:=Head[GoldenRatio]
Out[24]=Symbol
```

Вот основные точки зрения на константы, используемые в различных типах вычислений:

• С точки зрения численных вычислений все эти константы считаются *численными величинами* (`NumericQ[Pi]` дает `True`) и имеют численные значения, которые можно с любой точностью найти при помощи функции `N`.

• С точки зрения алгебраических вычислений все эти константы — включая золотое сечение, являющееся алгебраическим числом!!! — трактуются как *независимые переменные*. Следующий диалог иллюстрирует различие между *алгебраическим числом* $(1 + \sqrt{5})/2$ и *константой* `GoldenRatio`, которая трактуется как символ. Кроме того, этот диалог показывает, что для получения осмысленного ответа полезно применять функцию `Simplify` или `FullSimplify`.

```
In[25]:=((1+Sqrt[5])/2)^2-(1+Sqrt[5])/2-1
```

$$\text{Out[25]}= -1 + \frac{1}{2}(-1 - \sqrt{5}) + \frac{1}{4}(1 + \sqrt{5})^2$$

```
In[26]:=Simplify[%]
Out[26]=0
In[27]:=Simplify[GoldenRatio^2-GoldenRatio-1]
Out[27]=-1-GoldenRatio+GoldenRatio^2
In[28]:=FullSimplify[GoldenRatio^2-GoldenRatio-1]
Out[28]=0
```

• С точки зрения аналитических вычислений все эти константы считаются *константами*, они имеют атрибут `Constant`, и поэтому трактуются как константы при вычислении производных и интегралов по всем *допустимым* переменным.

**Упражнение.** Если Вы *внимательно* прочитали предыдущую фразу, скажите, что произойдет, если попытаться вычислить `D[Pi,Pi]`?

**Ответ.** А ничего, система просто ответит, что по `Pi` дифференцировать нельзя: `General: Pi is not a valid variable`.

## § 12. Непрерывные дроби и рациональные приближения

В настоящем параграфе мы рассматриваем еще один способ задания вещественных чисел и операции, превращающие приближенное вещественное число в точное рациональное число. Содержание этого параграфа носит чисто развлекательный характер и не используется в дальнейшем.

| | |
|---|---|
| `Rationalize[x,d]` | хорошее рациональное приближение к $x$ |
| `ContinuedFraction[x,n]` | разложение $x$ в непрерывную дробь |
| `FromContinuedFraction[x]` | восстановление $x$ по непрерывной дроби |

Функция `Rationalize[x]` строит хорошее рациональное приближение к *приближенному* вещественному числу $x$. Попытка вычислить `Rationalize[Pi]` вернет $\pi$, потому что система не знает, приближение какой точности нас интересует. Не приведет к успеху и попытка вычисления `Rationalize[N[Pi]]`. Дело в том, что по используемому функцией `Rationalize` критерию у вычисленного с машинной точностью числа `N[Pi]` нет *хороших* рациональных приближений. А именно, рациональное число $p/q$ рассматривается в качестве хорошего приближения вещественного числа $x$, если $|x - p/q| < c/q^2$, где по умолчанию $c = 10^{-4}$.

Правильная форма состоит в том, чтобы спросить `Rationalize[Pi,10^-n]`, явно задав допуск (`tolerance`). Вот построение рациональных приближений числа $\pi$ с точностью до $1, 2, \ldots, 10$ десятичных знаков после запятой:

```
In[29]:=Map[Rationalize[Pi,10^(-#)]&,Range[10]]
Out[29]={22/7, 22/7, 201/64, 333/106, 355/113, 355/113, 75948/24175,
                                100798/32085, 103993/33102, 312689/99532}
```

А вот, что такое *на самом деле* используемое машинное приближение числа $\pi$. Вычисление `Rationalize[N[Pi],0]` с нулевым допуском дает `245850922/78256779`.

Функция `ContinuedFraction[x,n]` выдает список первых $n$ членов в разложении $x$ в непрерывную дробь. Функция `ContinuedFraction[x]`, вызванная с одним аргументом порождает список всех членов, которые можно получить исходя из заданной точности $x$. Список `{n1,...,ns}` истолковывается обычным образом:

$$[n_1,\dots,n_s] = n_1 + \cfrac{1}{n_2 + \cfrac{1}{n_3 + \cfrac{1}{n_4 + \dots}}}$$

В случае квадратичных иррациональностей ответ имеет вид

`{m1,...,ms,{n1,...,nt}}`

где $m_1,\dots,m_s$ — начальные члены непрерывной дроби, а $n_1,\dots,n_t$ — период.

Вычислим для примера выражения нескольких важнейших квадратичных иррациональностей:

```
In[30]:=Map[ContinuedFraction,{GoldenRatio,GoldenRatio^-1,Sqrt[2],Sqrt[3]}]
Out[30]={{1,{1}},{0,{1}},{1,{2}},{1,{1,2}}}
```

Функция `FromContinuedFraction` является обратной к функции `ContinuedFraction`. Она действует обычным образом, восстанавливая исходное рациональное число — или при наличии периода, исходную квадратичную иррациональность!! — по его/ее разложению в непрерывную дробь. Например, вычисление `FromContinuedFraction[{1,{1,2,3}}]` дает $(-1+\sqrt{37})/3$.

Хорошее рациональное приближение к вещественному числу $x$ можно построить также применив к числу $x$ функцию `ContinuedFraction[x,n]` с большим значением $n$, а потом применив к результату функцию `FromContinuedFraction[x]`. Вот, для примера, первые 24 знака разложения $e$ в непрерывную дробь:

```
In[31]:=ContinuedFraction[E,24]
Out[31]={2,1,2,1,1,4,1,1,6,1,1,8,1,1,10,1,1,12,1,1,14,1,1,16}
```

Если Вы не видели этого раньше (например, не читали Кнута!), это *должно* произвести впечатление и Вы, конечно, тут же попытаетесь вычислить `ContinuedFraction[E,1000]`. Результат будет поучителен во всех отношениях — как с точки зрения рациональных приближений, так и с точки зрения работы системы. Итак, вот эта дробь в традиционной математической форме:

$$2 + \cfrac{1}{1 + \cfrac{1}{2 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{4 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{6 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{8 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{10 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{12 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{14 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{16+1}}}}}}}}}}}}}}}}}}}}}}}}$$

Несложное вычисление с использованием функции `FromContinuedFraction` показывает, что значение этой непрерывной дроби равно

```
In[32]:=FromContinuedFraction[{2,1,2,1,1,4,1,1,6,1,1,8,1,1,10,1,1,
                                                    12,1,1,14,1,1,16}]
Out[32]=14013652689/5155334720
```

Это рациональное число, для записи которого требуется 21 цифра, совпадает с $e$ с точностью до 20 десятичных знаков.

## § 13. Комплексные числа

В системе `Mathematica` комплексное число тоже может быть точным или приближенным. Обычно комплексное число представляется в алгебраической форме $z = x + yi$, причем оно считается точным в том и только том случае, когда как его вещественная часть $x = \mathrm{re}(z)$, так и его мнимая часть $y = \mathrm{im}(z)$ точные.

| | | |
|---|---|---|
| `I` | $i$ | мнимая единица |
| `z=x+I*y` | $z = x + iy$ | комплексное число $z$, где $x, y \in \mathbb{R}$ |
| `Re[z]` | $\mathrm{re}(z)$ | вещественная часть $z$ |
| `Im[x]` | $\mathrm{im}(z)$ | мнимая часть $z$ |
| `Abs[z]` | $\lvert z\rvert$ | модуль $x$ |
| `Arg[z]` | $\arg(z)$ | аргумент $z$ |
| `Conjugate[z]` | $\overline{z} = x - iy$ | сопряженное к $z$ |
| `ComplexExpand[z]` | | выделение $\mathrm{re}(z)$ и $\mathrm{im}(z)$ |

Использование большинства этих функций совершенно ясно из названия. Так, `I` обозначает мнимую единицу $i$, а `Re[z]` и `Im[z]` дают вещественную и мнимую часть комплексного числа $z$, соответственно. Функция `Conjugate[z]` сопоставляет числу $z = x + yi$ комплексно сопряженное $\overline{z} = x - yi$.

• Вопрос от точности или приближенности решается *отдельно* для вещественной и мнимой частей. Например, у комплексного числа `1.+1*I` вещественная часть приближенная, но мнимая часть точная. Тем самым, вычисление `Re[1.+1*I]` даст *приближенное* вещественное число `1.`, но вычисление мнимого числа `Im[1.+1*I]` дает точное `целое` число `1` — несмотря на то, что число `1.+1*I` приближенное!! Функция `N` обычным образом действует на комплексных числах. Так, например, вычисление `N[1+I]` дает `1.+1.*I`.

• Если мнимая часть комплексного числа *точно* равна 0, то оно автоматически упрощается до вещественного числа. Так, например, вычисление `Head[1+0*I]` дает `Integer`, а вычисление `Head[1.+0*I]` дает `Real`. В то же время, если мнимая часть комплексного числа *приближенно* равна 0, то оно все равно считается комплексным!!! Так, в частности, вычисление `Head[1+0.*I]` дает `Complex`. Это различие становится весьма существенным, когда мы начинаем реально использовать многозначные функции.

• Следующий в высшей степени поучительный диалог иллюстрирует разницу между `Simplify` и `FullSimplify`:

```
In[33]:=Simplify[Re[z]+I*Im[z]]
Out[33]=I*Im[z]+Re[z]
In[34]:=FullSimplify[Re[z]+I*Im[z]]
Out[34]=z
```

Кроме алгебраической формы комплексное число можно задавать и в полярной форме `Abs[z]*Exp[I*Arg[z]]`, которую в русской учебной литературе принято называть тригонометрической. В этой форме используется две другие функции:

- Модуль `Abs[z]` комплексного числа $z = x + yi$, равный $\sqrt{x^2 + y^2}$.
- Аргумент `Arg[z]`, т.е. такое число $\phi \in [-\pi, \pi]$, что $\mathrm{tg}(\phi) = y/x$. В тех случаях, когда это возможно, `Arg` возвращает *точные* значения. Так, например, `Arg[-1/2+Sqrt[3]*I/2]` дает $2\pi/3$, `Arg[-1/2-Sqrt[3]*I/2]` дает $-2\pi/3$.

Особенно интересна функция `ComplexExpand`, которая явным образом выделяет из комплексного числа вещественную и мнимую часть, считая, что все входящие в выражение этого числа переменные являются вещественными.

```
In[35]:=ComplexExpand[Exp[x+y*I]]
Out[35]=e^x Cos[y]+Ie^x Sin[y]
In[36]:=ComplexExpand[Log[x+y*I]]
Out[36]=I*Arg[x+I*y]+1/2*Log[x^2+y^2]
In[37]:=ComplexExpand[Cos[x+y*I]]
Out[37]=Cos[x]*Cosh[y]-I*Sin[x]*Sinh[y]
In[38]:=ComplexExpand[Sin[x+y*I]]
Out[38]=Cosh[y]*Sin[x]+I*Cos[x]*Sinh[y]
```

## § 14. Генерация случайных чисел

Для многих целей, например для тестирования программ, полезно уметь генерировать (псевдо)-случайные числа, которые можно вводить в качестве данных в вычисления для проверки корректности и скорости работы используемых алгоритмов. В дальнейшем мы будем опускать префикс *псевдо-* и говорить просто о *случайных* числах. В системе `Mathematica` случайные числа генерируются при помощи следующих программ.

| | |
|---|---|
| `Random[]` | случайное вещественное число на отрезке $[0, 1]$ |
| `Random[Integer]` | 0 или 1 с вероятностью $1/2$ |
| `Random[Integer,{m,n}]` | случайное целое число между $m$ и $n$ |
| `Random[Real,x]` | случайное вещественное число на отрезке $[0, x]$ |
| `Random[Real,{x,y}]` | случайное вещественное число на отрезке $[x, y]$ |
| `Random[Complex]` | случайное комплексное число в $[0, 1]^2$ |
| `Random[Complex,{z,w}]` | случайное комплексное число в прямоугольнике |
| `SeedRandom[]` | перезапустить генератор случайных чисел |
| `$RandomState` | текущее состояние генератора случайных чисел |

Функцию `Random` можно вызвать вообще без аргументов, с одним или двумя аргументами, причем первый аргумент обязан быть *типом* числа (`Integer`, `Real` или `Complex`), а второй аргумент может быть числом или списком из двух чисел.

• По умолчанию функция `Random[]`, вызванная без аргументов, или, что то же самое, `Random[Real]`, дает равномерно распределенное случайное вещественное число на отрезке $[0,1]$.

• Функция `Random`, вызванная в формате `Random[Real,x]` дает случайное число на отрезке $[0,x]$, при этом $x$ не обязано быть положительным! Наконец, функция `Random`, вызванная в формате `Random[Real,{x,y}]` дает случайное число на отрезке $[x,y]$.

• То же самое относится к целым числам. Функция `Random[Integer]` случайным образом генерирует 0 или 1 с вероятностью $1/2$. Функция `Random[Integer,n]` генерирует случайное целое между 0 и $n$ включительно. Наконец, функция `Random[Integer,{m,n}]` порождает случайное целое на отрезке $[m,n]\cap\mathbb{Z}$ — включая концы!!

• Чуть труднее приноровиться к вызову функции `Random` в комплексной области. В этом случае функция `Random[Complex,{z,w}]` порождает случайное комплексное число в прямоугольнике на комплексной плоскости с углами $w$ и $z$. Этот прямоугольник имеет вид

$$\{x+yi \mid a\le x\le b, c\le y\le d\},$$

где мы для краткости положили

$$\begin{aligned} a &= \min(\mathrm{re}(z),\mathrm{re}(w)), \qquad & b &= \max(\mathrm{re}(z),\mathrm{re}(w)),\\ c &= \min(\mathrm{im}(z),\mathrm{im}(w)), \qquad & d &= \max(\mathrm{im}(z),\mathrm{im}(w)). \end{aligned}$$

Конечно, этот прямоугольник может быть и вырожденным. Так, скажем, `Random[Complex,{-1+I,-1+3*I}]` даст случайное число на отрезке длины 2, параллельном мнимой оси. Разумеется, того же эффекта можно добиться посредством `-1+I*Random[Real,{1,3}]`. То же, конечно, относится и к общему случаю. Из общих соображений совершенно ясно, что никаких особых алгоритмов для генерации комплексных чисел не применяется, а отдельно генерируются вещественная и мнимая части в нужных пределах. В том, что это фактически так и есть, можно убедиться, например, следующим непритязательным образом:

```
 In[39]:=SeedRandom[117]; Random[Complex]
Out[39]=0.85043+0.266285*I
 In[40]:=SeedRandom[117]; Random[]+I*Random[]
Out[40]=0.85043+0.266285*I
```

Таким образом, для генерации комплексных чисел в прямоугольнике мы могли бы точно с таким же успехом пользоваться, например, такой командой:

```
Random[Real,{a,b}]+I*Random[Real,{c,d}]
```

**Упражнение.** Тип случайной переменной может быть только целым, вещественным или комплексным числом. Попытка задать случайное рациональное число посредством `Random[Rational]` немедленно приведет к следующему сообщению об ошибке:

```
Random: Type specification Rational in Random[Rational]
                    should be Real, Integer, or Complex.
```

Как можно породить случайное рациональное число?

**Упражнение.** Как породить случайное комплексное число в единичном круге?

В действительности, как должно быть ясно уже из приведенного выше примера, когда мы дважды разными командами породили одно и то же "случайное" комплексное число, функция `Random` является **точной** функцией, хотя *случайному* наблюдателю это не должно быть видно. Например, вызвав функцию

```
Table[Random[Integer,{0,9}],{i,1,50}]
```

Вы получите случайную последовательность из 50 цифр. Вызвав эту функцию еще раз, Вы получите уже совершенно другую последовательность. Это связано с тем, что *каждый* вызов функции `Random` меняет текущее значение системной переменной `$RandomState`. Однако часто при отладке или сравнении программ желательно несколько раз вызывать одно и то же случайное число. Это делается одним из двух способов:

• Функция `SeedRandom[n]` перезапускает генератор случайных чисел, используя $n$ в качестве затравки. Эта функция позволяет несколько раз получать одну и ту же последовательность случайных чисел!!! Например, вызвав функцию

```
SeedRandom[123]; Table[Random[Integer,{0,9}],{i,1,50}]
```

Вы как и раньше получите случайную последовательность из 50 цифр. Вызвав эту функцию еще раз, Вы получите ту же самую последовательность. Чтобы получить другую последовательность, Вам теперь нужно изменить аргумент `SeedRandom`!!! Функция `SeedRandom[ ]`, вызванная вообще без аргументов, перезапускает генератор случайных чисел, используя в качестве затравки время суток. Другой народный способ, смысл которого состоит в том, чтобы ослабить роль человеческого фактора, состоит в том, чтобы вначале исполнить команду

```
seed=Random[Integer,{2000,3000}],
```

а уже потом подставлять `SeedRandom[seed]`.

• Еще один способ получить ту же самую последовательность таков. Мы уже упоминали, что значение системной переменной `$RandomState` меняется при каждом обращении к команде `Random`. Например, исполнив несколько раз программу

```
Random[]; Total[IntegerDigits[$RandomState]]
```

можно убедиться в том, что `$RandomState` представляет собой целое число в котором от 2400 до 2800 цифр. В любой момент сессии выполнив присваивание `state=$RandomState;` можно записать текущее состояние генератора случайных чисел. В дальнейшем можно восстановить это значение посредством `$RandomState=state;` для последующего получения *тех же самых* случайных чисел.

**Предостережение.** Наивная точка зрения состоит в том, что мы могли бы снова явно породить `state=Random[Integer,{10^2400,10^2800}]` в нужных пределах. Однако в отличие от затравки `RandomSeed`, которую можно порождать произвольно, далеко не все что угодно может быть значением `$RandomState`!!! Скорее всего, результатом попытки исполнить команду `$RandomState=state` будет сообщение об ошибке

```
$RandomState: blabla<<bla>>blabla is not a valid random state.
```

## § 15. Бесконечность и неопределенность

Есть еще несколько очень важных вещественных/комплексных **не-чисел** (`not-numbers`), которые могут возникать в вычислениях, как арифметических, так и аналитических, в особенности при вычислении пределов, бесконечных сумм и произведений, интегралов и пр.

| | |
|---|---|
| `Infinity` | вещественная бесконечность |
| `Indeterminate` | неопределенность |
| `ComplexInfinity` | комплексная бесконечность |
| `DirectedInfinity` | направленная комплексная бесконечность |

При попытке вычислить `1/0` выдается сообщение об ошибке

```
Power: Infinite expression 1/0 encountered.
```

Тем не менее, `Mathematica` знает ответ `ComplexInfinity`. Как мы сейчас убедимся, при попытке вычислить `0/0` мы получим сразу два сообщения об ошибке. Вот четыре классических неопределенности:

```
In[41]:={0/0,0*Infinity,Infinity/Infinity,Infinity-Infinity}
        Power: Infinite expression 1/0 encountered.
        Infinity: Indeterminate expression 0*ComplexInfinity
                encountered.
        Infinity: Indeterminate expression 0*Infinity
                encountered.
        Infinity: Indeterminate expression 0*Infinity
                encountered.
        Infinity: Indeterminate expression -Infinity+Infinity
                encountered.
Out[41]={Indeterminate,Indeterminate,Indeterminate,Indeterminate}
```

Кроме того, в соответствии с аналитическими традициями, `Mathematica` верит, что `0^0` тоже является неопределенностью. Алгебраисты **знают**, что на самом деле $0^0 = 1$. В действительности, в это должен верить каждый,

кто верит в справедливость формулы $(x+y)^2 = x^2y^0 + 2x^1y^1 + x^0y^2$. Однако принятое в `Mathematica` соглашение весьма обычно при вычислении пределов и других аналитических вычислениях.

Неопределенность действует следующим образом. Если в процессе вычислений встретился неопределенный результат, то действие обычных правил арифметики прекращается и все последующие содержашие его результаты тоже автоматически становятся неопределенными.

Что касается бесконечностей, то они точное соотношение между ними следующее. Самой общей из них является функция `DirectedInfinity`. А именно, `DirectedInfinity[z]` порождает бесконечность в направлении комплексного числа $z$. С внутренней точки зрения остальные перечисленные разновидности бесконечностей являются просто сокращениями:

• Комплексная бесконечность `ComplexInfinity` является просто сокращением для `DirectedInfinity[]`, т.е. для направленной бесконечности вызванной вообще без направления (= с неопределенным направлением).

• Вещественная бесконечность `Infinity` рассматривается как сокращение для `DirectedInfinity[1]`, т.е. для направленной бесконечности вызванной в направлении положительной вещественной полуоси

• Вещественная бесконечность `-Infinity` рассматривается как сокращение для `DirectedInfinity[-1]`, т.е. для направленной бесконечности вызванной в направлении отрицательной вещественной полуоси

• Для многих аналитических вычислений, скажем, при вычислении контурных интегралов, нужна бесконечность в других направлениях, ну, скажем, `DirectedInfinity[I]` направленная вдоль положительной мнимой полуоси. В ответе такая бесконечность будет выводиться как `I*Infinity`, но ее полная внутренняя форма, все-таки `DirectedInfinity[I]`.

Вещественные бесконечности `Infinity` и `-Infinity` можно использовать при задании итераторов в суммах и произведениях, при вычислении пределов, задании пределов интегрирования и всех других аналогичных манипуляциях. В подавляющем большинстве случаев `Mathematica` верно истолковывает Ваши намерения. Лет 25 назад злопыхатели широко распространяли слухи о мнимых *ошибках* в `Mathematica` и `Maple`, однако, как выяснилось, в действительности, ошибки содержались в (составленных людьми!!!) таблицах рядов и интегралов, использовавшихся в библиотеках этих систем! В действительности, вычисление `Sum[1/i^2,{i,1,Infinity}]` дает точное значение $\pi^2/6$, как и задумано. Вероятность наткнуться на ошибку во всех обычных вычислениях такого рода меньше, чем вероятность наткнуться на ошибку в печатных версиях таблиц рядов и интегралов.

# Глава 5. ПЕРЕМЕННЫЕ И ИХ ЗНАЧЕНИЯ

> Язык их ясен и конкретен. Для простоты дела все животные, ракушки, птицы, рыбы, ягоды, мох и змеи, кроме самых ядовитых, называются у них одинаково.
>
> Александр и Ольга Флоренские. Движение в сторону Йые

> Osborn's Law: Variables won't; constants aren't.

В этой главе мы начинаем обсуждать ключевые понятия, которые по своей сути относятся к программированию и требуют уточнения традиционного математического языка посредством введения в него (не на лексическом, а на грамматическом уровне!!!) идеи времени. Математика не знает идеи времени, в то время как для *фактического* вычисления последовательность выполнения операций (`flow of calculation`) становится исключительно важным, а часто решающим обстоятельством. Однако при этом традиционное программирование имело дело с весьма ограниченным набором вычислений с достаточно примитивными типами объектов. Иными словами, математики проводили нетривиальные вычисления с интересными объектами, но делали это плохо — по крайней мере с точки зрения возможности перенести их вычисления на компьютер!!! Основным, чтобы не сказать *единственным*, инструментом математиков всегда было **понимание**, так что вопрос о воспроизводимости, не говоря уже об эффективности, вычисления их не интересовал. В то же время программисты традиционно проводили бессодержательные вычисления с лишенными интереса объектами, но всегда делали это правильно и систематически. В этом смысле точка зрения компьютерной алгебры является эффективным синтезом математической и программистской точек зрения. Как и в математике, мы проводим *любые* вычисления с *любыми* объектами, доступными нашему умозрению, но как и в программировании делаем это эффективно переносимыми методами, такими методами, которые опираются не на медитацию и непосредственный контакт с миром идей, в мистических и практических аспектах, а лишь на четко описанные простые, явные и действенные алгоритмы.

## § 1. Математика как неточная наука

> The problem with you mathematicians is that you are so imprecise.
>
> Real Programmer

Люди, не имеющие отношения к математике, обычно характеризуют ее как *точную науку*. В этой фразе верно все, кроме слов *точная* и *наука*. Как говорит само ее название, математика является *учением*, *доктриной* и *областью знания*. Кроме того, в настоящее время она организационно, а отчасти идеологически и психологически, близка к некоторым областям науки. В то же время математика, конечно, не является наукой, так как ее цели, метод и критерии ценности гораздо ближе к искусству или религии

и имеют мало общего с целями, методами и установками, характерными для естествознания. С другой стороны, и точность в математике важна только как инструмент, или побочный продукт, а вовсе не как самоцель. Более того, чем лучше мы понимаем, что говорится, тем менее точными мы можем позволить себе быть.

По стандартам многих других видов деятельности, скажем, программирования, математика является весьма неточным занятием. В предыдущей главе мы уже упоминали, что обычные математические обозначения крайне неоднозначны. Как отмечает Виктор Анатольевич Уфнаровский[50], все три минуса, встречающихся в выражении $-(-1-2)$, использованы в трех совершенно разных смыслах:

- первый минус обозначает *унарную* операцию $x \mapsto -x$;
- второй минус является частью *имени* числа $-1$;
- третий минус обозначает *бинарную* операцию $(x, y) \mapsto x - y$.

Причина, по которой это мало кого волнует и практически никогда не создает *серьезных* неудобств, состоит в том, что пользующийся этими обозначениями человек *иногда* до некоторой степени понимает смысл текста.

Но, вероятно, самым сомнительным математическим знаком является знак $=$. Вот простенький текст, подтверждающий эту декларацию: "Пусть $f = x^2 - ax + 1$. Положим $a = 2$. Тогда $f = (x-1)^2$ и, значит, уравнение $x^2 - ax + 1 = 0$ имеет единственный корень $x = 1$ кратности 2". Что мы подразумеваем, когда пишем $f(x) = g(x)$? Предлагаем ли мы положить значение неизвестной функции $f$ в точке $x$ равным $g(x)$? Или мы предлагаем положить его равным $g(x)$ в *любой* точке $x$? Утверждаем ли, что значения *известных* нам функций $f$ и $g$ равны при любом[51] $x$ из области определения? Или из какого-то содержащегося в ней множества? Или при каком-то/каких-то $x$, который/которые предлагается найти? Ясно, что все это абсолютно различные смыслы и читающий математический текст человек, который в состоянии следить за происходящим, *обычно* в состоянии истолковать, какой из этих смыслов имеется в виду: конечно, это зависит от степени его квалификации, сомнительная ситуация легко поставит начинающего в тупик, но зрелый читатель, скорее всего, решит, что либо в тексте допущена опечатка, либо автор сам не понимает, о чем говорит.

С точки зрения языка `Mathematica` знак равенства в математическом тексте может обозначать любую из (по крайней мере!!!) восьми абсолютно различных вещей:

- шесть **операторов** `=`, `:=`, `->`, `:>`, `^=`, `^:=`;
- два **отношения** `==`, `===`.

---

[50] В.А.Уфнаровский, Математический аквариум. — Кишинев, Штиинца, 1987, с.1–215; стр.30.

[51] Мы совершенно оставляем в стороне структурные вопросы, которые с таким трудом усваиваются начинающими, например, чем равенство семейств отличается от равенства функций, и почему это из того, что $f(x) = g(x)$ для любого $x$ из области определения еще не следует, что $f = g$?

Как смысл этих вещей, так и — тем более!!! — их использование, настолько отличаются между собой, что непоколебимое упорство математиков в желании обозначать все эти понятия одним и тем же символом = служит еще одним парадоксальным подтверждением сакраментальной дефиниции Пуанкаре: МАТЕМАТИКА — ЭТО ИСКУССТВО НАЗЫВАТЬ РАЗНЫЕ ВЕЩИ ОДНИМ ИМЕНЕМ. В частности, в предыдущем тексте употреблено ПЯТЬ СОВЕРШЕННО РАЗНЫХ ЗНАКОВ РАВЕНСТВА: `:=`, `=`, `===`, `==`, `->`.

А ведь до сих пор мы практически не упоминали того, что известно в `Computer Science` как **ход вычисления** (`flow of calculation`), иными словами, то, что сами значения $f$, $g$ и $x$ меняются в процессе вычисления — математик использовал бы в этом смысле тысячу индексов, что было бы, конечно, *чуть* точнее, чем принятое в программировании понятие **переменной**, но *гораздо* менее удобно!!! Например, когда мы говорим, ‘положим $a = 2$’, то совершенно неясно, имеем ли мы в виду

• **присваивание**: $a$ становится НАВСЕГДА равным 2, на языке `Mathematica` `a=2`, или

• **подстановку**: $a$ становится равным 2 ТОЛЬКО В ЭТОМ ВЫЧИСЛЕНИИ, на языке `Mathematica` `a->2`.

Но ведь если мы сами этого не понимаем, нам вряд ли удастся объяснить это даже такой интеллигентной системе, как `Mathematica`.

## § 2. Переменные и их значения

> Переменная — это область в памяти компьютера, где может храниться некоторое значение для использования его в программе.
>
> Харви Дейтел, Пол Дейтел, “Как программировать на `C++`”,

С точки зрения математика переменная — например, буковка $x$ в записи многочлена $f = a_n x^n + \ldots + a_1 x + a_0 \in K[x]$ над полем $K$ — является некоторым новым **трансцендентным** объектом. Этот объект не принадлежит основному полю $K$ и сам не имеет никакого *значения*, но именно поэтому его можно без противоречия **специализировать** в различные элементы поля $K$ — или, на классическом языке, *придавать* ему различные значения. В этом случае математик напишет $x \mapsto x_1$, $x \mapsto x_2$, …, но сама ПЕРЕМЕННАЯ НИКОГДА НЕ МЕНЯЕТСЯ — ее можно только *заменить*, но в ней нет ничего, что можно было бы *изменить*. Для традиционного программиста, который имел дело только с численными вычислениями, переменная это нечто совсем иное. Это не какой-то потусторонний объект, а просто метка или ярлык, вроде номера ячейки, которая обозначает конкретное *число*, получающееся из каких-то других данных в программе и/или хранящееся где-то в памяти компьютера. Число, хранящееся под этим ярлыком, может меняться, но программист ссылается на эти разные значения как на *одну и ту же* переменную, он считает, что сама переменная $x$ *равна* $x_1, x_2, \ldots$ В РАЗНЫЕ МОМЕНТЫ ВРЕМЕНИ. Точка зрения компьютерной алгебры представляет собой соединение двух этих противоположных точек зрения. С точки зрения `flow of calculation` переменная

компьютерной алгебры — это переменная традиционного программирования, но только теперь ее значения совсем не обязательно являются числами! В действительности, то, что делает компьютерную алгебру столь мощным инструментом, это как раз **полиномиальные вычисления**, т.е. способность истолковывать значения переменных (в компьютерном смысле) как переменные (в математическом смысле)!!

Основой любых символьных вычислений является способность системы рассматривать символы как **переменные** и присваивать им **значения**, которые могут быть любыми допустимыми в контексте проводимого вычисления выражениями, например, числами или другими символами. При этом, как и в традиционном программировании, в разные моменты вычисления один и тот же символ может иметь различные значения. То значение, которое символ имеет в данный момент вычисления, называется **текущим значением**. В следующих параграфах мы обсудим основные способы присваивания, временного присваивания (подстановки) и модификации значений переменных. Сделаем пока несколько общих замечаний:

• Если какому-то символу не присваивалось никаких значений, значением этого символа является он сам, рассматриваемый как независимая полиномиальная переменная. В частности, до дальнейших инструкций числа `Pi`, `E` – и даже `GoldenRatio`!!! – рассматриваются как *независимые* переменные.

• Все переменные сохраняют свои значения на протяжении сессии, до тех пор, пока эти значения не были модифицированы или вычищены. Основной причиной ошибок при вычислениях является использование в них переменных, которым уже присвоены значения, в §§ 7–10 мы обсудим несколько способов избежать этого (использование подстановок вместо присваиваний, чистка и создание локальных переменных). Однако самое простое — давать переменным говорящие имена.

• Некоторые вещи, не являющиеся символами, (целые и рациональные числа, стринги и объекты некоторых других специальных форматов) с точки зрения подавляющего большинства команд считаются имеющими **постоянное значение**, и их текущее значение *в любом выражении* всегда равно этому постоянному значению. Такие вещи называются **сырыми объектами** (`raw objects`) в том смысле, что перед употреблением в пищу их *не обрабатывают*.

**Предостережение.** Из приведенных выше примеров можно сделать поспешный вывод, что сырыми бывают только атомарные объекты. Однако в действительности сырой объект может быть устроен **сколь угодно сложно**, например, подавляющее большинство команд рассматривает разреженные объекты типа `SparseArray` как сырые, это делается для того, чтобы избежать вычислений с матрицами с миллиардами или десятками миллиардов коэффициентов. Ясно, что вычисления с такими матрицами на бытовом компьютере по обычным алгоритмам работы над списками, когда каждая компонента занимает собственный сектор памяти, *физически* невозможны. Поэтому их можно обрабатывать *только* при помощи небольшого количества специальных команд, которые правильно учитывают их структуру и специфику (некоторые из этих команд называ-

ются обычными именами, но работают совсем иначе). Точно также стринги являются сырыми объектами *только* с точки зрения обычных *математических* вычислений. Специальные **текстовые команды** работы над стрингами позволяют сливать, изменять и редактировать стринги, подставлять в них результаты математических вычислений и прочее и прочее и прочее.

Все правильно составленные выражения, в которые входят сырые объекты и символы, тоже приобретают текущие значения, которые определяются текущими значениями символов с учетом определений всех входящих в эти выражения функций и всех определенных для них правил преобразования, как тех, которые встроены в систему, так и (в первую очередь!!!) тех, которые определены пользователем. Вычисление текущего значения выражения называется просто **вычислением** или **эвалюацией** (`evaluation`) этого выражения. Текущим значением выражения является другое — либо то же самое!!! — выражение, причем система вычисляет значение выражение `f[x,y,...z]` следующим образом:

• вычисляется заголовок $f$ выражения;

• *поочередно* вычисляются все элементы $x, y, \ldots, z$ — кроме тех, вычисление которых ЯВНЫМ ОБРАЗОМ ЗАПРЕЩЕНО специальной командой `Hold` или ее вариантами и/или родственниками `HoldFirst`, `HoldRest`, `HoldAll`, `HoldPattern`, `HoldSequence` и т.д.;

• используются ассоциативность, дистрибутивность и коммутативность — именно в таком порядке!!! — если $f$ обладает этими свойствами.

• используются все правила и подстановки, определенные пользователем для функций, входящих в определение $x, y, \ldots, z$;

• используются все внутренние правила и подстановки, для функций, входящих в определение $x, y, \ldots, z$;

• используются все правила и подстановки, определенные пользователем для функции $f$;

• используются все внутренние правила и подстановки, для функций, входящих в определение $f$.

В некоторых случаях применение всех этих правил не упрощает выражения, тогда система оставляет его в первоначальном виде (`unevaluated`). Иными словами, текущим значением такого выражения является само это выражение.

## § 3. Многочлены

Как мы уже обсуждали, основой системы `Mathematica`, как и других систем компьютерной алгебры, являются полиномиальные вычисления. В Главе 3 мы видели, как выделить компоненты многочлена, *рассматриваемого как выражение*, при помощи команд работы с выражениями и списками `Part` и `Level`. В языке `Mathematica` для этой цели есть и специальные команды. Мы будем обсуждать их главным образом для многочленов от одной переменной, но в действительности все они работают и для

многочленов от нескольких переменных.

| | |
|---|---|
| `Variables[f]` | список переменных многочлена $f$ |
| `Exponent[f,x]` | наибольшая степень с которой $x$ входит в $f$ |
| `Coefficient[f,x,n]` | коэффициент при $x^n$ в $f$ |
| `CoefficientList[f,x]` | список коэффициентов $f$ как многочлена от $x$ |
| `PolynomialQ[f,x]` | является ли $f$ многочленом от $x$? |

Прежде всего заметим, что многочлен небольшой степени можно задать явным образом, как `a+b*x+c*x^2+d*x^3`. Однако, для многочленов высокой степени нужны какие-то более интеллигентные способы. Простейший состоит, конечно, в том, чтобы использовать итератор, например так:

```
poly[n_,a_][x_]:=Sum[a[i]*x^i,{i,0,n}]
```

С другой стороны, конечно, как всегда в `Mathematica` существуют десятки других способов добиться того же результата, скажем,

```
poly[n_,a_][x_]:=Dot[Table[a[i],{i,0,n}],Table[x^i,{i,0,n}]]
```

Теперь вычисление `poly[5,a][x]` даст

```
a[0]+x*a[1]+x^2a[2]+x^2a[3]+x^4a[4]+x^5a[5].
```

Для тестирования программ полезно выбирать многочлены со случайными коэффициентами, в этом случае мы просто напишем `Random[]` вместо `a[i]`.

Команда `Variables[f]` говорит нам, что именно в данный момент рассматривается в качестве входящих в $f$ независимых переменных. Например, когда мы пишем `f=a+b*v+c*x^2` мы, скорее всего, имеем в виду, что $f$ является многочленом от переменной $x$ с коэффициентами $a, b, c$. Однако система думает несколько иначе. Вычисление `Variables[f]` дает `{a,b,c,x}` (конечно, если переменным $a, b, c$ до этого не присваивалось никаких значений!) Тоньше всего использование этой команды с точными вещественными или комплексными числами. Поскольку для чисел `E`, `Pi` и т.д. тест `NumericQ[x]` дает значение `True`, они *не рассматриваются* в качестве независимых переменных!! Таким образом, вычисление `Variables[E+Pi]` дает {}. В этом месте мы предлагаем тому, кто раньше не задумывался над работой систем компьютерной алгебры, сделать глубокий вдох и немного помедитировать. С точки зрения всех точных вычислений `E` и `Pi` **являются** независимыми полиномиальными переменными. В то же время они **рассматриваются** как числа, а не как независимые переменные. Подобный моральный релятивизм и постоянная смена фокуса типичны скорее для чистой математики, чем для программирования!

Команда `Exponent[f,x]` вызванная с двумя аргументами возвращает **степень** многочлена $f$ по переменной $x$. Например, для `f=a+b*v+c*x^2` вычисление `Exponent[f,x]` даст 2, а вычисление `Exponent[f,a]` даст 1. В действительности, в полном формате команда `Exponent` вызывается с *тремя* аргументами как `Exponent[f,x,g]`, где $g$ — функция, которую следует применить к списку всех показателей степени, с которыми $x$ входит в $f$. По умолчанию параметр $g$ полагается равным `Max`, но, конечно, в качестве

$g$ можно взять много других функций, скажем `Min`, `List`, `Sequence`, `Plus`, etc., etc. Ну, например, при вычислениях с **многочленами Лорана**

$$f = a_{-l}x^{-l} + \ldots + a_{-1}x^{-1} + a_0 + a_1x^1 + \ldots + a_nx^n \in K[x, x^{-1}]$$

естественно одновременно рассматривать **порядок**, т.е. наименьшее $m$ такое, что $a_m \neq 0$ и **степень**, т.е. наибольшее $n$ такое, что $a_n \neq 0$. Это делается при помощи команды `Exponent[f,x,{Min[##],Max[##]}&]`. В этой команде мы вызываем функцию, сопоставляющую списку *пару*, состоящую из его наименьшего и наибольшего элементов в формате чистой функции. Обращение к элементам списка в формате последовательности аргументов `##` необходимо потому, что мы априори не знаем, сколько элементов в этом списке. Последовательность `##` вызывает их все. Точно так же `Exponent[f,x,List]` породит *список* всех показателей степени, с которыми $x$ входит в $f$ и т.д.

**Упражнение.** Что мы получим при вычислении

`Exponent[0,x,{Min[##],Max[##]}&]`?

**Ответ.** Да уж, конечно, правильный ответ {`Infinity,-Infinity`}. Как и принято в математике, функция `Min[]`, вызванная вообще без аргументов, дает `Infinity`, а функция `Max[]`, естественно, `-Infinity`.

Функции `Coefficient` и `CoefficientList` служат тому, чтобы породить индивидуальный коэффициент и список коэффициентов многочлена $f$, соответственно. Их использование абсолютно понятно исходя из названий. А именно, `Coefficient[f,x,i]` возвращает коэффициент с которым $x^i$ входит в $f$. С другой стороны, `CoefficientList[f,x]` возвращает список всех коэффициентов от степени 0 до `Exponent[f,x]`.

Может показаться, что функции `Coefficient` и `CoefficientList` дублируют друг друга. Действительно, функция `CoefficientList` легко выражается через `Coefficient`, например, так

```
clist[f_,x_]:=Table[Coefficient[f,x,i],{i,0,Exponent[f,x]}]
```

Естественно, функция `clist` всегда возвращает тот же результат, что и `CoefficientList`. Но для случайных многочленов степени несколько тысяч она делает это раз в 100 медленнее!! В этом можно убедиться, например, определив случайный многочлен `rapo=Sum[Random[]*x^i,{i,0,1000}];` — кстати, почему мы написали здесь `=`, а не `:=`?? — и вычислив

`TrueQ[clist[rapo,x]==CoefficientList[rapo,x]]`.

Точно так же, если $f$ проходит тест `PolynomialQ[f,x]`, т.е. если $f$ действительно является многочленом от $x$, то `Coefficient[f,x,i]` даст тот же результат, что `CoefficientList[f,x][[i+1]]`, хотя, конечно, снова гораздо быстрее! Обратите внимание на сдвиг номера, список коэффициентов начинается с **нулевой** степени, однако соответствующий коэффициент является **первой** компонентой списка!!! Таким образом *для многочленов* функции `Coefficient` и `CoefficientList` действительно выражаются друг через друга.

Однако в действительности функция `Coefficient` является значительно более общей! Для того, чтобы к выражению $f$ можно было применить функцию `CoefficientList`, оно действительно должно быть многочленом. В частности, оно не может содержать отрицательных или дробных степеней $x$, не говоря уже о трансцендентных функциях от $x$. Попытка вычислить что-нибудь в духе `CoefficientList[x^-3+a+b*x+c*x^2,x]` моментально приведет к сообщению об ошибке `General: a+1/x^3+b*x+c*x^2 is not a polynomial`. В то же время, функция `Coefficient` знает, что делать в подобных случаях. Вычисление

```
Map[Coefficient[a*x^-3+b*Sqrt[x]+c*Sin[x],x,#]&, {-3,1/2}]
```

дает `{a,b}`, как и задумано.

## § 4. Немедленное и отложенное присваивание: = Set versus := SetDelayed

> Символ нуля — 0. А символ ноля — О.
>
> Даниил Хармс, Нуль и ноль

Вот две основные команды, которые придают значения переменным: **немедленное присваивание** = `Set` и **отложенное присваивание** := `SetDelayed`. Уяснение различия между ними имеет ключевое значение для того, кто никогда раньше не занимался программированием!!! Собственно именно с этого места программирование начинает отличаться от математики!

| | | |
|---|---|---|
| `x=y` | `Set[x,y]` | немедленное присваивание |
| `x:=y` | `SetDelayed[x,y]` | отложенное присваивание |

Еще раз объясним основное различие между двумя этими командами, или как принято говорить в программировании, **операторами присваивания**:

• Когда мы выполняем *непосредственное* присваивание `lhs=rhs`, его правая часть `rhs` вычисляется один раз в тот момент, когда мы выполняем присваивание;

• Когда мы выполняем *отложенное* присваивание `lhs:=rhs`, его правая часть `rhs` заново вычисляется каждый раз, когда мы обращаемся к значению его левой части `lhs`.

В качестве иллюстрации этого рассмотрим следующий диалог:

```
In[1]:=x=1; y=x; x=2; y
Out[1]=1
In[2]:=x=1; y:=x; x=2; y
Out[2]=2
In[3]:=ClearAll[x,y]
```

Хорошо видно, что в первом случае мы придаем переменной $y$ текущее значение переменной $x$ *на момент присваивания*. В то же время в случае *отложенного* присваивания значение переменной $y$ вычисляется исходя из текущего значения переменной $x$ *на момент обращения*.

Еще раз поясним два момента, фигурирующих в этом вычислении. Во первых, в нем использована точка с запятой. Вместо этого мы могли бы просто напечатать присвоения `x=1`, `y=x`, `x=2` в трех отдельных строках, а `x` — в четвертой строке. Однако нас интересует только значение $x$ в конце вычисления, поэтому мы помещаем все четыре шага в одну строку, разделяя их точками с запятой. В этом случае все эти вычисления выполняются, но на экран выводится только результат последнего из них.

| | |
|---|---|
| `,` | разделение аргументов функции или компонент списка |
| `;` | разделение команд или частей аргумента |

Постановка `;` в конце инпута интерпретируется как выполнение **пустой операции**. Тем самым, предшествующая операция производится, но ее результат не выводится.

Во-вторых, в конце вычисления использована команда `ClearAll`. Она нужна, чтобы убрать определения и текущие значения переменных $x$ и $y$, иначе на протяжении всей сессии программа будет считать, что $x = y = 2$. Мы уже упоминали, что давать всем используемым переменным такие имена, как `x` или `y` — это не слишком хорошая идея.

Критическое различие между `Set` и `SetDelayed` особенно хорошо видно в следующем диалоге:

```
In[4]:=xxx=Random[]; Table[xxx,{5}]
Out[4]={0.325926,0.325926,0.325926,0.325926,0.325926}
In[5]:=yyy:=Random[]; Table[yyy,{5}]
Out[5]={0.116906,0.216847,0.821755,0.754748,0.485576}
```

В первом случае функция `Random` вызывается ОДИН РАЗ, а во втором — КАЖДЫЙ РАЗ ЗАНОВО при каждом обращении к ней. Обозвать переменные `xxx` и `yyy` — это еще один грязный трюк, позволяющий избежать использования присвоенных `xxx` и `yyy` значений в дальнейших вычислениях, в следующий раз мы просто назовем переменные `funx` и `funy` или `xbis`, `ybis`, или еще как-нибудь в таком духе.

• Не всякое присваивание фактически удается осуществить: написав `2=x`, Вы получите сообщение об ошибке `cannot assign to raw object 2`.

**Предостережение.** Обычно сопротивление системы удается преодолеть и в таких случаях, но не пытайтесь это сделать!!! Вот, что по этому поводу говорится в The Mathematica Book: `Raw` *should be* **only** *used under very special circumstances*. Присваивание 2 значения 3 является одним из самых надежных способов уронить ядро `Mathematica` и потерять все несохраненные во время сессии результаты.

• `Mathematica` помнит только последнее по времени присваивание, которое удалось осуществить: `x=y; x=z` даст нам $x$ равное $z$, а не $y$.

• Некоторым системным переменным (таким как `$RecursionLimit` и т.д.) нельзя присваивать произвольные значения. Не думайте об этом, пока не столкнетесь с сообщением `recursion limit exceeded`, после этого положите `$RecursionLimit=Infinity`.

## § 5. Секунды, такты и шаги

Допустим, нам требуется переименовать переменные $x$ и $y$. Мы не можем просто присвоить `x=y`, так как при этом старое значение $x$ будет утрачено. По той же причине мы не можем и присвоить `y=x`. Основные традиционные способы сделать это таковы[52]:

◦ Ввести новую переменную $z$ и выполнить следующую последовательность присваиваний: `z=x; x=y; y=z`.

◦ Рассмотреть $(x, y)$ как вектор и выполнить следующую последовательность присваиваний: `x=x+y; y=x-y; x=x-y`. С точки зрения линейной алгебры речь здесь идет, конечно же, о том, что перестановка двух координат вектора может быть реализована как цепочка операций, каждая из которых меняет не более одной координаты. Или, как нас учат/мы учим на первом курсе, элементарное преобразование третьего типа можно представить в виде произведения элементарных преобразований первого и второго типов.

Однако первый из этих способов требует создания дополнительной переменной, а второй — некоторой выдумки (или знания линейной алгебры!) Система `Mathematica` сконструирована так, чтобы *пользователь* мог обходиться без того и другого[53]. А именно, *естественный* способ выполнить эту операцию; более того, *единственно правильный* способ выполнить эту операцию; тот, который задуман для подобных ситуаций творцами системы; тот, под который она оптимизирована; состоит в том, чтобы выполнить оба присваивания **одновременно**:

◦ Выполнить присваивание списком `{x,y}={y,x}`.

Этот пример должен подготовить начинающего к мысли, что измеряемое шагами течение *внутреннего времени* в алгоритме отличается не только от математического отсутствия времени, не только от измеряемого секундами физического течения времени, но и от хода тактового времени при фактическом исполнении этого алгоритма процессором!! Фактически, разумеется, при выполнении присваивания `{x,y}={y,x}` присваивание `x=y` выполняется на несколько тактов раньше, чем присваивание `y=x`. Однако с точки зрения алгоритма они происходят **за один шаг**, поэтому в присваивании `y=x` используется старое, а не новое значение $x$. В то же время при нашей неудач-

---

[52] А.Шень, Программирование. Теоремы и задачи. — М., МЦНМО, 2004, с.1–296; стр.8.

[53] Разумеется, на уровне фактического вычисления компьютер при этом создает новые переменные, но это тот уровень вычисления, который спрятан в код на языке `C`, т.е. *сознательно* скрыт не только от пользователя, но и от от того уровня, на котором происходит мышление самой системы!

ной попытке написать `x=y; y=x` присваивания происходят последовательно не только с точки зрения процессора, но и с точки зрения алгоритма, т.е. не только в разных тактах, но и на разных шагах!!

В качестве метафоры подобного тройного контроля времени секундами, тактами и шагами можно привести съемку фильма. Измерение времени в шагах отвечает внутреннему времени сюжета. Измерение времени в тактах — съемочным дням. Ну и, наконец, процесс производства фильма занимает год или два — а то и десять лет!! — физического времени. Ясно, что хотя эти три способа контроля времени подчинены некоторым общим ограничениям, устанавливающим связи между ними, это соответствие вовсе не является прямым. В то же время, трудно не заметить, что связь съемочных дней с астрономическими гораздо более тесная, чем с сюжетными!! Нас как зрителей не интересует, снят более ранний по сюжету эпизод до или после последующего. Нам важно только как смонтирован фильм!!!

В этой книге в большинстве ситуаций нас интересует исключительно развитие сюжета — и в дальнейшем мы как правило обсуждаем только аспекты, связанные с шагами рассматриваемых алгоритмов. Тем не менее в некоторых случаях — ну, например, когда мы сами снимаем фильм — нас может начать волновать и прямой контроль времени, в том числе и физического. Такой контроль оказывается важным с практической точки зрения в тот момент, когда эффективность используемых алгоритмов становится недостаточной для того, чтобы фактически осуществить интересующее нас вычисление. Следующие команды дают глобальные временные параметры сессии:

| | |
|---|---|
| `AbsoluteTime[]` | время в секундах начиная с 1 января 1900 года |
| `SessionTime[]` | время в секундах с начала текущей сессии |
| `TimeUsed[]` | время работы CPU в секундах с начала сессии |

Вот например, как выглядят параметры сессии, сопровождавшей написание этого параграфа:

In[6]:=`{AbsoluteTime[],SessionTime[],TimeUsed[]}`

Out `{3.3062861285750416 10^9, 47872.8077168, 14.479}`

Обращает на себя внимание, что работа `CPU`, направленная собственно на вычисление, представляет собой ничтожно малую долю общей продолжительности сессии!!

Однако в большинстве случаев нас значительно больше интересует контроль времени индивидуального вычисления.

| | |
|---|---|
| `Timing[x]` | время работы CPU при вычислении $x$ |
| `AbsoluteTiming[x]` | полное время вычисления $x$ |
| `$TimeUnit` | минимальный интервал фиксируемый `Timing` |
| `TimeConstrained[x,t]` | вычисление $x$ с контролем времени $t$ секунд |

Команда `Timing[x]` вычисляет $x$ и возвращает время работы `CPU`, необходимое для этого вычисления и полученное значение $x$. Вот некоторые особенности ее использования:

• В некоторых случаях нас интересует только затраченное время, но не получившийся результат. В таких случаях нужно вычислять `Timing[x;]` — ну в самом деле, не спрашивать же `Timing[x];` — кстати, почему?

• Время показываемое `Timing[x]` отвечает *приращению* `TimeUsed[]` после вычисления $x$. Это можно сформулировать и в обратную сторону: `TimeUsed[]` суммирует `Timing[x]` для *всех* вычисленных на протяжении сессии выражений $x$.

• Команда `Timing[x]` контролирует только время работы `CPU` необходимое собственно для вычисления $x$. Время, затраченное на форматирование и вывод результата, общение с другими программами и пр. не учитывается. Если Вы хотите измерить все время необходимое для вычисления и представления результата, используйте `AbsoluteTiming[x]`.

• К порождаемым командами `Timing` и `AbsoluteTiming` результатам не следует относиться слишком серьезно. Конечно, разницу между 0.002 секундами, 2 секундами и 2000 секундами в большинстве случаев трудно игнорировать. В то же время разница между 2 и 3 секундами обычно не является показательной. Дело в том, что это время зависит не только от качества алгоритма, но и от используемой компьютерной системы, объема доступной памяти, особенностей операционной системы, эффективности компиляции используемого фрагмента кода, других процессов, которые одновременно исполняются на том же компьютере, и — не в последнюю очередь!!! — от того, в какой момент сессии это вычисление производится.

• Обычно необходимое для вычисления время может быть определено лишь с точностью до некоторой константы, описывающей степень самоконтроля компьютера, на котором производится вычисление. Наименьшее время, которое может быть *эффективно* замерено, называется `$TimeUnit`. Для большинства бытовых компьютеров, работающих под `Windows`, это время равно 1 миллисекунде, т.е. 1/1000 секунды. Таким образом, в этом случае относительно вычисления, про которое команда `Timing` заявляет, что оно требует 0 секунд работы `CPU`, можно с уверенностью утверждать лишь то, что оно занимает меньше 1 миллисекунды. Многие машины, работающие под `UNIX`, пользуются более мелким зерном.

Команда `TimeConstrained[x,t]` пытается вычислить $x$ за $t$ секунд и прерывает вычисление, если расчет не укладывается в отведенное время. Ясно, что в диалоговом режиме эта команда практически бесполезна, так как мы и так можем прервать вычисление в любое время. Эта команда становится однако вполне осмысленной в составе длинной программы, в которой мы предлагаем попробовать несколько различных подходов к вычислению одного и того же выражения.

## § 6. Модификация значения переменной

В отличие от математиков, программисты вопринимают переменную действительно как *переменную*, которая в разные моменты времени принимает разные значения. Команды $n = n + 1$ и $n = n - 1$ встречаются в традиционном программировании настолько часто, что им удобно присвоить специальные названия и обозначения.

| | | |
|---|---|---|
| `n++` | `Increment[n]` | увеличить $n$ на 1 *после* вычисления |
| `n--` | `Decrement[n]` | уменьшить $n$ на 1 *после* вычисления |
| `++n` | `PreIncrement[n]` | увеличить $n$ на 1 *перед* вычислением |
| `--n` | `PreDecrement[n]` | уменьшить $n$ на 1 *перед* вычислением |

Различие между `Increment[n]` и `Decrement[n]` совершенно понятно. После выполнения первой из этих команд значение $n$ увеличивается на 1, а после выполнения второй — уменьшается на 1. Различие между `Increment` и `PreIncrement` гораздо тоньше, по крайней мере с точки зрения того, кто никогда не практиковался в процедурном программировании. В отличие от предыдущей эта оппозиция не является математической, а целиком лежит в области `flow of calculation`. Ее можно объяснить так. Все эти команды берут **старое значение** переменной $n$ и перерабатывают его в **новое значение**, равное $n + 1$ или $n - 1$. При этом `++` как в постпозиции, так и в препозиции увеличивает его на 1, так что как `n=1; n++; n`, так и `n=1; ++n; n` дадут 2 — проверьте это!!! Так в чем же тогда разница между ними? Дело в том, что мы спросили про значение $n$ *слишком поздно*, чтобы заметить эту разницу!! До выполнения `++` используется старое значение, после выполнения — новое. Существует ровно один момент, обращение к $n$ *во время* инкрементации, который покажет эту разницу. А именно, в этот момент `n++` использует старое значение, в то время как `++n` — уже новое. Тем самым, `n=1; n++` даст 1, в то время как `n=1; ++n` даст 2. Подробнее за работой этих команд можно проследить по следующему диалогу, в котором мы предлагаем выдать 10 инкрементаций или декрементаций (вызов итератора в формате `{10}` без имени итератора интерпретируется просто как количество повторений), после чего смотрим на получившееся значение $i$. Мы видим, что 10 инкрементаций увеличивают значение на 10, а 10 декрементаций уменьшают его на 10, но вот текущие значения в процессе выполнения этих процедур различны:

```
 In[7]:={i=1; Table[i++,{10}], i}
Out[7]={{1,2,3,4,5,6,7,8,9,10},11}
 In[8]:={i=1; Table[++i,{10}], i}
Out[8]={{2,3,4,5,6,7,8,9,10,11},11}
 In[9]:={i=1; Table[i--,{10}], i}
Out[9]={{1,0,-1,-2,-3,-4,-5,-6,-7,-8},-9}
In[10]:={i=1; Table[--i,{10}], i}
```

Out[10]=`{{0,-1,-2,-3,-4,-5,-6,-7,-8,-9},-9}`

Обратите внимание на формат этих команд. Мы вызываем **список** из минипрограммы `i=1; Table[i++,{10}]` и `i`. Почему мы не могли спросить просто `i=1; Table[i++,{10}]; i`, без всякого списка? Потому что в этом случае мы увидели бы только окончательные значения $i$ после выполнения вычисления, а не сам ход вычислений. Еще раз обратите внимание на тот факт, что на уровне окончательных значений никакой разницы между `i++` и `++i` нет!!!

Примерно так же как инкремент и декремент используются и другие команды модификации значений переменной:

| | | |
|---|---|---|
| `x+=d` | `AddTo[x,d]` | прибавить $d$ к $x$ |
| `x-=d` | `SubtractFrom[x,y]` | вычесть $d$ из $x$ |
| `x*=c` | `TimesBy[x,c]` | умножить $x$ на $c$ |
| `x/=c` | `DivideBy[x,c]` | разделить $x$ на $c$ |

Таким образом, `x+=d` является просто сокращением для `x=x+d`, а `x-=d` — сокращением для `x=x-d`. Точно так же `x*=c` является сокращением для `x=x*c`, а `x/=d` — сокращением для `x=x/c`. Эти сокращения бывают удобны в тех случаях, когда переменная $x$ длинное имя, которое мы не хотим перепечатывать. Во всех остальных случаях мы предпочитаем пользоваться явными присваиваниями `x=x+d`, `x=x-d`, `x=x*c` и `x=x/c`. Между модификацией значения переменной и присваиванием имеется существенное различие:

• Чтобы значение переменной можно было модифицировать, она должна уже иметь значение. Попытка выполнить что-нибудь в духе `x+=y` не приведет к успеху, если $x$ до этого не присваивалось значения. Результатом такой попытки будет сообщение `AddTo: a is not a variable with a value, so its value cannot be changed`. С другой стороны, `x=y; x+=z` даст `y+z`. То же самое относится, конечно, и ко всем остальным операциям модификации значений, в том числе к инкременту и декременту.

Вот, например, программа для вычисления $1^m + \ldots + n^m$, выполненная с помощью команды `+=`:

```
summa[m_,n_]:=Block[{i,sum=0}, Do[sum+=i^m,{i,1,n}]; Return[sum]]
```

Использование остальных идиом нам уже известно. Однако в Махабхарате утверждается, что Брахма так выражался по этому поводу: повтореба мать учебы, глупец не поймет и после 10000 повторений, а мудрому достаточно всего 2500 повторений. Поэтому давайте еще раз резюмируем все лингвистические моменты:

◦ Бланк _ показывает, что переменные $m$ и $n$ здесь являются **пустышками** (`dummy variables` = фиктивные или немые переменные), которые можно заменить на любое другое выражение.

◦ Отложенное присваивание `:=` показывает, что правая часть является не окончательным значением, которое мы хотим присвоить левой части, а **программой** для вычисления этого значения. Эта программа будет запущена в тот момент, когда мы вместо пустышек подставим в левую часть настоящие численные значения.

◦ Блок `Block` служит для локализации значений переменных `i,sum`. В действительности локализация $i$ излишня, так как `Do` сама локализует свой итератор — но мы можем этого не знать!! С другой стороны, локализация `sum` тоже не обязательна, если мы не будем называть наших переменных `sum`— а мы этого делать не будем, так как никогда не используем в глобальном контексте имен, лишь в одной позиции отличающихся от внутренних имен. Но локализация промежуточных переменных/значений заведомо не может ничему помешать, а вот отсутствие локализации там, где она необходима, может привести к серьезным проблемам. Девиз серьезного пользователя: sicher ist sicher, sicuro é sicuro, better safe, than sorry.

◦ `Block` вызывается с двумя аргументами, списком локальных переменных и собственно программой, отдельные команды этой программы разделяются точкой с запятой. Использование запятой вместо точки с запятой для разделения команд является **грубой** синтаксической ошибкой.

◦ Команда `Do` имеет *два* аргумента, вначале один, описывающий упражнение, которое эта команда намеревается проделывать, а потом второй, который задает имя итератора, с которым команда собирается это проделать, его начальное и конечное значение. Как всегда, по умолчанию шаг итерации равен 1. Кроме того, по умолчанию и начальное значение итератора равно 1, так что в принципе мы могли бы задать второй аргумент в виде {`i,n`}, но мы считаем, что это bad form (= дурной стиль).

◦ Команда `Return[sum]` служит для возвращения текущего значения `sum` из блока на глобальный уровень.

**Упражнение.** Напишите программу, вычисляющую сумму $m$-х степеней первых $n$ *нечетных* натуральных чисел.

## § 7. Немедленная и отложенная подстановка: `->` `Rule` versus `:>` `RuleDelayed`

Следует отдавать себе отчет, что присваивание $x = 3$ имеет глобальный характер, и сохраняет силу на протяжении всей сессии, до тех пор, пока $x$ не было присвоено новое значение или пока значение $x$ не было модифицировано. Поэтому если Вы хотите просто посмотреть, чему равно значение $f(x)$ какой-то функции при $x = 3$, значительно удобнее пользоваться не присваиванием, а другой конструкцией, **подстановкой**. Подстановка оформляется посредством `Rule` и `Replace` или их вариантов. В простейшем виде для вычисления значения $f(3)$ можно ввести, например, текст, `f[x] /. x->3`, который с тем же успехом, что `x=3; f[x]` вычисляет $f(3)$,

но при этом обладает тем неоспоримым преимуществом, что не присваивает $x$ никакого перманентного значения.

**Хороший стиль программирования.** Старайтесь вообще избегать явного присваивания значений тем переменным, которые не имеют уникальных имен. Вместо этого пользуйтесь подстановками.

| | | |
|---|---|---|
| `x->y` | `Rule[x,y]` | немедленная подстановка |
| `x:>y` | `RuleDelayed[x,y]` | отложенная подстановка |

Различие между `Rule` и `RuleDelayed` в смысле `flow of calculation` *в точности* такое же, как между `Set` и `SetDelayed`. А именно,

• Когда мы выполняем *непосредственную* подстановку `lhs->rhs`, ее правая часть `rhs` ВЫЧИСЛЯЕТСЯ ОДИН РАЗ в тот момент, когда мы задаем подстановку;

• Когда мы выполняем *отложенную* подстановку `lhs:>rhs`, ее правая часть `rhs` ЗАНОВО ВЫЧИСЛЯЕТСЯ КАЖДЫЙ РАЗ, когда мы обращаемся к значению ее левой части `lhs`.

Приведем пример наглядно демонстрирующий **драматическое** различие между `Rule` и `RuleDelayed` с точки зрения `flow of calculation`. Определим список `xxx` посредством `Table[x,{10}]`. Тогда

`xxx===`{`x,x,x,x,x,x,x,x,x,x`}.

Посмотрим на два следующих диалога:

```
In[11]:=n=1; xxx /. x->n++
Out[11]={1,1,1,1,1,1,1,1,1,1}
In[12]:=n=1; xxx /. x:>n++
Out[12]={1,2,3,4,5,6,7,8,9,10}
```

В нашем инпуте использована команда инкремент `n++ Increment`, которая увеличивает значение $n$ на 1 **после** использования текущего значения $n$ в вычислении. Мы видим, что при применении `Rule` значение $n$ вычисляется ОДИН РАЗ, в тот момент, когда ЗАДАНО правило `x->n++`. С другой стороны при применении `RuleDelayed` значение $n$ вычисляется КАЖДЫЙ РАЗ ЗАНОВО при каждом обращении к $x$.

**Упражнение.** Ответьте, не включая компьютер, что получится, если заменить в этих программах инкремент на преинкремент `++n` который увеличивает значение $n$ на 1 **перед** использованием текущего значения $n$; а также на декремент `n--` или предекремент `--n`, которые уменьшают значение $n$ на 1 **после** использования или, соответственно, **перед** использованием текущего значения $n$ в вычислении.

• Правила `lhs->rhs` и `lhs:>rhs` применяются при помощи описанной в следующем параграфе команды `Replace` и ее усиленных вариантов `/. ReplaceAll`, `//. ReplaceRepeated` и т.д.

• Описанное в § 4 немедленное присваивание `lhs=rhs` трактуется системой как приказ на протяжении всей сессии применять немедленную подстановку `lhs->rhs` всюду, где ее можно применить, в то время как варианты команды `Replace` применяет ее только в явно указанных местах.

• Точно так же описанное в предыдущем параграфе отложенное присваивание `lhs:=rhs` трактуется системой как приказ на протяжении всей сессии применять отложенную подстановку `lhs:>rhs` всюду, где ее можно применить, в то время как команда `Replace` применяет ее только в явно указанных местах.

• Модификация опций большинства встроенных функций осуществляется путем включения подстановки `Option->Choice` в тело функции. Чтобы изменить значения нескольких опций, нужно написать

```
Option1->Choice1, Option2->Choice2, ...
```

Вот типичнейший пример изменения опции посредством команды `Rule`. По умолчанию команда `Factor` ищет разложение многочлена $f$ над $\mathbb{Z}$, но задав опцию `Extension->{a,b,c}` можно предложить ей раскладывать многочлен над $\mathbb{Z}[a,b,c]$:

◦ `Factor[x^4+1]` предлагает разложить этот многочлен над $\mathbb{Z}$, но над $\mathbb{Z}$ он неприводим, так что мы снова увидим `1+x^4`;

◦ `Factor[x^4+1,Extension->{I}]` предлагает разложить тот же многочлен над кольцом $\mathbb{Z}[i]$ целых гауссовых чисел. В этом случае мы получим ответ `(-I+x^2)(I+x^2)`;

◦ `Factor[x^4+1,Extension->{Sqrt[2]}]` предлагает разложить тот же многочлен над кольцом $\mathbb{Z}[\sqrt{2}]$. В этом случае мы получим совершенно другое разложение, $-(-1+\sqrt{2}-x^2)(1+\sqrt{2}+x^2)$.

• Многие команды выдают ответ в формате `x->c`. Например, вычисляя при помощи `Solve[a*x+b==0,x]` решение линейного уравнения $ax+b=0$ мы получим ответ в виде `{{x->-b/a}}`.

• В выражении можно одновременно произвести несколько подстановок. Самый удобный формат для этого состоит в том, чтобы задавать подстановки как *список*. Это можно сделать заранее. Например, вычисление

```
In[13]:=rules={x->a,y->b}; f[x+y] /.  rules
```

даст нам `f[a+b]` но при этом, как всегда, сами $x$ и $y$ не получают никаких постоянных значений. С другой стороны, список подстановок сохранится и чтобы вычислить `g[a,b]` нам достаточно будет напечатать `g[x,y] /. rules`.

• В подстановках и отложенных подстановках можно использовать бланки и паттерны для обозначения того, что подстановка должна осуществляться для *произвольных* объектов из некоторого домена. Подстановка, применямая не к индивидуальным объектам, а к паттернам — которые могут в свою очередь заменяться на произвольные объекты!!! — называется **правилом преобразования** (`transformation rule`).

Приведем игрушечный пример применения правил преобразования. Например, Mathematica автоматически не пользуется формулой для тангенса суммы для вычисления чего-нибудь наподобие Tan[a+Pi/3]. Применив к этой функции TrigExpand с последующим Simplify, мы получим что-нибудь в духе (√3*Cos[a]+Sin[a])/(Cos[a]-√3*Sin[a]). Но мы хотим *явно* увидеть, что получается в результате применения к Tan[a+Pi/3] именно формулы для тангенса суммы. Это позволяет сделать следующая конструкция:

```
In[14]:=Hold[Tan[a+Pi/3]] /.
        Tan[x_+y_]->(Tan[x]+Tan[y])/(1-Tan[x]*Tan[y])
```

Функция Hold здесь использована, чтобы удержать Mathematica от некоторых лишних в данном случае преобразований. Последующее применение ReleaseHold[%] даст желаемое представление:

```
Out[14]=(√3+Tan[a])/(1-√3*Tan[a])
```

Обратите внимание на синтаксис lhs[x_,y_]->rhs[x,y]:

• левая часть правила преобразования имеет формат lhs[x_,y_], бланки при переменных указывают, что это **пустышки**, называемые также **немыми** или **фиктивными переменные** (dummy variables), вместо которых можно подставить ЧТО УГОДНО;

• правая часть правила преобразования имеет формат rhs[x,y], но при этом бланки при немых переменных не ставятся!!!

• Если бы мы хотели применять это правило и в дальнейшем, мы бы, скорее всего, задали его как **отложенное правило преобразования**, в формате lhs[x_,y_]:>rhs[x,y].

• Как мы узнаем в Главе 5, применение *отложенных* правил преобразования имеет то преимущество, что они могут принимать условия, введенные посредством оператора /; Condition. Языковая тонкость здесь состоит в том, что если мы описываем правило преобразования *само по себе*, оно имеет формат

```
lhs[x_,y_] :>rhs[x,y] /; conditions
```

Однако если мы хотим применить это правило при помощи оператора замены /. ReplaceAll, то условный оператор вносится в левую часть правила преобразования:

```
(lhs[x_,y_] /; conditions):>rhs[x,y]
```

Mathematica автоматически не упрощает таких выражений, как $\ln(e\pi)$. Допустим, мы хотим провести вычисление в духе школьной математики, использующее формулу $\ln(xy) = \ln(x) + \ln(y)$ **только** в том случае, когда $x, y > 0$. Для этого правило нужно задавать в виде

```
In[15]:=Log[E*Pi]/.(Log[x_*y_] /; N[x]>0&&N[y]>0):>Log[x]+Log[y]
```

даст `1+Log[Pi]`.

То, что мы сейчас увидели, представляет собой первую **крошечную** демонстрацию тех огромных возможностей, которые появятся у читателя в тот момент, когда он овладеет **функциональным программированием**, которое позволит ему определять функции и правила преобразования для них, а также управлять применением функций к разным фрагментам выражений и правил преобразования на разных этапах вычисления.

## § 8. Просто замены и сугубые замены: `Replace`, `/. ReplaceAll`, `//. ReplaceRepeated`

> Как аллилуйи делятся на аллилуйи просто и сугубые аллилуйи.
>
> Венедикт Ерофеев, Из записных книжек

Применение замен и правил подстановки к выражениям производится при помощи **операторов замены** таких как `Replace` и его усиленные варианты `ReplaceAll`, `ReplaceRepeated` и `ReplaceList`. По сути, к той же категории относится и оператор `ReplacePart`, но он синтаксически отличается от остальных перечисленных операторов и мы обсуждаем его в части 3 в контексте работы со списками.

| | | |
|---|---|---|
| | `Replace[x,y->z,{n}]` | однократная замена на $n$-м уровне |
| `/.` | `ReplaceAll[x,y->z]` | однократная замена на всех уровнях |
| `//.` | `ReplaceRepeated[x,y->z]` | многократная замена |
| | `ReplaceList[x,y->z]` | список результатов всех замен |

Самой деликатной из этих команд является `Replace`, которая позволяет явно контролировать, на каких уровнях выражения производятся замены. При этом:

• Формат команды `Replace` таков: `Replace[expr,rule,level]` имеет два аргумента, выражение и правило подстановки/преобразования и один параметр — спецификацию уровня.

• К каждому подвыражению выражения `expr`, находящемуся на уровне действия `Replace`, описанному `level`, к которому можно применить подстановку `rule`, она применяется один раз.

• По умолчанию `Replace[x,y->z]` вызванная без указания уровня применяется только к уровню 0, т.е. ко всему выражению $x$!!!

• Уровень задается при помощи обычных спецификаций: `{n}` для $n$-го уровня, `n` для $n$-го и всех более высоких уровней, `{m,n}` для всех уровней между $m$-м и $n$-м, `Infinity` для *всех* уровней, `{-1}` для листьев и т.д. Все эти спецификации подробно обсуждаются в Главе 1.

• Второй аргумент команды `Replace` может быть **списком** правил подстановки/преобразования. В этом случае к каждому подвыражению выражения `expr` команда `Replace` применяет только одну подстановку, из этого списка, а именно, первую из тех, которые можно применить.

Таким образом,

```
Replace[f[x],x->a]===f[x],
Replace[f[x],x->a,1]===f[a],
Replace[f[x],{x->a,f[x]->f[b]}]===f[b].
```

Вот еще характерный пример того, как работает команда `Replace`. Полагая `glist=NestList[List,g,2]` или, *что то же самое*, `glist={g,{g},{{g}}}`, и применяя к этому списку командой `Replace` замену `g->a`, а потом список замен `{g->a,{g}->a,{{g}}->a}` на уровнях 0,1,2,3, мы получим следующие ответы:

```
In[16]:=Map[Replace[glist,g->a,{#}]&,Range[0,3]]
Out[16]={{g,{g},{{g}}},{a,{g},{{g}}},{g,{a},{{g}}},{g,{g},{{a}}}}
In[17]:=Map[Replace[glist,{g->a,{g}->a,{{g}}->a},{#}]&,Range[0,3]]
Out[17]={{g,{g},{{g}}},{a,a,a},{g,{a},{a}},{g,{g},{{a}}}}
```

В реальных вычислениях особенно часто применяются две следующие сильные версии команды `Replace`:

• Команда `ReplaceAll[expr,rule]` представляет собой просто *сокращение* для `Replace[expr,rule,Infinity]`. Иными словами, `ReplaceAll` состоит в ОДНОКРАТНОМ применении правила `rule` КО ВСЕМ УРОВНЯМ выражения `expr`. Этот оператор используется настолько часто, что для него имеется специальная операторная запись `expr /. rule`.

• С другой стороны, `ReplaceRepeated[expr,rule]`, или, в сокращенной операторной записи, `expr //. rule` предписывает не просто применять подстановку `rule` ко всем уровням выражения `expr`, но и проделывать это МНОГОКРАТНО, до тех пор, пока выражение продолжает меняться при очередной подстановке!!!

Следующий диалог иллюстрирует *драматическое* различие между тем, как исполняются `ReplaceAll` и `ReplaceRepeated`, снова лежащее в области `flow of calculation`. Команда `/.` предписывает применить подстановку, заменяющую $\ln(xy)$ на $\ln(x)+\ln(y)$, ко всем частям выражения, но ТОЛЬКО ОДИН РАЗ. С другой стороны, `//.` предписывает проделывать эту подстановку `quantum satis` (= до полного удовлетворения):

```
In[18]:=Log[a*b*c*d] /. Log[x_*y_]->Log[x]+Log[y]
Out[18]=Log[a]+Log[b*c*d]
In[19]:=Log[a*b*c*d] //. Log[x_*y_]->Log[x]+Log[y]
Out[19]=Log[a]+Log[b]+Log[c]+Log[d]
```

Обсудим, наконец, чем команда `Replace` отличается от `ReplaceList`. Команда `Replace` применяет ПЕРВОЕ из тех правил `rule`, которое можно применить, ПЕРВЫМ из тех способов, которыми его можно применить. В то же время команда `ReplaceList` дает список результатов, которые получаются при ВСЕВОЗМОЖНЫХ применениях ВСЕХ правил.

То, что происходит со списком подстановок, как раз совершенно ясно. Скажем, `Replace[x,{x->a,x->b,x->c}]` даст `a`, в то же время результатом вычисления `ReplaceList[x,{x->a,x->b,x->c}]` будет список `{a,b,c}`.

Если бы использование `ReplaceList` сводилось к этому, она была бы фактически излишня, так как она сразу выражалась бы через `Replace` и `Map`. Однако в действительности роль `ReplaceList` гораздо шире. То, что делает ее *по-настоящему* полезной, это не возможность одновременно увидеть результаты применения нескольких правил подстановки/преобразования, а возможность посмотреть на **все** способы применения ОДНОГО И ТОГО ЖЕ ПРАВИЛА ПРЕОБРАЗОВАНИЯ к данному выражению. Например,

```
Replace[a+b+c,x_+y_->x*y]
```

дает единственный результат `a(b+c)`. Команда `ReplaceList` делает *значительно* больше:

```
In[20]:=ReplaceList[a+b+c,x_+y_->x*y]
Out[20]={a(b+c),b(a+c),(a+b)c,(a+b)c,b(a+c),a(b+c)}
```

Обратите внимание, что при этом рассматриваются не только группировки соседних членов, а вообще все группировки, в том порядке, как ядро их замечает! Ясно, что во всех сколь-нибудь сложных случаях подобный учет **всех** возможностей преобразования выражений при помощи какого-то правила находится за пределами комбинаторных способностей обычного человека.

## § 9. ЧИСТКА: `Unset`, `Clear`, `ClearAll`, `Remove`

Если мы все же не пользуемся подстановками и заменами, а придаем переменным постоянные значения посредством операторов присваивания, то время от времени эти переменные следует чистить. Это производится при помощи следующих команд.

| | |
|---|---|
| `Unset[x]` | очистить значения $x$ |
| `Clear[x,y,z]` | очистить значения и *определения* $x, y, z$ |
| `ClearAll[x,y,z]` | кроме того, очистить *атрибуты и опции* $x, y, z$ |
| `Remove[x,y,z]` | смешать $x, y, z$ с грязью и забыть их имена |

Эти команды расположены здесь в порядке возрастающей силы:

• `Unset[x]`, или, в сокращенной записи `x=.` убирает все *значения* и правила для переменной $x$. В отличие от остальных команд `Unset` применяется к одному аргументу и отменяет немедленное присваивание $x = u$.

• `Clear[x,y,z]`, убирает не только все значения переменных $x, y, z$, но и все их *определения*. Иными словами, `Clear` отменяет не только немедленное присваивание $x = u$, но и *отложенное* присваивание $x := u$.

• `ClearAll[x,y,z]`, убирает не только *все* определения и значения переменных $x, y, z$, но и их *атрибуты*, дефолтные *опции* и связанные с ними сообщения. Тем не менее, `Mathematica` помнит, что переменные с такими именами использовались во время сессии.

• `Remove[x,y,z]`, убирает не только все определения и значения, атрибуты, и пр., но и сами *имена* переменных $x, y, z$. Эти переменные перестают

распознаваться программой — до тех пор, конечно, пока Вы их снова не создадите! Иными словами, программа считает, что эти переменные во время сессии вообще не использовались. Различие `ClearAll` и `Remove` иллюстрируется следующими двумя диалогами:

```
In[21]:=x=3; ClearAll[x]; NameQ["x"]
Out[21]=True
In[22]:=x=3; Remove[x]; NameQ["x"]
Out[22]=False
```

Иногда, однако, мы можем *слегка* перестараться в нашей любви к очистке — не вздумайте печатать что-нибудь в духе `ClearAll["*"]`!!! Наряду с тем, что Вы *намеревались* очистить, это вычистит **заодно** и все ответы на информационные запросы, сообщения об ошибках и пр. На такие случаи в системе предусмотрена двуступенчатая защита объектов от чисток, очисток и зачисток:

• Команды `Clear`, `ClearAll` и `Remove` не действуют на объекты, которые имеют атрибут `Protected`.

• На случай, если Вам вздумается положить $x\_ + y\_ := x * y$, или что-нибудь в таком духе, все встроенные объекты защищены атрибутом `Protected`, который *запрещает* менять их определения и значения.

• В некоторых случаях, скажем, если Вы пишете программу, которой пользуются несколько человек, возникает желание защитить некоторые из определенных Вами объектов и/или их значений от *случайного* переопределения. Это делается при помощи команды `Protect[object]`, которая устанавливает на объект `object` атрибут `Protected` и делает его *неуязвимым* (`immune`) по отношению к зачисткам.

• Защиту большинства внутренних и системных объектов можно снять командой `Unprotect[object]`. После этого с объектом можно поступать так же, как с любым другим символом. Например, примитивисты и сюрреалисты оценят возможности, которые предоставляют тексты наподобие следующего: `Unprotect[Log]; Log[10]=2`. После этого система будет *искренне* верить, что `Log[10]==2`. Однако обычно подобная модификация внутренних объектов не есть слишком хорошая идея (`is not such a smart idea`).

• Имеется небольшое число фундаментальных внутренних и системных объектов, любая модификация определений и/или значений которых привела бы к полному краху системы. Эти объекты защищены атрибутом `Locked`, который запрещает изменение *каких-либо* атрибутов этих объектов и, тем самым, *в частности*, удаление атрибута `Protected`. Это значит, что *внутри* системы определения этих объектов вообще не могут быть изменены.

## § 10. Создание локальных и уникальных переменных

На случай, если Вы используете систему `Mathematica` для программирования в процедурном духе, в языке `Mathematica` предусмотрено два основных способа локализации переменных или их значений, `Module` и `Block`, отвечающие тому, что традиционно называлось 'подпрограммой', 'процедурой', 'субрутиной' и т.д. и заключалось в текст `begin ... end`. Внутри этих конструкций Вы можете делать *все что угодно*!! Как и все, что помещено в новый контекст посредством `Begin[context]` и `End[]`, то, что вы не вернули в глобальный контекст, не окажет никакого влияния на все, что происходит в глобальном контексте. Кроме того, если Вы любите использовать буковку $x$ для обозначения своих переменных, Вы можете не устраивать никаких модулей и блоков, а непосредственно пользоваться командой `Unique`, которая каждый раз создает *новый* символ формата `x$nnn` или `xnnn`. Если Вы не меняли настроек системы, используемый при этом **серийный номер** `nnn` представляет собой ту же системную переменную `$ModuleNumber`, которая считает, сколько раз во время сессии использовались конструкции локализации переменных и при каждом таком использовании увеличивается на 1. Наконец, имеется *полупрозрачная* конструкция `With`, при помощи которой можно придавать *локальные значения* глобальным переменным.

| | |
|---|---|
| `Module[{x,y},body]` | лексическая локализация переменных $x, y$ |
| `Block[{x,y},body]` | динамическая локализация переменных $x, y$ |
| `With[{x=a,y=b},body]` | локальная фиксация значений $x, y$ |
| `Unique[x]` | создание уникальной переменной `x$nnn` |
| `Unique["x"]` | создание уникальной переменной `xnnn` |

Для всех обычных юзеров, не являющихся профессиональными программистами, действие `Module` и `Block` практически полностью эквивалентно. По причинам исторического, психологического и философского (нежелание увеличивать без нужды количество сущностей) характера авторы предпочитают пользоваться командой `Block`, которая не создает новых переменных, но оставляет внутри себя все значения переменных, перечисленных как локальные.

**Комментарий: локальные переменные и локальные значения.** На уровне имплементации разница между командами `Module` и `Block` может быть объяснена следующим образом. Команда `Module`, подобно большинству традиционных языков программирования, локализует переменные на *лексическом* уровне. Эта команда предписывает рассматривать `body` как часть системного *кода* и появляющиеся в этом коде *переменные* $x, y, \ldots$, рассматриваются как локальные. С другой стороны, команда `Block` вообще не смотрит на то, как выглядит `body` и какие переменные туда входят. Вместо этого она трактует все *значения* переменных $x, y, \ldots$, используемые во время вычисления `body`, как локальные. Команда `Module` несомненно удобнее, если Вы пишете *такие* программы, в которых субрутины вызываются сотни раз и которые нуждаются в отладке, так что Вам интересно знать, чем текущее значение переменной `x$273` отличается от текущего значения переменной `x$237`. Для тех, кто (как мы) пишет программы, проверяемые строка за строкой, необходимости в этом никогда не возникает.

Поясним некоторые важнейшие моменты, связанные с использованием конструкций, включающих `Module` или `Block`:

• Второй аргумент команд `Module` или `Block` может быть ПРОГРАММОЙ. В этом случае отдельные команды, входящие в эту программу должны разделяться **точкой с запятой**: `Block[{x,y},body1;body2;body3]`. Использование здесь запятой вместо точки с запятой является **грубой** синтаксической ошибкой.

• В первом аргументе этих команд локальным переменным можно явно присваивать НАЧАЛЬНЫЕ ЗНАЧЕНИЯ:

`Module[{x=a,y=b},body]` и `Block[{x=a,y=b},body]`.

Эффективно `Block[{x=a,y=b},body]` является просто более удобным способом ввести следующий текст: `Block[{x,y},x=a;y=b;body]`.

• Использование команд `Module` или `Block` **абсолютно** необходимо в тех случаях, когда в тексте встречаются ЦИКЛЫ, ИТЕРАЦИИ, УСЛОВНЫЕ ОПЕРАТОРЫ И ДРУГИЕ УПРАВЛЯЮЩИЕ СТРУКТУРЫ, в процессе своей работы меняющие значения входящих в них переменных!!

Поясним это на примере. Некоторые итерационные команды (такие как `Table`, `Sum`, `Product`, `Do`) автоматически локализуют значения своих итераторов. Однако это относится далеко не ко всем конструкциям!!! Допустим, Вы включили в текст определения функции **цикл**

`For[i=1,test,i++,body],`

предписывающий проделывать операцию `body` до тех пор, пока выполняется критерий `test`, после каждого выполнения этой операции увеличивая $i$ на 1. Знаете ли Вы, чему будет равно $i$ в конце выполнения этого цикла? Ну, скажем, в таком совсем простеньком случае

```
m:=1; For[i=1,Prime[i]<10000,i++,If[OddQ[i],m=m*i,m=m/i]].
```

Ну так вот, в конце этого цикла $i = 1230$ и если теперь Вы снова используете $i$ в качестве итератора без указания начального значения, итерация начнется с $i = 1230$, а вовсе не с $i = 1$, как Вы, скорее всего, имели в виду!!! В начале 1990-х годов мы *систематически* допускали трудно отслеживаемые ошибки такого рода. Против этого есть два лекарства:

◦ явное задание начальных значений итераторов при **каждом** обращении к ним;

◦ явная локализация **всех** итераторов и/или их значений при помощи `Module` или `Block`;

а еще гораздо лучше и то и другое!!!

Проиллюстрируем особенности использования конструкции, включающей команду `Block` на реальном примере. Следующий инпут определяет функцию `twin[n]`, вычисляющую первые $n$ пар близнецов $(p, p + 2)$, где $p$ и $p + 2$ оба простые.

```
In[23]:=twins[n_]:=
        Block[{i,p,m=0,x={}},
```

```
For[i=1,m<n,i++,p=Prime[i];
If[PrimeQ[p+2],AppendTo[x,{p,p+2}];m=m+1]];
Return[x]]
```

Здесь происходит следующее: мы локализуем значения четырех переменных $i, p, m$ и $x$. При этом $i$ является просто итератором, $p$ представляет собой $i$-е простое число, $x$ — промежуточный список близнецов, порожденный в процессе работы цикла, $m$ — длину этого списка. *Формально* переменные $p$, $m$ излишни, так как мы могли обращаться к ним как к `Prime[i]` и `Length[x]`, однако явное введение этих переменных не только сокращает и делает более понятным текст, но и ускоряет работу программы. Более того, если бы мы писали этот текст для его использования другими людьми, мы, скорее всего, дали бы переменным $p, m, x$ говорящие имена типа `prime`, `length`, `list`!! Если бы мы не произвели присваивания `p=Prime[i]`, то внутри условного оператора три раза происходило бы обращение к функции `Prime`. Точно так же явное увеличение значения $m$ на 1 работает быстрее, чем вычисление длины списка $x$.

Заключенная в `Block` программа работает следующим образом. В начале ее работы $x$ — пустой список, а $m = 0$. Для каждого $i$ начиная с 1, до тех пор пока текущая длина $m$ построенного списка пар $x$ меньше $n$, мы делаем следующее: присваиваем $p$ значение, равное $i$-му простому числу, после чего в случае, когда тест `PrimeQ[p+2]` дает значение `True` — т.е. число $p + 2$ тоже является простым — мы добавляем к $x$ пару $(p, p + 2)$ увеличивая при этом $m$ на 1. После этого текущее значение $i$ увеличивается на 1 и начинается новая итерация. Как только $m$ стало равным $n$, цикл прерывается и текущее значение $x$ возвращается на глобальный уровень.

Начинающий должен обратить внимание на несколько **ключевых** синтаксических моментов:

◦ Явное провозглашение итератора $i$ локальной переменной!!!

◦ Использование команды `Return` для возвращения значения локальной переменной $x$ на глобальный уровень.

◦ Разделение цикла `For` и возврата значения `Return` **точкой с запятой**;

◦ Внутри цикла разделение присваивания `p=Prime[i]` и условного оператора `If` **точкой с запятой**;

◦ Внутри условного оператора `If` разделение инкрементации $x$ и инкрементации $m$ **точкой с запятой**;

**Предостережение.** Условный оператор `If` вызывается с *двумя* аргументами, вместо обычных *трех* или *четырех*. Мы можем позволить себе делать это только потому, что эта функция написана нами в чисто учебных целях, и мы собираемся использовать ее для небольших значений $n$, когда `PrimeQ` гарантировано дает достоверный ответ `True` или `False`. В следующем параграфе мы объясняем, почему во всех серьезных случаях для оператора `If` мало и *трех* аргументов.

Конструкцию `With[{x=a,y=b},body]` можно воспринимать как совместное ослабление конструкций

`Module[{x=a,y=b},body]` и `Block[{x=a,y=b},body]`.

Однако между ними имеется существенное различие, состоящее в том, что с случае конструкций `Module` и `Block` значения $a$ и $b$ рассматриваются как **начальные значения** переменных $x$ и $y$ в вычислении `body`. В дальнейшем в процессе вычисления `body` эти значения могут меняться на каждом шаге и в вычислении окончательного ответа участвуют уже какие-то совсем другие значения $x$ и $y$. В то же время в случае конструкции `With` переменные $x$ и $y$ становятся внутри нее **локальными константами** $a$ и $b$ и именно они используются на всех шагах вычислении `body`. Эта конструкция не создает новых переменных, а локализует *только* их значения:

• Все *имена* переменных используемых конструкцией `With` являются глобальными. Однако присвоенные внутри нее *значения* этих переменных не влияют на их глобальные значения. Например, `x=1; With[{x=2},y=x^2]; x+y` даст Вам 5, а не 6.

• Если имена переменных входят в конфликт, то в соответствии с общим принципом `last in, first out` всегда используется самое глубокое значение переменной. Например, `With[{x=a},With[{x=b},f[x]]]` даст Вам `f[b]`.

• Конструкция `With` полупрозрачна: значения тех переменных, имена которых не входят в конфликт, продолжают оставаться видимыми локально. Например, `With[{x=a},With[{y=b},f[x,y]]]` даст Вам `f[a,b]`.

В то же время, команда `Unique` действует *прямо противоположным образом*. Она каждый раз явно порождает уникальную переменную. Эту команду можно использовать следующим образом:

• `Unique[]` создает новый символ с именем формата `$ModuleNumber`,

• `Unique[x]` создаст новый символ с именем формата `x$ModuleNumber`,

• `Unique["x"]` создаст новый символ с именем формата `xModuleNumber`.

Следующие два диалога хорошо иллюстрируют, что происходит при применении этой команды.

```
In[24]:=Table[Unique[x],{i,1,10}]
Out[24]={x$85,x$86,x$87,x$88,x$89,x$90,x$91,x$92,x$93,x$94}
In[25]:=Table[Unique["x"],{i,1,10}]
Out[25]={x71,x72,x73,x74,x75,x76,x77,x78,x79,x80}
```

В этот момент Вы должны всерьез задуматься, действительно ли Вам нужно несколько сотен **индивидуальных переменных**, имена которых начинаются с $x$ или, все же в этом случае удобнее рассматривать совокупность этих переменных как массив значений функции $x$ или как компоненты списка $x$. Более того, и с точки зрения компьютера вычисления с функцией или списком значительно эффективнее!!!

## § 11. Равенство и тождество: == Equal versus === Same

Равенство есть продукт непротивления сторон.

Математик

Сейчас мы объясним еще два значения, на которые символ = расщепляется в Computer Science, а именно, **равенство** == и **тождество** ===.

| | | |
|---|---|---|
| x==y | Equal[x,y] | равно |
| x!=y | Unequal[x,y] | не равно |
| x===y | SameQ[x,y] | тождественно |
| x=!=y | UnsameQ[x,y] | нетождественно |

В первом приближении разницу в использовании этих операторов можно объяснить так: сравнивая *значения* двух выражений, нужно писать lhs==rhs в то же время сравнивая *форму* этих выражений, следует писать lhs===rhs.

В частности, *во всех командах*, связанных с символьным или численным решением уравнений, **уравнение** $f(x) = g(x)$ должно записываться как f[x]==g[x]. Запись уравнения в форме f[x]=g[x] представляет собой **грубейшую ошибку**!!! Эта ошибка настолько очевидна, что даже обсуждать ее смешно, ведь запись с помощью Set представляет собой *присваивание*, т.е. попытку изменить текущее значение $f(x)$ на $g(x)$. В большинстве случаев, если $f(x)$ имеет защищенный заголовок или когда $f(x)$ уже является сырым объектом, система просто **не даст** нам записать уравнение в такой форме!! Например, попытка написать a*x^2+b*x+c=0, приведет к сообщению об ошибке: Tag Plus in c+b*x+a*x^2 is Protected.

Однако хотя различие между == и === менее очевидно, смешение этих операторов приводит к столь же драматическим последствиям. Вот, например, как Mathematica решает квадратные уравнения.

In[26]:=Solve[a*x^2+b*x+c==0,x]

$$\text{Out[26]=}\left\{\left\{\texttt{x->}\frac{\texttt{-b-}\sqrt{\texttt{b\^{}2-4*a*c}}}{\texttt{2*a}}\right\},\ \left\{\texttt{x->}\frac{\texttt{-b+}\sqrt{\texttt{b\^{}2-4*a*c}}}{\texttt{2*a}}\right\}\right\}$$

Однако попытка предложить системе вычислить

```
Solve[a*x^2+b*x+c===0,x]
```

приведет к ответу {}. Понятно, почему? В первый раз мы спросили, при каких $x$ *значение* $ax^2 + bx + c$ равно 0. А во второй раз — при каких $x$ внутренняя форма *выражения* a*x^2+b*x+c **текстуально** совпадает с внутренней формой *выражения* 0. Да ни при каких!!!

**Упражнение.** Как Вы думаете, почему команда решения уравнения имеет формат Solve[f[x]==g[x],x]? При чем тут $x$?

**Решение.** Ну это тоже понятно. Это мы знаем, что $x$ обозначает неизвестную, но система об этом не должна догадываться. Ну в самом деле,

не предлагать же ей *по умолчанию* решать все уравнения относительно $x$!! Предложение вычислить `Solve[a*x^2+b*x+c==0,a]` приведет к ответу `a=(-c-bx)/x^2`.

Так как решение уравнений встречается довольно часто, еще раз подчеркнем, что предикаты `Equal[x,y]` и `SameQ[x,y]` следует *тщательнейшим образом* различать:

• Справедливость равенства `x==y` представляет собой *математический* вопрос, функция `Equal[x,y]` возвращает значение `True`, если $x$ и $y$ имеют равные значения и значение `False`, если $x$ и $y$ имеют различные значения. Во многих случаях `Mathematica` на основе имеющихся у нее данных НЕ В СОСТОЯНИИ РЕШИТЬ, равны $x$ и $y$ или нет!!! В этом случае она просто оставляет выражение `Equal[x,y]` **неэвалюированным**.

• В то же время справедливость тождества `x===y` представляет собой *алгоритмический* вопрос. Функция `SameQ[x,y]` возвращает значение `True`, только если $x$ и $y$ реализованы одинаковым образом — ИМЕЮТ ОДИНАКОВУЮ ВНУТРЕННЮЮ ФОРМУ — и `False` во всех остальных случаях. Это значит, что `Mathematica` **всегда** может дать ответ на вопрос `SameQ[x,y]`.

• Эти ответы ОЧЕНЬ ЧАСТО ОТЛИЧАЮТСЯ — две равные вещи не обязательно тождественны — как учили нас на уроках диалектического материализма, ОДИНАКОВОЕ ОДИНАКОВОМУ РОЗНЬ! Чтобы сознательно использовать эти операторы, полезно постараться понять логику системы. Например, `1==1.` принимает значение `True` — ведь это *равные* числа — но, конечно, `1===1.` принимает значение `False` — хотя бы потому, что `1.` не является целым числом!

• **Единственный** случай, когда тождество `x===y` не требует текстуального совпадения внутренних представлений — это равенство *приближенных* вещественных или комплексных чисел. В этом случае система может решить, что два числа тождественны, если их разность меньше, чем точность каждого из них.

Еще одной типичной ошибкой, которую мы *систематически* совершали в начале 1990-х годов, было неосторожное использование `==` в условных операторах. По наивности мы исходили из того, что оператор `If[x==y,u,v]` должен возвращать либо $u$, либо $v$. Но это совершенно не так, в тех случаях, когда система не может решить, действительно ли `x==y`, она просто оставляет это выражение неэвалюированным, что часто приводит к бесконечной рекурсии. После нескольких крайне неприятных эпизодов (в те годы ни в интерфейсе `Mathematica` ни в `Mac OS` не было предусмотрено прерывание вычисления и мы были вынуждены *физически* выключать компьютер, теряя все предшествующие вычисления!!!) мы сделали три важных вывода:

• BETTER SAVE THAN SORRY;

• в сомнительных случаях условные операторы необходимо задавать в виде `If[x==y,u,v,w]`, **явно** предписывающим значение условного выраже-

ния и в том случае, когда система не в состоянии решить, равны $x$ и $y$ или, все-таки, нет!

• Часто система не может решить, равны ли два выражения, *по чисто формальным причинам*. Дело в том, что во многих ситуациях она не производит автоматических упрощений. Это значит, что СРАВНИВАЯ ДВА ВЫРАЖЕНИЯ, ВСЕГДА ПОЛЕЗНО ПРИМЕНИТЬ К НИМ ВСЕ ВОЗМОЖНЫЕ[54] УПРОЩЕНИЯ!!! Если Вы хотите проверить, равны ли $x$ и $y$, Ваши шансы на получение правильного ответа резко возрастут, если вместо `x==y` Вы напечатаете `Simplify[x]==Simplify[y]`, `FullSimplify[x]==FullSimplify[y]`, или что-нибудь соответствующее ситуации: `FunctionExpand`, `LogicalExpand`, `Refine`, `Reduce` и тому подобное.

• Другой народный способ состоит в том, чтобы задавать условные операторы с использованием `===` но тогда, конечно, *тем более* вначале нужно либо упрощать сравниваемые выражения,

```
If[Simplify[x]===Simplify[y],u,v],
```

либо принудительно приводить их к одинаковой форме посредством `Factor`, `Expand` и т.д.

**Предостережение.** Как и в языке `C++` выражения `==`, `!=`, `===`, `=!=`, `<=`, `>=` и т.д. представляют собой **командные слова**, а внутри слова пробел не ставится. Выражение, содержащее пробел внутри командного слова считается синтаксически неправильным и приводит к сообщению об ошибке.

**Предостережение.** `Mathematica` исходит из того, что реляционные операторы транзитивны!!! Выражение `x!=y!=z` оценивается как истинное только если $x, y, z$ попарно различны! В частности, `1!=0!=1`, истинное с точки зрения математических традиций, будет оценено как `False`.

## § 12. Решение алгебраических уравнений

Как мы знаем, с точки зрения `Mathematica` выражение `f[x]==g[x]` представляет собой уравнение. В системе имплементировано несколько функций для точного (когда это возможно) и приближенного (во всех остальных случаях) решения уравнений. Вот наиболее употребительные из них.

| | |
|---|---|
| `Roots[f==g,x]` | решить *полиномиальное* уравнение $f(x) = g(x)$ |
| `Solve[f==g,x]` | решить уравнение $f(x) = g(x)$ относительно $x$ |
| `Reduce[f==g,x]` | упростить уравнение $f(x) = g(x)$ |

Команды `Roots`, `Solve` и `Reduce` служат для *точного* решения уравнений. В то же время команды `NRoots`, `NSolve` и `FindRoot` служат для *приближенного* вычисления корней. Поясним различие между ними.

• `Roots[f==g,x]` является **самой слабой** командой, которая умеет решать *только* полиномиальные уравнения в школьном духе. В случае неал-

[54] всевозможные!

гебраического уравнения или когда она не может найти корни алгебраического уравнения в духе школьной математики, она просто оставляет их неэвалюированными:

```
In[27]:=Roots[x^5-x-1==0,x]
Out[27]=x==Root[-1-#1+#1^5&,1] || x==Root[-1-#1+#1^5&,2] ||
        x==Root[-1-#1+#1^5&,3] || x==Root[-1-#1+#1^5&,4] ||
        x==Root[-1-#1+#1^5&,5]
```

• По умолчанию `Roots` решает кубические уравнения и уравнения четвертой степени в радикалах, иными словами опции `Cubics` и `Quartics` поставлены на `True`. Для того, чтобы решать уравнения совсем уж в школьном духе, нужно включить в тело функции замены `Cubics->False` и `Quartics->False`.

Однако так как команда `Roots` использует только полиномиальные алгоритмы, в некоторых случаях (например при вычислениях в конечных кольцах, полях алгебраических чисел и т.д.) она оказывается весьма полезной. Перечислим некоторые команды, связанные с командой `Roots`.

| | |
|---|---|
| `Roots[f==g,x]` | корни полиномиального уравнения $f(x) = g(x)$ |
| `Root[f,n]` | $n$-й корень полиномиального уравнения $f = 0$ |
| `RootReduce[y]` | попытка свести выражение к одному `Root` |
| `ToRadicals[y]` | попытка выразить все `Root` в радикалах |
| `ToRules[y]` | преобразовать формат вывода `Roots` в формат `Solve` |

Смысл этих команд ясен сам по себе. Обратите внимание, что в команде `Root` нет указания на то, относительно какой переменной следует решать уравнение!! Это значит, что первый аргумент должен вводиться не как многочлен, а в формате **чистой функции**, скажем, вместо `x^2+ax+b` следует печатать `(#^2-a#+b)&`.

В качестве универсальной команды, дающей — в тех случаях, когда это возможно в элементарных терминах!! — *точные* решения алгебраических уравнений или систем уравнений, но *пытающаяся* решить вообще любые числовые уравнения *имеющие конечное число корней* в том числе и трансцендентные, начинающий должен рассматривать `Solve[f==g,x]`. Кроме чисто полиномиальных алгоритмов решения `Solve` использует другие инструменты, такие как обратные функции (опция `InverseFunctions`). Однако при этом в случае бесконечного количества решений `Solve` как правило ограничивается одной ветвью обратной функции и указывает *какие-то* решения, а не все решения.

• Команду `Solve` можно применять для решения системы уравнений относительно нескольких переменных. При этом как уравнения, так и неизвестные, относительно которых эти уравнения решаются, можно задавать в форме списка: `Solve[{f1==g1,...,fm==gm},{x1,...,xn}]`. Уравнения можно связывать также логическими связками, скажем, `f1==g1 && f2==g2`. В этом случае ответ тоже будет записан как список замен `{{x->a,y->b},...}`.

• Команда `Solve` порождает только общие решения и игнорирует решения, возникающие при специальных значениях параметров. В таких случаях нужно применять команду `Reduce`. Кроме того, под влиянием команды `Solve` система приходит в такой восторг, найдя какие-то решения, что перестает искать остальные решения, в таких случаях также нужно использовать команду `Reduce`!! Например, попытавшись вычислить `Solve[Cos[x]==2,x]`, Вы получите следующий эфемерный ответ

```
{{x->-ArcCos[2]},{x->ArcCos[2]}}
```

вместе с сообщением о *возможной* ошибке

```
Solve: Inverse functions are being used by Solve,
                    so some solutions may not be found
```

и советом `use Reduce for complete solution information.`

• С большим отрывом самой общей и самой мощной командой для решения уравнения $f(x) = g(x)$ является `Reduce[f==g,x]`. Дело в том, что команда `Reduce[expr,x]` пытается упростить *любое* выражение `expr` от $x$, содержащее равенства, неравенства, включения, логические связки и кванторы. В случае, когда это выражения состоит только из уравнений и неравенств, команда `Reduce` пытается их решить, но делает это гораздо более изощренным образом, чем просто подстановка $x$ в одну из ветвей обратной функции. Таким образом, в отличие от команды `Solve`, она может указать не какие-то, а **все** решения уравнения, даже если этих решений бесконечно много. Вот два типичных примера использования `Reduce` — напоминаем, что в действительности `Reduce` умеет значительно больше, в частности, решает неравенства и логические формулы!!!

◦ Вычисление `Reduce[Cos[x]==2,x]` дает правильный ответ

```
C[1]∈Integers&&(x==-ArcCos[2]+2*Pi*C[1]||x==ArcCos[2]+2*Pi*C[1])
```

Обратите внимание, что $C_1$ здесь — произвольная целочисленная постоянная. В других контекстах `Reduce` может порождать несколько таких постоянных, принадлежащих различным доменам.

◦ В следующем вычислении мы пытаемся найти *целочисленные* решения системы уравнений $x = y + 1$, $x^2 = y^3 + 1$:

```
Reduce[x^2==y^3+1&&x==y+1,{x,y},Integers]
```

Она находит три таких решения:

```
x==0&&y==-1 || x==1&&y==0 || x==3&&y==2
```

Следующие команды являются приближенными вариантами обсуждавшихся раньше точных команд.

| | |
|---|---|
| `NRoots[f==g,x]` | численное решение *полиномиального* уравнения |
| `NSolve[f==g,x]` | численное решение уравнения $f(x) = g(x)$ |
| `FindRoot[f==g,x,x0]` | корень уравнения $f(x) = g(x)$ вблизи $x_0$ |

Мы не будем подробно обсуждать использование этих команд, так как оно ясно само по себе. Команды `NRoots` и `NSolve` являются численными аналогами команд `Roots` и `Solve`, осуществляющими *приближенное* вычисление корней полиномиального уравнения или, соответственно, произвольного уравнения, имеющего лишь конечное число корней (в случае, если существует алгоритм для их вычисления). С другой стороны команда `FindRoot[f==g,x,x0]` ищет корень уравнения $f(x) = g(x)$ начиная итеративную процедуру со значения $x_0$.

# Глава 6. ФУНКЦИИ

> Выше было указано, что функции должны определяться независимо от того, кому приписано их выполнение. Из перечисления функций можно было убедиться, что они должны определяться и независимо от того, как, каким способом они выполняются.
> Это иногда затрудняет определение отдельных случаев, так как разные функции могут выполняться совершенно одинаково. По-видимому, здесь имеется влияние одних форм на другие. Это явление может быть названо ассимиляцией способов исполнения функций.
> В полном объеме это сложное явление здесь не может быть освещено. Оно может быть рассмотрено лишь постольку, поскольку это необходимо для последующих анализов.
>
> В.Я.Пропп, Морфология волшебной сказки

Все на свете является функцией — с точки зрения компьютерной алгебры все, что делается, и все, что происходит, любое изменение значения или формы выражения, любая модификация состояния системы или способа ее работы называется **функцией**. Фактически и в предыдущих главах нам не встречалось ничего, кроме функций. В этой главе мы начинаем *систематическое* изучение этого понятия. Мы детально обсудим основные обозначения и понятия, связанные с функциями, и основные классы функций языка `Mathematica` и чуть более подробно остановимся на арифметических операциях, числовых функциях и булевых функциях (предикаты и отношения). Это подготовит нас к тому, чтобы в следующей части подступить к самой сути языка `Mathematica`: определению новых функций и функциям работы со списками/применению функций к спискам.

## § 1. Функциональная и операторная запись функций

> Для каждой функции дается: 1) краткое изложение ее сущности, 2) сокращенное определение одним словом, 3) условный знак ее. (Введение знаков позволит впоследствии сравнить построение сказок схематически). Вслед за этим следуют примеры. Примеры большей частью далеко не исчерпывают нашего материала. Они даны лишь как образцы.
>
> В.Я.Пропп, Морфология волшебной сказки

В обычном математическом словоупотреблении, по крайней мере в большинстве элементарных учебных текстов, слово *функция* используется как синоним термина *отображение*. Иными словами, считается, что функция $F$ задается *тройкой* $F = (X, Y, f)$ или, в обычных обозначениях $f : X \longrightarrow Y$. При этом $X$ является **областью определения** функции $F$, $Y$ — **областью значений**, а $f$ состоит в сопоставлении *каждому* элемента $x$ множества $X$ *единственного* элемента $y$ множества $Y$, обозначаемого $f(x)$ и называемого **образом** элемента $x$ под действием $f$. Для обозначения того, что $x$ переходит в $y$ под действием $f$, используется специальная стрелка $x \mapsto f(x)$. Запись образа $x$ под действием $f$ как $f(x)$ с явным заключением аргумента в скобки называется **функциональной**, кроме нее для образа $x$ под действием $f$ часто употребляются **операторные** записи $fx$ и $xf$, **индексная** запись $f_x$ и некоторые другие более специальные нотации, использующие особые символы, указывающие на применение функции, подчеркивающие симметрию между функцией и аргументом, etc. Например, в теории групп и теории Галуа чрезвычайно часто используется **экспоненциальная** запись $x^f$, в линейной алгебре и функциональном анализе — **копуляционная** запись $\langle f, x\rangle$ и т.д.

До Кантора, Дедекинда и Пеано считалось, что функция задается *формулой* или *законом*, но начиная с конца XIX века надолго восторжествовала чисто экстенсиональная точка зрения, согласно которой $f$ отождествляется с **графиком** $\Gamma(f) = \{(x, f(x)) \mid x \in X\}$. Как правило в математических текстах допускается вольность речи, состоящая в обозначении $F$ и $f$ одной и той же буквой: *функция* $f$. Тем не менее, математики твердо придерживаются того принципа, что область определения и область значений входят в определение функции. Иными словами, для того, чтобы *две функции* $f : X \longrightarrow Y$ и $g : U \longrightarrow V$ были равны, необходимо, чтобы $X = U$, $Y = V$ и $f(x) = g(x)$ для всех $x \in X$. Мы не обсуждаем здесь абсурдную методистскую терминологию, при которой областью значений функции называется ее **образ** $f(X) = \{f(x) \mid x \in X\}$. Множество всех отображений из $X$ в $Y$ обычно обозначается через $\mathrm{Map}(X, Y)$ или $Y^X$.

Однако даже в самой математике в некоторых случаях удобно исключить область значений из определения функции. Получающийся при этом объект называется **семейством**. Иными словами, для того, чтобы *два семейства* $f : X \longrightarrow Y$ и $g : U \longrightarrow V$ были равны, необходимо, чтобы $X = U$ и $f(x) = g(x)$ для всех $x \in X$. В этом случае, конечно, $f(x) = g(x) \in Y \cap V$, но не утверждается, что $Y = V$. Таким образом, семейство *представляется*

функцией, но является не функцией, а *классом эквивалентности функций* относительно отношения эквивалентности, забывающего область значений. В упомянутом выше школьном понимании функция определяется именно как семейство! Для семейств в математике обычно используется индексная запись, аргументы называются **индексами**, а образ $x$ под действием $f$ обозначается через $f_x$ или $f^x$, или еще как-нибудь в таком духе. Семействами являются векторы, матрицы и много других важнейших объектов.

Компьютерная алгебра делает следующий шаг в том же направлении и исключает из понятия равенства функций не только область значений, но и область определения. Вернее, наподобие того, как это делается в школьной математике для вещественных функций, системы компьютерной алгебры считают, что у каждой функции имеется ЕСТЕСТВЕННАЯ ОБЛАСТЬ ОПРЕДЕЛЕНИЯ, а именно множество всех выражений, для которых имеют смысл **все** функции, которые входят в определение функции $f$, определения всех фигурирующих в ее определении функций, etc. Иными словами, функция понимается как *правило*, *алгоритм* или *инструкция*, которая перерабатывает ввод в вывод, а откуда именно берется ввод, не конкретизируется. Две функции считаются равными, если они *всегда* перерабатывают один и тот же вход в один и тот же выход.

**Комментарий.** Разумеется, такая установка становится возможной *только* потому, что `Computer Science` имеет дело исключительно с конечными объектами. В контексте бесконечных множеств подобная точка зрения невозможна просто по той причине, что совокупность *всех* объектов, над которыми можно произвести какую-то операцию, как правило, не является множеством. Тем самым, попытка сравнивать функции игнорируя их область определения сразу порождает словесные артефакты и языковые ошибки известные как “теоретико-множественные парадоксы”.

В языке `Mathematica` основной формой представления функций во всех случаях считается **функциональная запись**. При этом образ элемента $x$ под действием функции $f$ обозначается через `f[x]`. Однако, **кроме того**, в `InputForm` для этого образа существуют еще и две **операторных** записи, префиксная `f@x` и постфиксная `x//f`. В дальнейшем мы узнаем, что к тому же образу можно обратиться, вызывая его через конструкции **чистой/анонимной функции**. Разумеется, для случая, когда у используемой функции уже есть имя — здесь, например, это имя `"f"` — эти конструкции не только не содержат ничего нового, но и являются более громоздкими, чем описанные ранее. Однако позже мы увидим, что они дают нам инструмент поразительной силы, позволяющий ОБРАЩАТЬСЯ ПО ИМЕНИ К ЛЮБОЙ ФУНКЦИИ, НЕ ДАВАЯ ЕЙ ИМЕНИ!!! Иначе говоря, образ $f(x)$ можно ввести одним из пяти следующих *эквивалентных* способов:

| | |
|---|---|
| `f[x]` | функциональная запись $f(x)$ |
| `f @ x` | префиксная операторная запись $f(x)$ |
| `x // f` | постфиксная операторная запись $f(x)$ |
| `Function[z,f[z]][x]` | запись $f(x)$ как значения чистой функции |
| `f[#]&[x]` | запись $f(x)$ как значения анонимной функции |

Следует иметь в виду, что с *внутренней* точки зрения операторные записи функций являются чисто лингвистическими конструкциями, идиомами используемыми в качестве **сокращений** функциональной записи, которая является **единственной** внутренней формой:

```
Sin[x]===Sin@x===(x//Sin)===Function[z,Sin[z]][x]===Sin[#]&[x].
```

В этой формуле сразу бросается в глаза то, что `(x // Sin)` заключено в скобки, а `Sin @ x` — нет. Это связано со следующим:

• **Постфиксная форма** `//` имеет ВЕСЬМА НИЗКИЙ ПРИОРИТЕТ, более низкий, чем **все** арифметические и логические операции. В большинстве случаев функция, примененная при помощи `//` будет применяться ко всему выражению в целом. Например, `x+y//f` интерпретируется как `f[x+y]`, а не `x+f[y]`. Для того, чтобы получить `x+f[y]`, нужно напечатать `x+(y//f)`.

• **Префиксная форма** `@` имеет ЧРЕЗВЫЧАЙНО ВЫСОКИЙ ПРИОРИТЕТ, более высокий, чем все арифметические операции и большинство других операций. Как правило функция $f$ примененная при помощи `@` будет применяться к первому символу, а не ко всему выражению в целом. Так, например, `f@x*y` будет интерпретировано как `f[x]*y`, а вовсе не как `f[x*y]`. Для того, чтобы получить `f[x*y]`, нужно напечатать `f@(x*y)`.

Низкий приоритет постфиксной операторной формы задания функции является ее достоинством, а не недостатком. Мы, например, часто пользуемся постфиксной формой для просмотра или форматирования вывода. Дело в том, что напечатав в конце ввода `// MatrixForm`, `// TableForm`, или что-нибудь в таком духе, Вы можете `post factum` применить эти команды ко всему предшествующему вводу, без необходимости возвращаться к его началу и ставить скобки. В то же время, за исключением обозначения композиции, где она действительно чрезвычайно полезна, префиксная форма `@` используется только в некоторых очень специальных ситуациях.

**Пояснение.** Высокий приоритет `@` связан с тем, что его использование в указанном смысле вторично. Основное значение этого оператора состоит в том, чтобы служить в качестве сокращения для **композиции функций** `Composition`. А именно, функция `Composition[f,g]` представляет собой композицию функций $f \circ g$. По определению $(f \circ g)(x) = f(g(x))$ — читается *образ $x$ под действием композиции функций $g$ и $f$*, именно в **таком** порядке!!!. Полная запись значения $(f \circ g)(x)$ с использованием функции `Composition` должна выглядеть следующим образом: `Composition[f,g][x]`. Ясно, однако, что это все равно будет переработано во внутреннюю форму `f[g[x]]` и поэтому продвинутые пользователи пишут просто `f@g@x`, без всяких там скобок. С другой стороны, операция $\circ$ ассоциативна и обладает тем свойством, что композиция *одной* функции есть сама эта функция. Иными словами, `Composition` обладает атрибутами `Flat` и `OneIdentity`. В частности, это значит, что значение $f(x)$ можно записать *еще и так*: `Composition[f][x]`. Так вот, `f@x` как раз и является сокращением этой записи.

Как мы узнаем в следующих параграфах, все символы, кроме букв, цифр и $, и многие сочетания двух или трех символов интерпретируются `Mathematica` как операторы. Однако не для всех из них заданы алгоритм вычисления или трансформационные правила. Это сделано для того, чтобы пользователь мог ввести операторные сокращения для наиболее часто встречающихся определенных им функций.

## § 2. Специфика функций компьютерной алгебры

Everything is a function.
Steven Wolfram

Многие языки программирования проводят строгие **формальные** различия между функциями, операциями, операторами, командами, процедурами, типами или классами объектов и т.д. В свою очередь, скажем, операции столь же жестко подразделяются на арифметические операции, отношения, логические операции и т.д. Язык `Mathematica` переносит все подобные различия из области синтаксиса в область *интерпретации*. С лингвистической и, *тем более* с **внутренней** точки зрения любые преобразования объектов или формы их записи, любые модификации их значений или приписывания им свойств, любые группировки, любые контрольные и управляющие структуры, любые инструкции и предписания по дальнейшей работе системы, любые форматирования выражений, управление вводом и выводом, и пр., и пр., и пр. рассматриваются языком `Mathematica` как **функции** — или, *что то же самое*, как **команды**.

С принципиальной точки зрения *понятие* функции в компьютерной алгебре постижимо разумом и гораздо ближе к общематематическому пониманию, чем понятие функции в традиционных языках программирования. В то же время имеется немало прагматических моментов, которые отличают *использование* функций в языке `Mathematica`:

• Прежде всего, нужно понимать, что функции компьютерной алгебры работают не с объектами, а с выражениями. Это значит, что даже если два выражения являются просто двумя именами или двумя различными формами записи имени одного и того же математического объекта, (как, скажем, $(x+1)^2$ и $x^2+2*x+1$ в кольце многочленов), то переписывание выражения из одной формы в другую рассматривается как нетривиальная функция!!! При стандартных установках ядра такое переписывание отнюдь не происходит автоматически, а производится при помощи специальных структурных команд.

• Для вычислений чрезвычайно важным является понимание того, что одна и та же функция или команда может вести себя различным образом: дефолтные установки опций, используемые алгоритмы и т.д. могут меняться в зависимости от того, к какому типу объектов она применяется. Например, большинство числовых функций работают абсолютно по разному, в соответствии с тем, являются ли их аргументы **точными** целыми, рациональными или гауссовыми числами (с которыми производятся обычные безошибочные вычисления, основанные на модулярной арифметике и пр.); **точными** алгебраическими числами (такими как `Sqrt[2]`, `GoldenRatio`, вычисления с которыми производятся при помощи чрезвычайно тонких алгебраических алгоритмов основанных на методах теории Галуа и алгебраической геометрии), **точными** вещественными или комплексными числами (такими как `E`, `Pi`, `Cos[1]` или `2*Pi*I` — такие числа трактуются **как символы** и обрабатываются полиномиальными алгоритмами) или,

наконец, **приближенными** вещественными или комплексными числами (для которых числовые функции по умолчанию используют все прелести округления, абсолютные и относительные погрешности и пр.)

• Одна и та же функция может вызываться с разным количеством аргументов — в некоторых случаях с любым количеством аргументов или вообще без аргументов. Например, ассоциативные арифметические операции, логические связки, команды `List` и `Sequence` и многие другие функции могут вызываться с любым конечным количеством аргументов. В то же время многие другие функции (большинство числовых функций, многие предикаты и отношения, команды построения списков, структурные команды, управляющие структуры и т.д.) могут вызываться только со строго определенным количеством аргументов (обычно одним, двумя, тремя или четырьмя) строго определенных форматов и вызов их с любым другим количеством аргументов или с аргументами нештатных форматов рассматривается как синтаксическая ошибка.

• Разные аргументы функций имеют абсолютно различный статус. Простейшим из таких различий является различие между собственно аргументами, присутствие которых при любом вызове функции обязательно, и параметрами, некоторые из которых могут быть опущены. Кроме того, как мы узнаем в § 5, некоторые аргументы — в исключительных случаях, например, для команд построения графики *два–три десятка* аргументов!!! — (опции и атрибуты) и вовсе скрыты от начинающего пользователя.

• Самым мощным — но и самым трудным для понимания и грамотного использования, по крайней мере для того, кто не знаком с $\lambda$-исчислением и не программировал в `Lisp`!! — инструментом программирования в языках `Maple` и `Mathematica` являются **чистые функции** `pure functions`, у которых нет не только области определения, не только области значений, не только собственного имени, но даже аргументов (если быть совсем точным, их аргументы не имеют имен, а являются просто щелями, в которые можно засунуть что угодно!!) В чистой функции от функции остается **только** правило, по которому аргументу сопоставляется его образ. Чистые функции позволяют ссылаться по имени на функции, у которых нет обычных имен!!

## § 3. Основные классы функций языка `Mathematica`

> В этой главе мы перечислим функции действующих лиц в том порядке, в каком это диктуется самой сказкой.
>
> В.Я.Пропп, Морфология волшебной сказки

Сейчас мы просто *перечислим* основные типы функций, в соответствии с их интерпретацией, а не синтаксисом!!! Наша классификация в основном следует классификации функций в разделе `Built-in Functions` встроенной помощи, хотя мы не всегда перечисляем функции в том

же порядке. Некоторые из этих функций детально обсуждаются в следующих параграфах, в то время как некоторые другие относятся к предметной области, а не к языку. Поскольку между классами функций нет ярких языковых контрастов, некоторые функции описываются в нескольких подразделах (например, как структурные манипуляции и как числовые функции!)

• **Алгебраические и символьные вычисления** (`algebraic computation`). Это одна из самых многочисленных и важных групп функций, которые покрывают практически всю школьную математику, значительную часть вузовской программы высшей математики и некоторые более специальные темы.

◦ Арифметические операции: `Plus`, `Minus`, `Subtract`, `Times`, `Divide`, `Power`, `Sqrt`, `NonCommutativeMultiply`, etc.

◦ Итерационные операции `Sum`, `Product`, etc.

◦ Функции, связанные с многочленами: `PolynomialQ`, `Variables`, `Expand`, `Coefficient`, `CoefficientList`, `CoefficientArrays`, `Exponent`, etc.

**Предостережение.** В языке `Mathematica` такие функции, как `Coefficient`, `Variables`, а также арифметические функции в кольце многочленов называются `полиномиальными функциями`. Мы избегаем это название, так как в русском — да и в английском!!! — языке выражение *полиномиальная функция* устойчиво используется для отображения $K \longrightarrow K$, $c \mapsto f(c)$, где $f \in K[x]$ фиксированный многочлен.

◦ Арифметика многочленов: `Factor`, `FactorList`, `PolynomialQuotient`, `PolynomialRemainder`, `PolynomialGCD`, `PolynomialLCM`, `PolynomialMod`, `PolynomialReduce`, `Resultant`, `GroebnerBasis`, etc., etc., etc.

◦ Решение уравнений (`equation solving`): `Solve`, `Roots`, `Root`, `RootSum`, `ToRadicals`, `Reduce`, `RootReduce`, `Eliminate`, `SolveAlways`, `ToRules`, etc., etc.

◦ Символьное дифференцирование и интегрирование (`symbolic calculus`): `Limit`, `D`, `Dt`, `Derivative`, `Integrate`, `Series`, `O`, `DSolve`, `Residue`, `Maximize`, `Minimize`, etc., etc.

◦ Символьные интегральные преобразования (`integral transforms`): `FourierTransform`, `InverseFourierTransform`, `LaplaceTransform`, `InverseLaplaceTransform`, etc., etc., etc., etc.

• **Структурные манипуляции** (`expression manipulation`). Большая часть описанных здесь ‘функций’ не являются функциями в математическом смысле, так как они состоят либо в определении структуры выражения, либо в переписывании одной формы представления объекта в другую. Однако с точки зрения `Mathematica` это один из важнейших классов функций.

◦ Форма выражений и выделение их частей: `FullForm`, `TreeForm`, `Head`, `Part`, `Level`, `Length`, `Depth`, etc., etc.

◦ Заголовки типов: `Integer`, `Rational`, `Real`, `Complex`, `Symbol`, `String`, `List`, etc., etc.

◦ Названия доменов: `Integers`, `Reals`, `Complexes`, `Rationals`, `Algebraics`, `Primes`, `Booleans`, `Interval`, etc.

◦ Общие структурные манипуляции: `Simplify`, `FullSimplify`, `PowerExpand`, `FunctionExpand`, `Refine`, etc., etc., etc.

◦ Структурные операции с многочленами `Collect`, `Expand`, `ComplexExpand`, `Decompose`, `Factor`, `FactorTerms`, etc., etc., etc.

◦ Структурные операции с дробями: `Denominator`, `Numerator`, `Together`, `Apart`, `ExpandDenominator`, `ExpandNumerator`, `ExpandAll`, `Cancel`, etc., etc., etc.

◦ Структурные операции в кольце $\mathrm{Trig}_{\mathbb{R}}$ тригонометрических многочленов: `TrigExpand`, `TrigFactor`, `TrigFactorList`, `TrigReduce`,

◦ Изменение формата (`format transcription`). Многие такие функции имеют имена формата `SmthToSmth`, `ToSmth`, `FromSmth`. Типичными примерами таких функций являются `TrigToExp`, `ExpToTrig`, etc., etc.

• **Булевы функции** (`boolean functions`). Будевы функции принимают значения в домене `Booleans={True,False}`.

◦ Булевы константы и логические операции (связки), т.е. функции с булевыми аргументами и булевыми значениями: `True`, `False`, `Not`, `And`, `Or`, `Xor`, `Implies`, `LogicalExpand`, etc.

◦ Побитовые логические операции: `BitNot`, `BitAnd`, `BitOr`, `BitXor`.

◦ Кванторы (`quantifiers`) `ForAll`, `Exists`, `Resolve`.

◦ Предикаты (`predicates`), которые понимаются как функции **одного небулева** аргумента с булевыми значениями (разумеется, в математике обычно говорит о предикатах от *любого* числа аргументов): `AtomQ`, `EvenQ`, `IntegerQ`, `NameQ`, `NumberQ`, `NumericQ`, `OddQ`, `PrimeQ`, `StringQ`, `TrueQ`, `VectorQ`.

◦ Отношения (`relations`, `relational operators`): `MemberQ`, `Element`, `Equal`, `Unequal`, `Same`, `Unsame`, `Order`, `Greater`, `GreaterEqual`, `Less`, `LessEqual`, etc., etc.

• **Числовые функции** (`numeric functions`) предназначены как для символьных, так и для численных вычислений. Если их аргументы являются символами или **точными** числами, они всегда пытаются возвратить **точные** значения. Однако если их аргументы имеют формат приближенных вещественных или комплексных чисел, они работают как соответствующие приближенные функции.

◦ Константы `E`, `Pi`, `Degree`, `I`, `GoldenRatio`, `Infinity`, `ComplexInfinity`, `DirectedInfinity`, `Indeterminate` и случайные числа `Random`, `SeedRandom`, etc., etc., etc.

**Предостережение.** Тонкость, которая может показаться **весьма** неожиданной тем, кто не знаком с компьютерной генерацией (псевдо)случайных чисел, состоит в том, что `Random` является точной, а не приближенной функцией!!! Это сделано для того, чтобы можно было использовать *одно и то же* случайное число в нескольких различных вычислениях, например в процессе тестирования программ.

◦ Представление чисел: `DigitCount`, `IntegerDigits`, `RealDigits`, `FromDigits`, `MantissaExponent`, `ContinuedFraction`, `FromContinuedFraction`, etc.

◦ Численные функции (`numerical functions`): `Abs`, `Sign`, `Max`, `Min`, `Round`, `IntegerPart`, `Floor`, `Ceiling`, `Rationalize`, `Re`, `Im`, `Arg`, `Conjugate`.

◦ Элементарные функции: экспонента и логарифм `Exp`, `Log` все основные тригонометрические `Cos`, `Sin`, `Tan`, `Cot`, обратные тригонометрические, гиперболические и обратные гиперболические функции.

◦ Специальные функции: в ядре `Mathematica` и пакетах содержатся определения **сотен** специальных функций таких, как гипергеометрические и связанные с ними функции, функции Бесселя, ортогональные многочлены, интегральные косинус и синус, эллиптические функции и интегралы, функции Матье, etc., etc., etc., etc., etc., etc. Мы не будем даже пытаться перечислять или описывать эти функции, так как обращение к ним ясно само по себе.

◦ Комбинаторные функции: `Factorial`, `Binomial`, `Multinomial`, `Fibonacci`, `BernoulliB`, `StirlingS1`, `StirlingS2`, `PartitionsP`, `PartitionsQ`, `Signature`, etc., etc., etc., etc.

◦ Арифметика целых чисел: `FactorInteger`, `IntegerExponent`, `Mod`, `Quotient`, `PowerMod`, `Prime`, `PrimeQ`, `GCD`, `LCM`, `ExtendedGCD`, etc., etc.

◦ Теоретико-числовые функции: `EulerPhi`, `MoebiusMu`, `PrimePi`, `Divisors`, `DivisorSigma`, `JacobiSymbol`, `CarmichaelLambda`, etc., etc., etc.

• **Приближенные функции** (`approximate functions`) предназначены для численных, а не символьных вычислений. В отличие от числовых функций приближенные функции заведомо используют приближенные алгоритмы и даже для точных данных никогда не пытаются вычислить точные значения. Название многих приближенных функций начинается с `N`.

◦ Приближенные вычисления (`numerical calculation`) и численное решение уравнений (`numerical equation solving`): `N`, `Chop`, `NSolve`, `NRoots`, `FindRoot`, `NSum`, `NProduct`, `NIntegrate`, `NDSolve`, etc., etc., etc.

◦ Функции интерполяции (`interpolation`): `Fit`, `FindFit`, `InterpolationFunction`, `ListInterpolation`, `FunctionInterpolation`, etc.

◦ Функции численной оптимизации (`numerical optimization`): `NMaximize`, `NMinimize`, `FindMaximum`, `FindMinimum`, etc., etc., etc.

◦ Функции статистической обработки (`data manipulation functions`) вычисляют все обычные параметры выборок, рассматриваемые в теории вероятностей и математической статистике: `Mean`, `Median`, `Variance`, `Quantile`, `StandardDeviation`, etc., etc., etc.

◦ Численная обработка списков: `Fourier`, `InverseFourier`, `ListConvolve`, `ListCorrelate`, etc.

◦ Функции управления точностью (`numerical precision functions`). Многие из этих функций сами являются точными, но они служат для контроля ошибок приближенных вычислений и поэтому включаются в эту категорию: `Precision`, `Accuracy`, `SetPrecision`, `SetAccuracy`, `PrecisionGoal`, `AccuracyGoal`, etc., etc., etc.

• **Команды работы со списками** (`list and matrix manipulation`). Наиболее характерный для языка `Mathematica` класс функций. Мы считаем, что этот класс относится непосредственно к языку и детально обсуждаем его в дальнейшем. Кроме того, сюда относятся команды построения таблиц и массивов, операции над векторами и матрицами и другие функции линейной алгебры, а также операции над *разреженными* векторами и матрицами.

◦ Выделение частей списка `Part`, `Extract`.

◦ Вычеркивания `Take`, `Drop`, `First`, `Last`, `Most`, `Rest`, `Delete`.

◦ Вставки и замены: `Insert`, `ReplacePart`, `Append`, `AppendTo`, `Prepend`, `PrependTo`.

◦ Выборки: `Select`, `Cases`, `DeleteCases`, `Count`, `Position`.

◦ Сортировка списков: `Sort`, `Order`, `Ordering`, `OrderedQ`, `Permutations`, `Signature`, etc.

◦ Структурные манипуляции со списками `Join`, `Flatten`, `Partition`, `Split`, `Reverse`, `RotateLeft`, `RotateRight`, `PadLeft`, `PadRight`.

◦ Команды формирования списков: `Range`, `Table`, `Array`.

◦ Команды работы с разреженными списками: `SparseArray`, `Normal`, `ArrayRules`, etc.

◦ Форматирующие функции `ColumnForm`, `TableForm`, `MatrixForm`, etc., etc.

◦ Теоретико-множественные операции: `Union`, `Intersection`, `Complement`, etc.

◦ Операции над векторами: `Dot`, `Cross`, `Inner`, `Outer`, `Total`, etc.

◦ Функции линейной алгебры: `LinearSolve`, `NullSpace`, `LinearProgramming`, `LatticeReduce`, `Eigenvalues`, `Eigenvectors`, `Eigensystem` etc., etc., etc.

◦ Матричные функции: `Dot`, `Det`, `Minors`, `Tr`, `Transpose`, `MatrixPower`, `MatrixExp`, `MatrixRank`, etc., etc., etc.

• **Процедурное программирование** (`flow control`, `procedural programming`). Язык `Mathematica` полностью поддерживает все стили программирования, *среди прочего* в ней есть **все** обычные операторы традиционных языков программирования:

◦ Присваивание и чистка (`assignements`): `Set`, `SetDelayed`, `Unset`, `Clear`, `ClearAll`, `Remove`, etc.

◦ Подстановки и замены (`replacement rules and replacements`): `Rule`, `RuleDelayed`, `Replace`, `ReplaceAll`, `ReplaceRepeated`, `ReplaceList`, etc.

◦ Маркировки (`tags`): `TagSet`, `TagSetDelayed`, `TagUnset`, etc.

◦ Команды организации циклов: `Do`, `For`, `While`, etc.

◦ Управляющие структуры (`control structures`): `GoTo`, `Label`, `Continue`, `Break`, etc.

◦ Конструкции локализации переменных (`scoping constructs`): `Block`, `Module`, `With`, `Begin`, `End`, etc.

◦ Условные операторы (`conditionals`): `If`, `Which`, `Switch`, etc.

◦ Удерживающие функции (`holding functions`): `Hold`, `HoldAll`, `HoldForm`, `HoldComplete`, `HoldFirst`, `HoldRest`, `HoldPattern`, etc.

◦ Вбрасывающие функции: `Return`, `Throw`, `Catch`, `Sow`, `Reap`, etc.

• **Функциональное программирование** (`functional programming`). Значительно более характерным для языка `Mathematica` является функциональное программирование, которое приводит к более простым и более эффективным программам. Так как большинство этих конструкций являются специфическими для языка `Mathematica` и либо отсутствуют в других языках программирования, либо присутствуют там в рудиментарном виде и под другими именами, то в дальнейшем мы все их детально обсуждаем.

◦ Чистые функции (`pure functions`): `Function`, `Slot`, `SlotSequence`, etc.

◦ Паттерны (`pattern`): `Pattern`, `Blank`, `BlankSequence`, `BlankNullSequence`, `Condition`, `Count`, etc.

◦ Композиции: `Composition`, `InverseFunction`, `Identity`, `ComposeList`, etc.

◦ Итерации: `Nest`, `NestList`, `FixedPoint`, `FixedPointList`, `NestWhile`, `NestWhileList`, `Fold`, `FoldList`, etc.

◦ Применение функций к спискам: `Apply`, `Map`, `MapAt`, `MapAll`, `MapThread`, `MapIndexed`, `Scan`, etc.

◦ Протаскивание и распределение действия функций: `Thread`, `Through`, `Operate`, `Inner`, `Outer`, `Distribute`, etc.

• **Генерация и форматирование графики и звука**. Эти важнейшие функции имеют *настолько* развитую и сложную систему опций, что мы *не будем даже пытаться* описать или перечислить эти опции здесь. Кроме того, в ядре и, в особенности, в пакетах определено **громадное количество** графических элементов возникающих в геометрии, комбинаторике, теории графов и т.д.

◦ Двумерная графика: `Plot`, `ListPlot`, `ParametricPlot`, etc., etc., etc.

◦ Трехмерная графика: `Plot3D`, `ListPlot3D`, `ParametricPlot3D`, etc., etc., etc.

◦ Другие графические форматы: `ContourPlot`, `ListContourPlot`, `DensityPlot`, `ListDensityPlot`, etc., etc., etc.

◦ Графические примитивы: `Show`, `Graphics`, `Point`, `Line`, `Rectangle`, `Polygon`, `Circle`, `Disk`, etc., etc., etc.

◦ Мультипликация: `Animate`, `ShowAnimation`, `MoviePlot`, `MoviePlot3D`, etc., etc.

◦ Генерация звука: `Play`, `ListPlay`, etc., etc.

Кроме того, в ядре системы `Mathematica` имплементированы **многие сотни** (а вместе с пакетами — **многие тысячи**) других функций, которые либо относятся к специальным предметным областям, либо управляют не работой ядра системы, а, скорее, его взаимодействием с внешним миром (`FrontEnd`, системой, другими программами, файловой структурой и т.д.) В настоящем тексте мы практически не обсуждаем следующие типы команд, за исключением нескольких абсолютно необходимых:

- **системные команды**;
- **команды ввода–вывода**;
- **команды работы с файлами**;
- **команды управления ядром**;
- **команды управления записными книжками**;
- **команды редактирования текста**;
- **команды взаимодействия с другими программами** (`MathLink`).

По нашему мнению, только овладев в достаточной степени внутренними командами, относящимися собственно к языку, пользователь может начинать интересоваться значительно более сложными экстралингвистическими феноменами.

## § 4. Функции нескольких аргументов

В этом параграфе мы обсудим, в чем с точки зрения языка `Mathematica` состоит различие между `f[x,y]`, `f[x][y]` и `f[{x,y}]`. Небрежность в формате задания аргументов функций является еще одной типичнейшей причиной ошибок программирования. Дополнительный нюанс здесь состоит в том, что, как мы увидим в следующих параграфах, многие функции в `Mathematica` могут вызываться с различным количеством аргументов. В этом месте чрезвычайно важно понять извращенную логику программистов, которая состоит в том, что функция определяется *только* тем, **что** она делает, а с кем она это делает — не имеет никакого значения.

Как всегда, мы начинаем с напоминания того, что понимается под функцией нескольких аргументов в математике. Как обычно, рассмотрим отображение $f : X \longrightarrow Y$. Предположим, что $X \subseteq X_1 \times \ldots \times X_n$. В этом случае значение $f((x_1, \ldots, x_n))$ отображения $f$ на $n$-ке $(x_1, \ldots, x_n)$ обычно обозначается просто через $f(x_1, \ldots, x_n)$, а $f$ рассматривается как **функция** $n$ **аргументов** — или, как принято говорить в анализе, **функция** $n$ **переменных**, хотя где здесь *переменные* и кто они такие, никто не знает.

Та операция стирания скобок, которую мы только что произвели, является абсолютно недопустимой с точки зрения языка Mathematica. Дело в том, что теперь аргументами **всех** функций являются *выражения*. А с этой точки зрения у функции $f$, значения которой записываются в формате `f[x1,...,xn]`, имеется $n$ различных аргументов $x_1, \ldots, x_n$, а вот у функции $f$, запись значений которой имеет формат `f[{x1,...,xn}]`, — всего один, а именно **список** $(x_1, \ldots, x_n)$!!! Иными словами, теперь стирание второй пары скобок в выражении $f((x,y))$ ЯВЛЯЕТСЯ ГРУБЕЙШЕЙ СИНТАКСИЧЕСКОЙ ОШИБКОЙ, которая, как правилов, приводит к тому, что система отказывается вычислять значение функции $f$, либо вычисляет совсем не то, что изначально имелось в виду. Иными словами, в языке Mathematica необходимо тщательнейшим образом различать следующие два понятия:

• **список аргументов** $(x_1, \ldots, x_n)$, который записывается в Mathematica как `{x1,...,xn}` или, в полной форме, `List[x1,...,xn]`;

• **последовательность аргументов** $x_1, \ldots, x_n$, которая записывается в Mathematica как `x1,...,xn` или, в полной форме, `Sequence[x1,...,xn]`.

Как и для функций одной переменной основной формой представления функций нескольких переменных считается **функциональная запись**, при которой значение функции $f$ на последовательности аргументов $x_1, \ldots, x_n$ обозначается через `f[x1,...,xn]`. Однако, **кроме того**, в InputForm для этого образа существуют еще **префиксная операторная запись** `f@@x`, при помощи оператора `@@` называемого `Apply`. Для выражения значения `f[x,y]` функции *двух* аргументов, кроме того, используется **инфиксная операторная запись** `x~f~y`. Иначе говоря, образ $f(x,y)$ можно ввести одним из трех следующих *эквивалентных* способов. Кроме того, в иллюстративных целях мы указываем еще и что происходит со списком `{x1,...,xn}` под действием рассмотренного нами в § 1 оператора `@ Composition` и под действием оператора `/@ Map`. Видно, что эти операторы применяются к функциям *одного* аргумента и дают результаты радикально отличающиеся от $f(x,y)$.

| | |
|---|---|
| `f[x,y]` | функциональная запись $f(x,y)$ |
| `x ~ f ~ y` | инфиксная операторная запись $f(x,y)$ |
| `f @@ {x,y}` | префиксная операторная запись $f(x,y)$ |
| `f @ {x,y}` | префиксная операторная запись $f((x,y))$ |
| `f /@ {x,y}` | префиксная операторная запись $(f(x), f(y))$ |

Таким образом, оператор `@ Composition` применяет функцию $f$ к **списку** $x$ с компонентами $x_1, \ldots, x_n$, в то время как оператор `@@ Apply` применяет функцию $f$ к **последовательности** элементов $x_1, \ldots, x_n$ — п(р)очувствуйте разницу!!! Наконец, оператор `/@ Map` применяет функцию $f$ **к каждой компоненте** списка $x$.

Дополнительные переливы смысла появляются вследствие того, что многие функции могут вызываться с различным числом аргументов — даже если забыть про параметры!!! Скажем, операции Plus, Times, And, Or и

другие ассоциативные алгебраические операции могут вызываться с любым количеством аргументов. При этом $f(x)$ истолковывается как $x$, $f(x,y)$ имеет обычные значения, $f(x,y,z)$ истолковывается индуктивно как $f(f(x,y),z)$ и т.д. Это значит, что напечатав, например, `Plus[x,y]` или, *что то же самое*, `x~Plus~y` или, *что то же самое*, `Plus@@{x,y}`, мы вызвали операцию сложения с *двумя* аргументами, что естественно, даст нам обычный результат `x+y`. В то же время, напечатав `Plus[{x,y}]` или, *что то же самое*, `Plus@{x,y}` мы вызвали эту функцию с *одним* аргументом и ответом будет `{x,y}`, скорее всего, не совсем то, что имелось в виду.

Еще один способ задания функций нескольких переменных состоит в рассмотрении **парциальных функций**. Пусть, скажем, $f : X \times Y \longrightarrow Z$ функция двух аргументов. Тогда:

• Зафиксировав $x \in X$ мы получим функцию $f(x,*) : Y \longrightarrow Z$ одного аргумента. Ясно, что функция $f$ полностью определяется сопоставлением каждому $x \in X$ соответствующей парциальной функции $f(x,*)$.

• Зафиксировав $y \in Y$ мы получим функцию $f(*,y) : X \longrightarrow Z$ одного аргумента. Ясно, что функция $f$ полностью определяется сопоставлением каждому $y \in Y$ соответствующей парциальной функции $f(*,y)$.

• В языке `Mathematica` мы выразили бы эти сопоставления с помощью понятия чистой/анонимной функции следующим образом. Парциальная функция $f(x,*)$ записывается как `Function[y,f[x,y]]` или `f[x,#]&`. Точно так же парциальная функция $f(*,y)$ записывается как `Function[x,f[x,y]]` или `f[#,y]&`.

Математики имеют обыкновение резюмировать только что сказанное следующей строкой *канонических* изоморфизмов:

$$\mathrm{Map}(X, \mathrm{Map}(Y,Z)) \cong \mathrm{Map}(X \times Y, Z) \cong \mathrm{Map}(Y, \mathrm{Map}(X,Z))$$

Таким образом, функция $f$ двух переменных находятся в *естественном взаимно однозначном соответствии* с каждой из двух функций одной переменных, значениями которых являются функции одной переменной. Однако то, что две вещи находятся в естественном взаимно однозначном соответствии, еще абсолютно не значит, что эти две вещи совпадают и их следует отождествлять! У каждого гражданина имеется единственный паспорт, а каждый паспорт принадлежит единственному гражданину, но ведь, как замечает по этому поводу Чжуан Чжоу, гражданин и паспорт — это совсем не одно и то же.

Парциальная функции широко используются в математике. Выражение `по каждому аргументу` относится именно к тем свойствам, которые выражаются в терминах парциальных функций. Например, `билинейно` значит именно `линейно по каждому аргументу`. Например, если $U, V$ и $W$ — три векторных пространства над полем $K$, то отображение $U \times V \longrightarrow W$ называется билинейным, если все парциальные функции $f(u,*)$, $u \in U$, и $f(*,v)$, $v \in V$, линейны. Аналогично вводится непрерывность по каждому аргументу, дифференцируемость по каждому аргументу и т.д.

Поэтому в языке Mathematica необходимо *тщательнейшим образом* различать функцию $f$ **двух** аргументов f[x,y] и функцию $g(x)$ **одного** аргумента g[x][y]. Разумеется, между функциями первого типа и семействами $x \mapsto g(x)$ функций второго типа существует естественное взаимно однозначное соответствие. Однако синтаксически это совсем не одно и то же. Более того, во многих случаях, например, когда нам нужно обращаться к функции g[x] *по имени* — скажем, в командах применения к списку — или в тех случаях, когда она *естественно* мыслится как самостоятельная функция, второй формат удобнее. Подчеркнем, что если в первом случае речь идет о *техническом* удобстве, то во втором случае — всего лишь о *психологическом* комфорте!! Приведем несколько примеров:

◦ Пусть $f$ — функция одного аргумента. В этом случае мы привыкли интерпретировать выражение $f^{-1}(x)$ как значение функцию $x \mapsto f^{-1}(x)$, а не как значение функции $(f, x) \mapsto f^{-1}(x)$. В соответствии с этим принято писать InverseFunction[f][x]. Хотя в принципе мы могли бы ввести функцию inversefunction[f_,x_]:=InverseFunction[f][x].

◦ В языке Mathematica $\log_b(x)$ интерпретируется как значение функции *двух* аргументов, а именно, $b$ (base) и $x$ (или, говоря техническим языком, *одного* **параметра** $b$ и *одного* **собственно аргумента** $x$). Обращение к этой функции происходит в формате Log[b,x]. Однако при этом единственный способ обратиться к *парциальной* функции $x \mapsto \log_b(x)$, состоит в том, чтобы вызвать ее как **анонимную функцию** Log[b,#]&. Однако начинающий, вероятно, будет чувствовать себя комфортнее, дав функции $x \mapsto \log_b(x)$ *собственное имя*, например, положив log[b_][x_]:=Log[b,x]. Однако с точки зрения Mathematica функции Log и log абсолютно различны!!! Как мы уже знаем, у функции Log *два* аргумента. Конечно, функция Log может вызываться и с *одним* аргументом $x$, по умолчанию $b = e$, так что значение Log[x] функции Log в $x$ представляет собой натуральный логарифм числа $x$. В то же время, у функции log *один* аргумент, а именно, $b$. И ее *значение* log[b] в $b$ представляет *функцию* $\log_b : x \mapsto \log_b(x)$!!! В свою очередь log[b][x] есть *значение* $\log_b$ в $x$.

**Предостережение.** Мы не уверены, что давать собственным функциям имена, лишь в одной позиции отличающиеся от имен встроенных функций, представляет собой блестящую идею. Например, попытавшись ввести напечатанный выше текст, вы получите в ответ следующее сообщение: General: Possible spelling error: new symbol name "log" is similar to existing symbol "Log". Дальше все зависит, конечно, от присущего Вам **уровня тревожности** (anxiety level). Начинающий пользователь, не уверенный в том, что он все делает правильно, скорее всего, запаникует и постарается переименовать функцию. **Матерый** (mature) программист выключит **все** сообщения об ошибках при помощи команды Off[], а чуть менее циничный выключит *только* сообщение об опечатках посредством Off[General:...]. Мы предпочитаем этого не делать, а просто второй раз нажимаем Shift-Enter, чтобы сообщение об ошибке не записалось в Notebook. Дело в том, что случайные опечатки в одной букве действительно встречаются очень часто!!! Когда Вы вводите log второй раз, система уже воспринимает его как законный символ и больше не жалуется. Тем не менее, мы предпочитаем давать функциям имена, отличающиеся от внутренних функций по крайней мере в *двух* позициях. Например, если мы хотим дать собственное определение экспоненты — а такое желание

у алгебраистов возникает довольно часто!!! — мы называем ее `expo`, а не просто `exp`.

Разумеется, все сказанное относится и к функциям *трех* или большего числа аргументов. Например,

`f[x,y,z]`, `f[x,y][z]`, `f[x][y,z]` и `f[x][y][z]`

представляют собой абсолютно различные способы задания функции!!! Необходимо четко понимать, что только первая из них представляет собой функцию *трех* аргументов, вторая — функцию *двух* аргументов, а третья и четвертая — функции *одного* аргумента. В свою очередь все эти форматы следует тщательно отличать от функций *одного* аргумента

`f[{x,y,z}]`, `f[{x,y}][z]`, `f[x][{y,z}]`!!

Вот наиболее характерные примеры:

∘ Пусть $f$ и $g$ — две функции одного аргумента. В этом случае мы привыкли интерпретировать выражение $(f \circ g)(x)$ как значение функции $x \mapsto f(g(x))$, а не как значение функции $(f, g, x) \mapsto f(g(x))$. В соответствии с этим принято писать `Composition[f,g][x]`. Хотя в принципе мы могли бы ввести функцию *трех* аргументов

```
compositio[f_,g_,x_]:=Composition[f,g][x].
```

∘ Функция `Nest` представляет собой вариант функции `Composition`, описывающий последовательное применение *одной и той же* функции $f$. Однако обычно `Nest` вызывается в формате `Nest[f,x,n]`, где $f$ есть функция, $x$ — аргумент, значение в котором вычисляется, а $n$ — количество применений функции $f$. Допустим, мы хотим дать *имя* функции $f^{\circ n}$. Это снова можно сделать при помощи парциальных функций, например, так. Положим `iterate[f_,n_]:=Nest[f,#,n]&`. Тогда `iterate[f,n]` как раз и представляет $f^{\circ n}$. Например, `iterate[f,5][x]` даст нам `f[f[f[f[f[x]]]]]`.

## § 5. Аргументы функций

> Будда говорил бхикшу, что благодаря самости могут быть атрибуты самости. Если нет самости, нет атрибутов самости.
>
> Сутра неисчислимых смыслов

Аргументы встроенных функций подразделяются на несколько типов, в зависимости от степени их явности, обязательности, а также возможности (и желательности!) их изменять:

- собственно **аргументы**,
- **параметры**,
- **опции**,
- **атрибуты**.

Собственно аргументы — это обычные аргументы в *математическом* понимании функции, и все они *обязаны* явно фигурировать в задании функции в виде `Blabla[x,y]`. Например, у функций `Exp`, `Cos`, `Sin`, `Minus`, `Not` один аргумент, в то время как у функций `Subtract`, `Divide`, `Power`, `Implies`

— два. В случае, если функция вызывается с неправильным количеством аргументов, появится сообщение об ошибке. Ассоциативные алгебраические операции такие как `Plus`, `Times`, `And`, `Or` и некоторые другие функции могут иметь *любое* конечное число аргументов — в том числе не иметь вообще ни одного!

В то же время указание параметров не является обязательным, они могут указываться явно или только подразумеваться. В последнем случае система **по умолчанию** (`by default`) выбирает в качестве параметра что-то хорошо ей известное: `0, 1, {0,1}, 2, 8, 10, 16, 256, E, Pi, GoldenRatio, Infinity, Integer, Real` и тому подобное. **Стандартный выбор** параметров указывается в полном описании функции. При необходимости пользователь может включать явный выбор параметров, отличающихся от стандартных, в тело функции как дополнительные аргументы:

◦ Например, функция `Log` может вызываться с *одним* аргументом, в форме `Log[x]` и в этом случае дает значение *натурального* логарифма числа $x$. В то же время она может вызываться с *двумя* аргументами (собственно аргументом и параметром!!), в форме `Log[b,x]` и в этом случае дает логарифм числа $x$ по **основанию** (`base`) $b$. Например, `Log[2,x]` — *двоичный* логарифм $x$, `Log[10,x]` — *десятичный* и т.д. Иными словами, в этом случае **дефолтное значение** параметра $b$ равное $e$, может быть заменено на любое другое допустимое значение этого параметра.

◦ Точно так же функция `IntegerDigits` может вызываться с *одним* аргументом, в форме `IntegerDigits[n]` и в этом случае дает список *десятичных* цифр целого числа $n$. В то же время она может вызываться с *двумя* или *тремя* аргументами (собственно аргументом $n$ и одним или двумя параметрами!!). Функция `IntegerDigits[n,b]` вычисляет цифры $n$ по основанию $b$, а функция `IntegerDigits[n,b,m]`, кроме того, дополняет получающийся список приписывая к нему слева нули до длины $m$. Иными словами, в этом случае дефолтное значение $b$ равно 10, и — внимание! — дефолтное значение $m$ равно `Length[IntegerDigits[n,b]]`.

◦ Функция `Random` вообще не имеет аргументов, а только *три* параметра:

⋆ **тип** (`Head: Integer`, `Real` или `Complex`),

⋆ **область изменения** (`Range`),

⋆ **количество значащих разрядов** (`Precision`),

причем по умолчанию значение `Head` равно `Real`, значение `Range` — `[0,1]`, и значение `Precision` — `MachinePrecision`. Иными словами, функция `Random` вызванная вообще без аргументов, в форме `Random[]`, даст Вам вещественное число от 0 до 1 с 16 значащими цифрами — из которых только 6 отображаются на экране.

В свою очередь опции и атрибуты описывают режим работы функции, используемые при ее вычислении параметры, тождества и процедуры и тому подобное. Они имеют **стандартные настройки** (`default settings`, `factory defaults`). Все эти настройки можно менять, опции меняются

при помощи специальной конструкции `Rule`, посредством включения текста `Option->Choice` в тело функции, а атрибуты — внешним образом при помощи специальных команд таких как `SetAttributes`.

Если Вы хотите убедиться в том, что выбрана фабричная установка опции, можно явным образом задать правило `Option->Automatic`, кроме того, возможны варианты `Option->None`, `Option->All` и десятки других. Во многих случаях (в первую очередь при выводе графики) выбор опций диктуется *исключительно* эстетическими соображениями. Например, по умолчанию картинка выводится в формате `1:GoldenRatio`:

⋆ Иными словами, дефолтная установка состоит в задании трансформационного правила `AspectRatio->1/GoldenRatio`, дающего стандартный **альбомный формат** (`landscape orientation`).

⋆ Задание правила `AspectRatio->GoldenRatio`)) порождает картинку в стандартном **книжном формате** (`portrait orientation`).

В то же время, будучи математиками, а не книгоиздателями, при выводе графики авторы предпочитают пользоваться одной из следующих установок:

⋆ Правилом `AspectRatio->1`, приводящим к тому, что картинка вписывается в квадрат;

⋆ Правилом `AspectRatio->yscale/xscale`, приводящим к одинаковому масштабу по осям $x$ и $y$;

⋆ Правилом `AspectRatio->Automatic`, приводящим к одинаковому масштабу по осям $x$ и $y$ или, в случае невозможности этого, наиболее удовлетворительному с точки зрения системы результату.

Изменить атрибуты встроенной функции `Blabla` сложнее, для этого нужно проделать ритуальную последовательность телодвижений, начинающуюся с `Unprotect`, `ClearAttributes` или чего-нибудь в таком духе. Это значит, что подразумевается, что пользователь не обязан задавать их — не должен задавать их — а во многих случаях он будет гораздо здоровее, если вообще не подозревает об их существовании!! Дело в том, что изменение атрибутов встроенных функций может *легко* привести к серьезным конфликтам и артефактам при выполнении вычислений, бесконечной рекурсии и т.д. Поэтому — *unless you* **really** *know what you are doing* — ДАЖЕ НЕ ПЫТАЙТЕСЬ менять атрибуты встроенных функций. Более того, обычно в этом нет никакой нужды, гораздо проще ввести новую функцию *с тем же определением*, но С ДРУГИМ ИМЕНЕМ и изменить атрибуты у этой новой функции!! Чтобы у неопытного пользователя и не возникало желания играть с определениями и настройками, встроенные функции имеют АТРИБУТ `Protected`. **Парадокс Рассела**: `Attributes[Protected]===`{`Protected`}.

Укажем несколько обычных соглашений, связанных с только что описанной иерархией аргументов встроенных функций:

• *В большинстве случаев* первыми указываются основные аргументы функции и только потом параметры.

Впрочем, это правило используется *только* в тех случаях, когда оно не входит в конфликт с традиционными математическими обозначениями.

• Если в традиционном математическом обозначении параметры указываются как индексы, то параметры *предшествуют* основным аргументам. Например, $\log_b(x)$ записывается как `Log[b,x]`, а вовсе не как `Log[x,b]`.

• В случае, когда в стандартном обозначении функции фигурируют как верхние, так и нижние индексы, нижние индексы, как правило, *предшествуют* верхним. Например, многочлен Лежандра $P_n^m(x)$ обозначается через `LegendreP[n,m,x]`.

• В ответе на информационный запрос `?Blabla` перечисляются все аргументы и параметры функции с именем `Blabla`.

• В ответе на информационный запрос `??Blabla`, кроме того, перечисляются все опции и атрибуты функции с именем `Blabla`.

## § 6. Алгебраические операции

Напомним, что **(внутренней, бинарной, всюду определенной) алгебраической операцией** на множестве $X$ называется отображение

$$f : X \times X \longrightarrow X, \quad (x, y) \mapsto f(x, y).$$

По умолчанию мы будем называть такие отображения просто **операциями** или **законами композиции** (`Verknüpfung`). При этом $x$ и $y$ называются **операндами**, а $f(x, y)$ — **результатом** операции. В старинных книжках алгебраические операции назывались еще **действиями** — *четыре действия арифметики: служение, почитание, угождение и давление* — однако мы используем термин `действие` (`action`) исключительно в точном техническом смысле: *действие чего-то на чем-то*.

Для алгебраических операций традиционно применяется несколько различных способов записи. Вот некоторые наиболее распространенные:

• Такая запись, когда знак операции пишется **между** операндами, называется **инфиксной**. Такая запись операций применяется особенно часто: $x \bullet y$, $x + y$, $x - y$, $x \cdot y$, $x/y$, $x \circ y$, $x \cup y$, $x \cap y$, $x \vee y$, $x \wedge y$, $x \uparrow y$ и т.д.

• Часто применяются другие записи алгебраических операций, скажем, **префиксная**, когда знак операции пишется **перед** операндами. Обычно используется **функциональная запись** $f(x, y)$. Однако в этом случае можно обойтись вообще без скобок и писать $fxy$. Эта форма записи, называемая по-научному **префиксной операторной записью**, в быту известна как **запись Лукасевича**, а в литературе для младших школьников как **польская запись**. Она весьма обычна в логике и `Computer Science`, но алгебраисты практически никогда ей не пользуются.

• Во многих старых микрокалькуляторах использовалась **постфиксная** запись, когда знак операции вводится **после** операндов. В популярной литературе такая запись обычно называется **обратной польской записью**.

• В многих случаях традиционно используется **циркумфиксная** запись, которая охватывает операнды **с двух сторон** (сочетание совместно работающих префикса и постфикса), скажем, $[x, y]$, $(x, y)$ и т.д.

• Часто используется также **экспоненциальная** запись $x^y$, и другие двумерные формы записи.

В настоящем курсе при обсуждении математических понятий мы в соответствии с алгебраической традицией пользуемся почти исключительно *инфиксной* записью $x * y$. В то же время на уровне программирования мы попеременно пользуемся *инфиксной* записью, которую в языке `Mathematica` принято называть **операторной** или **сокращенной**, и *префиксной* записью $f(x, y)$, которую в языке `Mathematica` принято называть **функциональной** или **полной**.

В дальнейшем мы будем часто ссылаться также на некоторые важнейшие тождества для алгебраических операций.

• Операция $*$ называется **ассоциативной**, если $(x*y)*z = x*(y*z)$ для любых $x, y, z \in X$. В функциональной записи это тождество принимает форму $f(f(x, y), z) = f(x, f(y, z))$. В `Mathematica` ассоциативность операции $f$ выражается атрибутом `Flat`. Этот термин связан с тем, что любое выражение, составленное из последовательных применений $f$, ну, скажем, `f[f[x1,f[f[x2,x3],x4]],x5]`, может **выровнено** (`flattened`), т.е. сведено к форме `f[x1,...,xn]`, в данном случае `f[x1,x2,x3,x4,x5]`.

• Операция $*$ называется **коммутативной**, если $x * y = y * x$ для любых $x, y \in X$. В функциональной записи это тождество принимает форму $f(x, y) = f(y, x)$. В `Mathematica` коммутативность операции $f$ выражается атрибутом `Orderless`. Для операций, обладающих этим атрибутом, переменные автоматически сортируются, иными словами `f[y,x]` преобразуется в `f[x,y]`. Если операция *одновременно* ассоциативна и коммутативна, то же самое относится к любому числу переменных, иными словами `f[y,x,u,v]` будет преобразовано в `f[u,v,x,y]`.

• Говорят, что $e$ — **левый нейтральный** элемент относительно $*$, если $e * x = x$ для любого $x \in X$. Аналогично определяется **правый нейтральный** элемент $e$, для которого $x * e = x$ для всех $x \in X$. Элемент $e \in X$ называется **нейтральным** элементом операции $*$, если он является как левым, так и правым нейтральным. Например, 0 является нейтральным элементом относительно сложения, 1 — относительно умножения, а тождественное отображение id — относительно композиции. Если операция $f$ одновременно ассоциативна и обладает нейтральным элементом $e$, то в `Mathematica` обычно придается смысл результату применения этой операции к *любому* количеству аргументов. А именно, в этом случае `f[]` истолковывается как $e$, а `f[x]` — как $x$. Второе из этих свойств выражается атрибутом `OneIDentity`.

Нам будет часто встречаться еще два важнейших тождества, в которых участвуют **две** алгебраических операции. Рассмотрим множество $X$ с операциями $*$ и $\circ$.

• Определенная на множестве $X$ операция $*$ называется **дистрибутивной слева** относительно $\circ$, если $x * (y \circ z) = (x \circ y) * (x \circ z)$ для любых $x, y, z \in X$.

• Операция $*$ называется **дистрибутивной справа** относительно $\circ$, если $(x \circ y) * z = (x \circ z) * (y \circ z)$ для любых $x, y, z \in X$.

Если $*$ дистрибутивно относительно $\circ$ как слева, так и справа, говорят о **двусторонней дистрибутивности** или просто о **дистрибутивности** $*$ относительно $\circ$. Если операция $*$ коммутативна, то для проверки дистрибутивности $*$ относительно $\circ$ достаточно проверить одностороннюю дистрибутивность. Например, умножение чисел дистрибутивно относительно сложения: $x(y+z) = xy + xz$ и $(x+y)z = xz + yz$. В то же время, потенцирование часто бывает дистрибутивным относительно умножения *справа*, в том смысле, что $(xy)^z = x^z y/z$, но трудно ожидать, чтобы оно было дистрибутивно относительно умножения слева, т.е. чтобы выполнялось тождество $x^{yz} \neq x^y x^z$.

Каждый, кто серьезно пытается понять работу системы `Mathematica`, должен как можно раньше освоиться с мыслью, что в отличие от ассоциативности и коммутативности ДИСТРИБУТИВНОСТЬ НИКОГДА НЕ ПРИМЕНЯЕТСЯ АВТОМАТИЧЕСКИ НИ В ОДНУ, НИ В ДРУГУЮ СТОРОНУ!!! Вместо того, чтобы оценивать, правильно это или нет, стоит вначале задуматься над тем, с чем это связано. Дело в том, что ЛЮБОЕ ПРИМЕНЕНИЕ ДИСТРИБУТИВНОСТИ РАССМАТРИВАЕТСЯ КАК НЕТРИВИАЛЬНОЕ ИЗМЕНЕНИЕ ФОРМЫ ВЫРАЖЕНИЯ. В первую очередь это связано с тем, что применение ассоциативности и/или коммутативности не требует значительных затрат времени или памяти и приводит к результату предсказуемой сложности. Для применения дистрибутивности это совершенно не так!!! Для того, чтобы убедиться в этом, достаточно взглянуть на следующие примеры:

• Попытайтесь раскрыть скобки в чем нибудь *совсем* простеньком, ну, хотя бы в $(x+y)^{100000}$ — раскрытие скобок это в точности применение дистрибутивности *слева направо*. На нашем компьютере выполнение операции `Expand[(x+1)^100000]` занимает около секунды. Но главным соображением с точки зрения конструкторов системы `Mathematica`, как нам кажется, было следующее: длина `(x+y)^100000` равна 2, в то время, как длина `Expand[(x+y)^100000]` равна 100001, не говоря уже про размер коэффициентов. А если еще чуток увеличить экспоненту? Ясно, что автоматическое раскрытие скобок в таких случаях привело бы к еще гораздо большему разбуханию промежуточных результатов, чем то, которое наблюдается в действительности.

• Ну а про применение дистрибутивности в обратную сторону и говорить нечего!!! В частном случае сложения и умножения применение дистрибутивности *справа налево* представляет собой задачу разложения многочленов на множители. Известно, что это непростая задача. Попробовав решить ее с показателями такого-же порядка, как в предыдущем примере, мы сразу получим задачку, решение которой при помощи сегодняшних

алгоритмов может занять довольно значительное время. Поэтому скорее всего результатом такой попытки будет сообщение: `function: Exponent is out of bounds for function Factor`. Но дело, опять же даже не в этом, а в том что при разложении на множители длина выражения тоже может чудовищным образом увеличиваться! Например, длина выражения `x^100000-y^100000` равна 2, но вот длина выражения `(x-z1*y)...(x-z100000*y)`, получающегося после его разложения на множители в кольце $\mathbb{C}[x,y]$, равна 100000. Ну а разложение столь крошечной вещи, как $x^{200}-y^{200}$ на множители над кольцом $\mathbb{Z}[i]$ целых гауссовых чисел занимает считанные секунды!!

Применение дистрибутивности слева направо, т.е. раскрытие скобок, производится при помощи команды `Distribute`. Для различных конкретных ситуаций оно производится также при помощи более сильных команд `Expand`, `PowerExpand` и других, которые раскрывают скобки и, кроме того, обычно делают еще кое что. Точно также применение дистрибутивности справа налево осуществляется либо при помощи специальных команд таких как `Factor`, либо при помощи правил преобразования, которые мы обсуждали в Главе 2:

```
In[1]:={(a+b)*c,Expand[(a+b)*c],a*c+b*c,Factor[a*c+b*c]}
Out[1]={(a+b)*c,a*c+b*c,a*c+b*c,(a+b)*c}
In[2]:={(a^c*b^c,(a*b)^c, PowerExpand[(a*b)^c]}
Out[2]={a^c*b^c,(a*b)^c,a^c*b^c}
```

## § 7. Арифметические операции

> The different branches of Arithmetic — Ambition, Distraction, Uglification and Derision.
>
> Lewis Carroll

Простейшими алгебраическими операциями являются арифметические операции над числами. В системе `Mathematica` они имеют обычный смысл, но применяются к также к многочленам, рациональным дробям, векторам, матрицам, etc., etc., etc.

| | | |
|---|---|---|
| `x+y+z` | `Plus[x,y,z]` | сложение |
| `-x` | `Minus[x]` | переход к противоположному |
| `x-y` | `Subtract[x,y]` | вычитание |
| `x*y*z` | `Times[x,y,z]` | умножение |
| `x/y` | `Divide[x,y]` | деление |
| `x^y` | `Power[x,y]` | потенцирование |

Как всегда, начнем с нескольких важнейших синтаксических моментов. В языке `Mathematica` различие между оператором и функцией является чисто лингвистическим и начинающему нет нужды в него вникать. Сокращенная инфиксная запись алгебраических операций в виде $x \bullet y$ с использо-

ванием специальных символов `+,-,*,/,^` рассматривается как **операторная запись**, в то время как полная запись вида $f[x, y]$ — как **функциональная запись**. Отметим, что знание полных функциональных названий арифметических операций совершенно необходимо, если Вы собираетесь обращаться к ним при написании программ. Вот простейший пример: `Apply[Plus,x]` вычисляет сумму элементов списка $x$, а `Apply[Times,x]` — произведение элементов этого списка. При описании свойств этих операций мы часто будем пользоваться их полными названиями. Отметим некоторые особенности использования арифметических операций в системе `Mathematica`.

• Умножение в языке `Mathematica` выражается посредством $*$ либо пробела. Таким образом, как `x*y`, так и `x y` обозначают произведение $x$ и $y$. В то же время `xy` без пробела представляет собой не произведение $x$ и $y$, а **новую переменную**, с именем `"xy"`.

**Хороший стиль программирования.** Для лучшей читаемости программ и уменьшения вероятности возникновения ошибок НИКОГДА НЕ ОБОЗНАЧАЙТЕ УМНОЖЕНИЕ ПРОБЕЛОМ!!! В самом начале использования системы `Mathematica` смешение `x y` и `xy` было для нас одним из заметных источников ошибок. Никогда не ленитесь напечатать лишнюю $*$. Сейчас мы ставим $*$ даже в тех случаях, когда синтаксически для обозначения умножения не требуется никакого знака. Например, `2x` автоматически истолковывается как `2*x`, `x(y+z)` — как `x*(y+z)`, а `x^yz` — как `x^y*z`. Тем не менее, мы призываем читателя, особенно начинающего, ставить $*$ даже в таких случаях!!! ИСПОЛЬЗОВАНИЕ $*$ ДЛЯ ОБОЗНАЧЕНИЯ УМНОЖЕНИЯ НЕ МОЖЕТ ПРИНЕСТИ НИКАКОГО ВРЕДА, в то же время пропуск $*$ там, где она нужна, может привести к серьезной ошибке.

• В `InputForm` для обозначения потенцирования используется не экспоненциальная $x^y$, а инфиксная запись `x^y`. `Nothing special`: использование инфиксной записи для обозначения потенцирования типично для всех языков программирования.

**Комментарий.** В большинстве языков потенцирование обозначается `x↑y`, но, к сожалению, не на всех клавиатурах есть знак ↑. Поэтому в `Mathematica` используется TEX'овское обозначение `^`. Через палитры начинающий может при желании вводить потенцирование в `TraditionalForm`, т.е. в экспоненциальной записи. Однако мы решительно не рекомендуем это делать! Во первых, это просто **медленнее**, а, во-вторых, как и любой неявный формат, значительно менее удобно **во всех отношениях**, в том числе с точки зрения согласования с другими программами и дальнейшей работы с текстом.

• Основные арифметические операции `Plus`, `Times` и `Power` имеют атрибут `OneIdentity`. Это значит, что их можно вызвать *с одним аргументом*, причем `Plus[x]===x`, `Times[x]===x` и `Power[x]===x`. Более того, `Plus`, `Times` и `Power` можно вызвать *вообще без аргументов*, при этом, *естественно*, `Plus[]===0`, `Times[]===1` и `Power[]===1`.

• Следующий ключевой вопрос состоит в том, как истолковывается выражение `x•y•z` в общем случае? Ответ состоит в том, что РАЗНЫЕ ОПЕРАТОРЫ ИСПОЛЬЗУЮТ РАЗНУЮ ГРУППИРОВКУ. Вот три основных возможно-

сти:

⋆ Операция • **ассоциативна**. Мы уже знаем, что в этом случае $x \bullet y \bullet z = (x \bullet y) \bullet z = x \bullet (y \bullet z)$. К этому типу относятся функции `Plus` и `Times`. А именно, они имеют атрибут `Flat` (ассоциативность) — и, кроме того, атрибут `Orderless` (коммутативность). Это значит, что их можно вызвать *с любым количеством аргументов*, причем ни расстановка скобок, ни порядок аргументов не имеют никакого значения, `Plus[Plus[z,x],y]` значит ровно то же самое, что `Plus[x,y,z]`. Позже мы поговорим об изменении атрибутов, которым можно *заставить* систему думать, что это не одно и то же!

⋆ Для операции • используется **левая группировка**. В этом случае в выражении $x \bullet y \bullet z$ по умолчанию применяется **левонормированная** расстановка скобок, $(x \bullet y) \bullet z$. Операции `Subtract` и `Divide` относятся к этому типу. Это значит, что `x-y-z` равно `(x-y)-z` — в действительности, это некоторое упрощение, через мгновение мы увидим, что как то, так и другое автоматически истолковывается как `x+(-1)*y+(-1)*z`. Точно так же, `x/y/z` равно `(x/y)/z` или, что то же самое, `x/(y*z)` — снова это заведомое упрощение, внутренняя форма всех этих выражений `x*y^(-1)*z^(-1)`.

⋆ Для операции • используется **правая группировка**. В этом случае в выражении $x \bullet y \bullet z$ по умолчанию применяется **правонормированная** расстановка скобок, $x \bullet (y \bullet z)$. К этому типу относится операция `Power`, которая, естественно, не имеет атрибутов `Flat`, $(2^2)^3 \neq 2^{(2^3)}$, и `Orderless`, $2^3 \neq 3^2$. Так вот, не только `x^y^z` автоматически истолковывается как `x^(y^z)`. В действительности в этом случае можно сказать значительно больше: функцию `Power` можно вызвать с любым количеством аргументов! При этом по умолчанию выражение `Power[x,y,z]` истолковывается как `Power[x,Power[y,z]]`.

• В отличие от сложения, умножения и возведения в степень, ВЫЧИТАНИЕ И ДЕЛЕНИЕ НЕ РАССМАТРИВАЮТСЯ КАК САМОСТОЯТЕЛЬНЫЕ ОПЕРАЦИИ. Так, например, спросив `FullForm[Subtract[x,y]]`, Вы получите в ответ `Plus[x,Times[-1,y]]`. Кстати, это значит, что `FullForm[Minus[x]]` имеет вид `Times[-1,x]`. Совершенно аналогично `FullForm[Divide[x,y]]` дает `Times[x,Power[y,-1]]`. В то же время записи `Plus[x,y]`, `Times[x,y]` и `Power[x,y]` представляют собой полные формы. Таким образом, `Subtract`, `Minus` и `Divide` используются исключительно в качестве сокращений для удобства ввода. В частности это значит, что функции `Subtract` и `Divide` могут быть вызваны *только* с двумя аргументами, а функция `Minus` — *только* с одним аргументом.

• Все арифметические операции имеют атрибут `Listable`. Это значит, что их можно применять к векторам, матрицам и т.д., причем они дают поэлементные операции: `{x,y}+{u,v}` совпадает с `{x+u,y+v}`; `{x,y}*{u,v}` — с `{x*u,y*v}`, `{x,y}^{u,v}` — с `{x^u,y^v}` и т.д.

**Предостережение.** В частности, для матриц `x*y` представляет собой **умножение по Адамару**, а вовсе не обычное матричное умножение!!! Умножение матриц (и, в частно-

сти, эвклидово скалярное произведение векторов) обозначается через `x.y` или, в полной форме, `Dot[x,y]`. Точно так же `Power[x,n]` выражает $n$-ю степень матрицы $x$ по Адамару, а вовсе не ее обычную $n$-ю степень, которая обозначается `MatrixPower[x,n]`. Одна из **самых** типичных ошибок при проведении вычислений в линейной алгебре состоит в том, что начинающие используют функции, определенные в терминах поэлементного умножения `Times`, вместо соответствующих функций, определенных в терминах матричного умножения `Dot`, скажем `Exp[x]` вместо `MatrixExp[x]`. Точно также совершенно недопустимо путать `Times` с векторным умножением `x`$\times$`y`, или, в полной форме, `Cross[x,y]`.

**Некоммутативное умножение.** Во многих разделах математики систематически используются некоммутативные операции. Умножение $*$ можно сделать некоммутативным — а при желании даже неассоциативным — при помощи команд:

`ClearAttributes[Times,Orderless]` и `ClearAttributes[Times,Flat]`.

Эти команды действует только на протяжении одной сессии и даже в пределах этой сессии удаленные атрибуты можно снова вернуть командами

`SetAttributes[Times,Orderless]` и `SetAttributes[Times,Flat]`.

Кроме того, в ядре системы `Mathematica` имеется операция `NonCommutativeMultiply[x,y]` имеющая также сокращенное обозначение `x**y`. По умолчанию эта операция предполагается ассоциативной, но не коммутативной.

## § 8. Приоритет!!!

I'm not tense, just terribly A*L*E*R*T!!

American proverb

Рассмотренный нами вопрос группировки произведения $x \bullet y \bullet z$ является частным случаем более общей проблемы **приоритета** (`precedence`) выполнения арифметических операций. `Mathematica` использует обычную **иерархию** арифметических операций: все возведения в степень выполняются до умножений и делений, а все умножения и деления — до сложений и вычитаний. Таким образом, `x-y^zw` будет будет автоматически истолковано как `x-((y^z)w)`.

Стоит отметить, что арифметические операции стоят довольно высоко в общей иерархии операций. Выше них стоят только атомарные выражения, операции, связанные с определением или применением функций, факториалы, некоммутативные умножения и некоторые другие операции. В порядке убывания приоритета общая иерархия операций в `Mathematica` выглядит примерно так (исчерпывающее описание порядка выполнения **всех** операций приведено в таблице в секции A.2.7 книги Вольфрама):

⋆ атомы (числа, символы и стринги);

⋆ слоты `#` и бланки `_`;

⋆ импорт файлов `<<`;

⋆ квадратные скобки `f[x]` и `x[[i]]`;

⋆ инкремент и декремент `++, --`;

⋆ префиксные и инфиксные применения функций `@, @@, /@, //@,` $\sim$, etc.;

⋆ факториал `!`;

⋆ дифференцирование `Derivative`;

⋆ потенцирование `^`, `Sqrt`, etc;

⋆ интегрирование `Integrate` и вычисление частных производных `D`;

⋆ некоммутативное умножение `**`;

⋆ векторное умножение `Cross`;

⋆ матричное умножение `Dot`;

⋆ деление `/`;

⋆ умножение `*`;

⋆ сложение `+`, `-`, etc.;

⋆ отношения `==, !=, <, ,=, >, >=`, etc.;

⋆ отношения `===, =!=, Element`, etc.;

⋆ кванторы `ForAll, Exists`;

⋆ отрицание `Not`

⋆ логические операции `&&, Xor, ||, Implies`, etc.;

⋆ паттерны и условия `:, /;`, etc.;

⋆ подстановки, правила, замены `->, :>, /., //.`, etc.;

⋆ модификации значения переменной `+=, -=, *=, /=`;

⋆ чистые функции `&`;

⋆ постфиксное применение функции `//`;

⋆ присваивания и тяги `=, :=, ^=, ^:=, /:, =.`;

⋆ экспорт в файл `>>, >>>`;

⋆ последовательность команд `;`.

Эти приоритеты выбраны так, чтобы грамотному пользователю не пришлось ставить чрезмерное количество явных скобок в своих программах.

**Упражнение.** Стобы убедиться в том, что Вы поняли, что такое приоритет, ответьте, не включая компьютер, на следующие вопросы:

- Что даст исполнение программы `i=1; Sin[i++]`?
- Что даст исполнение программы `i=1; j=i+=1`?
- Как истолковывается `x!y`? Почему?

Через месяц интенсивного использования программы (или около того!) у Вас выработается достаточно отчетливое понимание того, в каком порядке выполняются операции и когда нужно ставить круглые скобки, убеждающие систему отступить от своих априорных представлений. Тем не менее, нельзя гарантировать, что Вы не сделаете ошибку в случае редко встречавшихся Вам команд или конструкций!!!

**Хороший стиль программирования.** Для лучшей читаемости программ и уменьшения количества ошибок в сомнительных случаях всегда ставьте скобки!!!! Когда мы начинали использовать систему `Mathematica`, мы часто не ставили скобок в таких выражениях, как `x/y*z` — а, кстати,

что такое `x/y*z`??? Действительно ли это `x/(y*z)`, как было задумано, или, все же `(x/y)*z` и, тем самым, равно `(x*z)/y`. Чтобы ответить на этот вопрос, нужно вспомнить, что внутренняя форма выражения `x/y*z` имеет вид `x*y^-1*z`. Но, во-первых, иногда бывает трудно вспомнить то, чего не знаешь (it takes a **long** time to find a black cat in a dark room, if it isn't there — Конфуций), и, во-вторых, не вспоминать же это каждый раз!! Поэтому НИКОГДА НЕ ЛЕНИТЕСЬ НАПЕЧАТАТЬ ЕЩЕ ОДНУ ПАРУ СКОБОК!!! Постановка лишней пары скобок стоит доли секунды, поиск ошибки из-за пропуска пары скобок в сложной программе обычно продолжается гораздо дольше.

## § 9. Итераторы

> Одна старуха от чрезмерного любопытства вывалилась из окна, упала и разбилась. Из окна высунулась другая старуха и стала смотреть вниз на разбившуюся, но от чрезмерного любопытства тоже вывалилась из окна, упала и разбилась. Потом из окна вывалилась третья старуха, потом четвертая, потом пятая. Когда вывалилась шестая старуха, мне надоело смотреть на них.
>
> Даниил Хармс, 'Вываливающиеся старухи'

Допустим, мы хотим ввести многочлен $f_n(x) = 1 + x + x^2 + \ldots + x^n$, основной вопрос, который фактически возникает в этом контексте у начинающих, состоит в том, как ввести многоточие? Иными словами, как повторить какую-то операцию много раз?? В `Mathematica` существует **много** различных ответов на этот вопрос, отличающихся по синтаксису и использованию: явные формулы, рекурсия, применение функции к списку, многократная композиция, организация циклов и т.д.:

◦ Циклоидный программист старой школы, находящийся под *одурманивающим* влиянием стиля `ПошелНа`, скорее всего, организует **цикл**:

```
faa[n_][x_]:=Block[{i,y=1},For[i=1,i<=n,i++,y=y+x^i];Return[y]],
```

**Безнадежно плохо** — и тому, кто *так* понимает программирование, уже ничем не поможешь. Конечно, в *этом* примере цикл тоже является решением, но в большинстве случаев это **худшее** из всех возможных решений, с точки зрения написания программы, а часто и с точки зрения времени исполнения!!!

◦ Наивный, но неиспорченный цивилизацией пользователь (ну, скажем, профессиональный математик) задаст функцию $f_n$ **рекурсией**:

```
fbb[0][x_]=1; fbb[n_][x_]:=fbb[n-1][x]+x^n.
```

Этот стон у них песней зовется, мы сами начинали именно с такого стиля программирования, дурное дело нехитрое. Более того, подобный **дуболомный стиль** работает, ему вообще не нужно учиться и на большинстве учебных примеров время исполнения подобных программ мало отличается от сотых долей секунды. Однако не забудьте установить

```
$RecursionLimit=Infinity!!!
```

◦ Продвинутый программист, воспитанный на $\lambda$-исчислении, `Lisp`, `C++` и `Java`, задаст ту же функцию в стиле функционального программирования:

```
fcc[n_][x_]:=Nest[Expand[1+x#]&,1,n]
```

**Стильно, эффектно**, демонстрирует понимание того, как работают `Nest` и `Function`, и, конечно, несколько лучше, чем

```
fdd[n_][x_]:=Expand[Nest[(1+x#)&,1,n]]
```

но все равно *безнадежно* медленно!!!

◦ Однако тот, кто *реально* овладел концептуальным стилем программирования, свойственным системе `Mathematica`, знает, что *простейшим* ответом на большинство подобных вопросов — ответом, который уже на порядке $n = 10000$ работает **в тысячи раз быстрее** всех предыдущих — является использование **итеративных конструкций** `Table`, `Array`, `Sum`, `Product`, основанных на работе со списками. Иными словами, правильным способом определить $f_n(x)$ является бесхитростное

```
fee[n_][x_]:=Sum[x^i,{i,0,n}]
```

Или, на худой конец, организация цикла, исполненного с помощью `Do` — раза в четыре медленнее, чем работа со списком, но в сотни раз быстрее, чем цикл, организованный с помощью `For`, `While` или чего-нибудь в таком духе!!! Общим для всех этих конструкция является задание **итераторов**, описывающих то, по какой переменной, начиная с какого значения, до какого значения, и с каким шагом происходит повторение операции. Итераторы задаются в одном из следующих форматов:

| | |
|---|---|
| `{n}` | повторить $n$ раз не инкрементируя переменных |
| `{i,n}` | инкрементировать $i$ от 1 до $n$ с шагом 1 |
| `{i,m,n}` | инкрементировать $i$ от $m$ до $n$ с шагом 1 |
| `{i,m,n,d}` | инкрементировать $i$ от $m$ до $n$ с шагом $d$ |
| `{i,k,l},{j,m,n}` | для каждого $i$ от $k$ до $l$<br>инкрементировать $j$ от $m$ до $n$ |

Мы считаем, что в большинстве случаев правильно задавать итераторы в формате `{i,m,n}`, явно указывающем переменную, по которой происходит итерирование, ее начальное и конечное значение. При этом по умолчанию при каждом вызове переменной ее значение увеличивается на 1. Мы обычно задаем итераторы именно таким образом даже в том случае, когда начальное значение равно 1 так что формально вместо `{i,1,n}` было бы печатать просто `{i,n}`. Во-первых, мы считаем, что явное задание начального значения итератора в виде `{i,1,n}` большей степени соответствует математическим традициям, а, во-вторых, получающаяся при этом программа легче читается. При задании итератора в формате `{i,m,n,d}` на каждом шаге происходит инкрементация на $d$.

Вот три важнейших команды, использующие итераторы в таком формате

| | |
|---|---|
| `Table[a[i],{i,m,n}]` | вектор $(a_i)$ с компонентами $a_i$ |
| `Table[a[i,j],{i,k,l},{j,m,n}]` | матрица $(a_{ij})$ с коэффициентами $a_{ij}$ |
| `Sum[a[i],{i,m,n}]` | сумма $\sum_{i=m}^{n} a_i$ |
| `Product[a[i],{i,m,n}]` | произведение $\prod_{i=m}^{n} a_i$ |

Как видно следующего диалога, функции `Sum` и `Product` действуют обычным образом:

```
In[3]:=Sum[Cos[n*x],{n,1,5}]
Out[3]=Cos[x]+Cos[2*x]+Cos[3*x]+Cos[4*x]+Cos[5*x]
In[4]:=Product[x-a[i],{i,1,5}]
Out[4]=(x-a[1])(x-a[2])(x-a[3])(x-a[4])(x-a[5])
```

Пожалуй одна из самых замечательных особенностей этих функций состоит в том, что в качестве начального и/или конечного значения итератора в этих функциях можно взять `-Infinity` и `Infinity`. Например, как вычисление

```
Sum[1/n^2,{n,1,Infinity}]
```

так и вычисление

```
Product[(1-1/Prime[n]^2)^-1,{n,1,Infinity}]
```

даст `Pi^2/6`. Таким образом, `Mathematica` знает, чему равно значение $\zeta(2)$, в том числе в форме Эйлеровского произведения!

## § 10. Делимость целых чисел

Несколько важнейших алгебраических операций определяются в терминах делимости целых чисел. В действительности, большая часть вычислений бесконечной точности с большими числами основана на использовании модулярной арифметики, т.е. вычислениями не с самими этими числами, а с их остатками по нескольким взаимно простым модулям.

Как известно, правильной формой деления целых чисел является деление с остатком. А именно, для любых целых чисел $n, m$ таких, что $m \neq 0$, существуют *единственные* целые $q, r$, такие, что $n = qm + r$ и $0 \leq r < |m|$. Такое $q$ называется **неполным частным** (`quotient`) при делении $n$ на $m$, а $r$ — остатком (`remainder`)

**Предостережение.** *Неполное частное* (`quotient`) целых чисел $n, m$ не следует путать с их *частным* (`ratio`) $n/m$. Единственность здесь достигается за счет того, что здесь мы отступаем от обычного определение деления с остатком в эвклидовых кольцах, добавляя условие положительности остатка!! Ясно, что для общих эвклидовых колец требование положительности остатка абсолютно бессмысленно. На самом деле, обычно на остаток накладывается более слабое условие $|r| < |m|$, в этом случае в кольце целых чисел $\mathbb{Z}$ можно гарантировать лишь, что существует *не более двух* пар $(q, r)$ таких, что $n = qm + r$,

причем это свойство вместе с *неединственностью* остатка однозначно характеризует кольцо $\mathbb{Z}$. В действительности *единственными* эвклидовыми кольцами, в которых деление с остатком выполняется единственным образом, являются кольца многочленов от одной переменной над полем.

Кроме обычных арифметических операций в кольце целых чисел определены еще две важнейшие арифметические операции, сопоставляющие двум целым числам их неполное частное и остаток.

| | |
|---|---|
| `Mod[n,m]` | остаток при делении $n$ на $m$ |
| `Quotient[n,m]` | неполное частное при делении $n$ на $m$ |

Функция `Mod` может вызываться не только с двумя, но и с тремя аргументами, в формате `Mod[n,m,d]`. В этом случае она возвращает не наименьший положительный остаток, а остаток в полуинтервале $[d, d+m)$.

Математики часто применяют деление с остатком не только для целых, но и для вещественных чисел, Например, часто используются факторгруппы аддитивной группы вещественных чисел $\mathbb{R}^+$ по модулю 1 или по модулю $2\pi$. Ни один из аргументов функции `Mod` не обязан быть *целым* числом `Mod[7*Pi,E]` даст `-8*E+7*Pi`. Тем не менее, предполагается, что аргументы `Mod` являются числами. Например, ничего не даст попытка вычислить `Mod[x^2,x^2+1]`, чтобы получить результат в этом случае, вместо `Mod` нужно использовать функцию полиномиального деления с остатком `PolynomialMod`.

Мы уже упоминали, что в `Mathematica` имплементировано большое количество теоретико-числовых функций. Упомянем лишь некоторые наиболее простые из них. Следующая группа функций связана с вычислением наибольшего общего делителя и наименьшего общего кратного.

| | |
|---|---|
| `GCD[l,m,n]` | наибольший общий делитель $l, m, n$ |
| `LCM[l,m,n]` | наименьшее общее кратное $l, m, n$ |
| `ExtendedGCD[l,m,n]` | линейное представление $\gcd(l, m, n)$ |

Снова эти функции не работают для многочленов, где нужно использовать функции `PolynomialGCD` и `PolynomialLCM`.

Следующие функции связаны с делителями, в частности простыми делителями целых чисел.

| | |
|---|---|
| `Divisors[n]` | список делителей $n$ |
| `DivisorSigma[n]` | сумма делителей $n$ |
| `FactorInteger[n]` | разложение $n$ на простые множители |

Однако в действительности уже для чисел с несколькими десятками цифр факторизация представляет абсолютно небанальную задачу Мы подробно обсуждаем связанные с этим вопросы в других местах учебника.

## § 11. Сравнение вещественных чисел

Следующие функции связаны с обычным порядком на множестве вещественных чисел.

| | |
|---|---|
| `Max[x,y,z]` | максимум $x, y, z$ |
| `Min[x,y,z]` | минимум $x, y, z$ |
| `Abs[x]` | абсолютная величина $x$ |
| `Sign[x]` | знак $x$ |

Отметим несколько особенностей этих функций. По умолчанию `Max` и `Min` интерпретируют свои аргументы как вещественные числа! Если Вы попытаетесь вызвать эти функции с символьными переменными или комплексными числами, результатом будет отказ системы возвратить значение функции или сообщение об ошибке. Абсолютная величина интерпретируется обычным образом, как `Abs[x]=Max[x,-x]`. Знак принимает *три* возможных значения $\{-1, 0, 1\}$, а именно,

$$\operatorname{sign}(x) = \begin{cases} -1 & \text{при } x < 0, \\ 0 & \text{при } x = 0, \\ 1 & \text{при } x > 0. \end{cases}$$

В ядре системы `Mathematica` определены и все обычные функции округления до ближайшего целого.

| | | |
|---|---|---|
| `Floor[x]` | $\lfloor x \rfloor$ | пол $x$ |
| `Ceiling[x]` | $\lceil x \rceil = -\lfloor -x \rfloor$ | потолок $x$ |
| `Round[x]` | | ближайшее к $x$ целое |
| `IntegerPart[x]` | | целая часть $x$ |
| `FractionalPart[x]` | | дробная часть $x$ |

За одним исключением использование этих функций ясно из их названия. Отличие же между функциями `Floor` и `IntegerPart` состоит в следующем. Функция `Floor[x]`, которую принято обозначать $\lfloor x \rfloor$, возвращает наибольшее целое, не превосходящее $x$. Это то, что называется **целой частью** числа $x$ и традиционно обозначалось $[x]$ или $\operatorname{Ent}(x)$ (**антье**). В то же время функция `IntegerPart` в отличие от функции `Floor` игнорирует знак числа $x$. Это значит, что для отрицательных чисел `IntegerPart[x]` равно `-Floor[-x]`. Таким образом `IntegerPart[x]` соврадает с `Floor[x]` для $x > 0$ и с `Ceiling[x]` для $x < 0$.

## § 12. Предикаты

В математике **предикатом** называется функция, принимающая значение в множестве `Booleans={True,False}`. Однако в языке `Mathematica` предикатами называются только предикаты от одного аргумента, в то время как предикаты от двух аргументов называются отношениями. Нам уже

встречался предикат `AtomQ`. Вот еще несколько важнейших относящихся к числам предикатов, которые реализованы в ядре `Mathematica`:

| | |
|---|---|
| `IntegerQ[x]` | целочисленность $x$ |
| `EvenQ[x]` | четность $x$ |
| `OddQ[x]` | нечетность $x$ |
| `PrimeQ[x]` | простота $x$ |
| `NumberQ[x]` | $x$ является *явно заданным* числом |
| `NumericQ[x]` | $x$ является числом |
| `MachineNumberQ[x]` | $x$ является машинным числом |

За исключением двух последних предикатов их пафос абсолютно понятен исходя из названий:

• `IntegerQ[x]` спрашивает, имеет ли $x$ *формат* целого числа. Тем самым, `IntegerQ[1]` возвращает значение `True`, в то время как `IntegerQ[1.]` — `False`. В самом деле, 1. является не целым, а приближенным вещественным числом, оно только *кажется* целым (wonderful day, it's your hangover that makes it *seem* terrible).

• `EvenQ[n]` спрашивает, четно ли целое число $n$. Тем самым, `EvenQ[n]` возвращает значение `True`, если $n$ четно и `False`, если $n$ нечетно. В свою очередь, `OddQ[n]` поступает ровно противоположным образом — хотя добиться от системы явного утверждения, что `EvenQ[n]` всегда совпадает с `Not[OddQ[n]]` не научившись вначале объяснять ей, что $n$ целое, т.е. не овладев паттернами, Вам вряд ли удастся!!

• Предикат `PrimeQ[n]` спрашивает, является ли натуральное число $n$ простым. *По идее* `PrimeQ[n]` возвращает `True`, если $n$ является простым и `False` в противном случае. Однако в действительности это вероятностная команда основанная на **тестах простоты**. Если число не проходит хотя бы один из этих тестов, оно заведомо является составным, поэтому значению `False` можно верить. С другой стороны, неизвестно, действительно ли число, проходящее все эти тесты, является простым. Поэтому для больших чисел нельзя исключить возможность того, что `PrimeQ[n]` возвратит ответ `True` и в том случае, когда $n$ является составным (хотя **вероятность** этого чрезвычайно мала!!!). Пакет `NumberTheory‘PrimeQ‘` содержит дополнительные функции, в частности `ProvablePrimeQ[n]`, которая работает гораздо медленнее, чем `PrimeQ[n]`, но зато *гарантирует* правильность ответа для любого $n$.

• Особенно важно уяснить разницу между `NumberQ[x]` и `NumericQ[x]`. Вопрос `NumberQ[x]` это *программистский* вопрос, состоящий в том, является ли $x$ *явно заданным* числом, вычисления с которым производятся по численным, а не символьным алгоритмам. В то же время `NumericQ[x]` является *математическим* вопросом, состоящим в том, принадлежит ли $x$ какому-то числовому множеству (в действительности, конечно, не множеству, а *домену*!) Таким образом, `NumberQ[Pi]` дает `False` — с точки зрения

всех используемых алгоритмов Pi представляет собой не число, а независимую полиномиальную переменную! В то же время NumericQ[Pi] дает True, так как с математической точки зрения $\pi$ является вещественным числом. В свою очередь MachineNumberQ[x] возвращает True, если $x$ является приближенным вещественным или комплексным числом машинной точности (в большинстве бытовых компьютеров, использующих систему Windows, машинная точность 15.9546 знаков).

**Предостережение.** В действительности Mathematica работает не в двузначной логике, как можно заключить из предыдущего, а в трехзначной!!! Скажем, условный оператор If имеет полный формат If[Q,x,y,z] и работает следующим образом. Его вычисление дает

- $x$, если Q==True;
- $y$, если Q==False;
- $z$, если на основе имеющихся у нее сведений программа не может решить, какое из равенств Q==True или Q==False имеет место.

Многие другие предикаты тоже имеют имена вида ClassQ, все эти предикаты дают значение True, если аргумент относится к классу Class и False в противном случае. Приведем список *основных* содержащихся в ядре Mathematica предикатов такого вида (кроме некоторых системных предикатов, связанных с проверкой внешних форматов, MathLink и всем таким). Прежде всего, это предикаты, относящиеся к определению формы и значения выражений.

| | |
|---|---|
| TrueQ[x] | истинность $x$ |
| AtomQ[x] | $x$ является атомом |
| ValueQ[x] | переменная $x$ имеет значение |
| NameQ[x] | стринг "x" является именем |
| StringQ[x] | объект $x$ является стрингом |
| OptionQ[x] | $x$ можно рассматривать как опцию |

Кроме того, имеются предикаты, относящиеся к составу стрингов.

| | |
|---|---|
| DigitQ[x] | стринг "x" состоит только из цифр |
| LetterQ[x] | стринг "x" состоит только из букв |
| LowerCaseQ[x] | стринг "x" состоит из строчных букв |
| UpperCaseQ[x] | стринг "x" состоит из прописных букв |
| SyntaxQ[x] | стринг "x" синтаксически правилен |

И, наконец, предикаты, относящиеся к определению того, является ли объект списком определенного вида.

| | |
|---|---|
| ListQ[x] | $x$ является списком |
| VectorQ[x] | $x$ является вектором |
| MatrixQ[x] | $x$ является матрицей |
| ArrayQ[x] | $x$ является массивом |
| OrderedQ[x] | список $x$ упорядочен |

Важный нюанс здесь состоит в том, что `VectorQ`, `MatrixQ` и `ArrayQ` распознают не только плотные, но и разреженные форматы. Сделаем несколько общих замечаний относительно имен формата `BlablablaQ`.

• Нужно различать два близких слова, `Question` и `Query`. `Question` — это **вопрос**, ответом на который служит `True` или `False`, в то время как `?object` или `??object` представляет собой **запрос** (на программистском жаргоне `Query` или `Information Escape`) и состоит в требовании *информации* об определениях и/или атрибутах объекта `object`! Иными словами, `Query` подразумевает *консультацию*, а не ответ в форме `да/нет`.

• Далеко не все функции с именами такого вида являются предикатами от одного аргумента, многие из них в действительности являются **отношениями**, т.е. требуют два или три аргумента: `ArgumentCountQ`, `FreeQ`, `IntervalMemberQ`, `MatchQ`, `MemberQ`, `PolynomialQ`, `SameQ`, `StringMatchQ`, `UnsameQ`.

• Далеко не все имена, заканчивающиеся на `Q`, являются именами предикатов или отношений!!! Многие являются обычными числовыми функциями, в традиционные математические обозначения которых входит $q$. Например, `PartitionsQ` дает число $q(n)$ разбиений $n$ на различные части, `LegendreQ` — многочлен Лежандра второго рода $Q_n(x)$, etc., etc.

• Еще несколько сотен других подобных функций определено в пакетах. Впрочем, если Вы уже знаете, что такое кватернионы, пафос команды `QuaternionQ[z]` и ее использование очевидны без дальнейших разъяснений.

Однако не все предикаты имеют такой формат, вот некоторые примеры:

| | | |
|---|---|---|
| `x>0` | `Positive[x]` | положительность $x$ |
| `x<0` | `Negative[x]` | отрицательность $x$ |
| `x<=0` | `NonPositive[x]` | неположительность $x$ |
| `x>=0` | `NonNegative[x]` | неотрицательность $x$ |

Использование этих предикатов тоже совершенно понятно. `Positive[x]` дает значение `True` в том и только том случае, когда $x > 0$, и `False` в противном случае. Таким образом, `Positive[x]` означает ровно то же самое, что `TrueQ[x>0]`. Значения остальных предикатов определяются совершенно аналогично.

## § 13. Булевы функции и кванторы

Обсудим теперь булевы функции, т.е. такие функции, у которых как аргументы, так и значения принадлежат домену `Booleans`. Вот основные булевы функции, содержащиеся в ядре `Mathematica`

| | | |
|---|---|---|
| `!x` | `Not[x]` | отрицание $x$ |
| `x&&y&&z` | `And[x,y,z]` | конъюнкция, все $x, y, z$ имеют место |
| `x\|\|y\|\|y` | `Or[x,y,z]` | дизъюнкция, хотя бы одно $x, y, z$ имеет место |
| | `Nand[x,y,z]` | хотя бы одно из $x, y, z$ не имеет места |
| | `Nor[x,y,z]` | ни одно из $x, y, z$ не имеет места |
| | `Xor[x,y,z]` | имеет место *нечетное* число из $x, y, z$ |
| | `Implies[x,y]` | импликация, $x$ влечет $y$ |

Прежде всего, обратите внимание на следующий важный синтаксический момент:

• Знаки `&` и `|` в сокращенном обозначении `And` и `Or` удваиваются!!! Это сделано потому, что одинарные `&` и `|` уже используются в других чрезвычайно важных смыслах. А именно, `&` используется для операторного обозначения чистой функции, а `|` — для обозначения альтернатив в обозначениях паттернов, подстановок, правил преобразования и т.д.

Все эти операции имеют обычный в логике и дискретной математике смысл. Операция `Not` выражает логическое отрицание:

```
In[5]:=Map[Not,{True,False}]
Out[5]={False,True}
```

Таким образом, `Not[x]` в том и только том случае принимает значение `True`, когда `x` принимает значение `False`. Например, `Not[EvenQ[n]]===OddQ[n]`. Вот таблицы истинности остальных логических функций.

```
In[6]:=bool={True,False}; bool2=Flatten[Outer[List,bool,bool],1]
Out[6]={{True,True},{True,False},{False,True},{False,False}};
In[7]:=Map[Apply[And,#]&,bool2]
Out[7]={True,False,False,False}
In[8]:=Map[Apply[Or,#]&,bool2]
Out[8]={True,True,True,False}
In[9]:=Map[Apply[Nand,#]&,bool2]
Out[9]={False,True,True,True}
In[10]:=Map[Apply[Nor,#]&,bool2]
Out[10]={False, False, False, True}
In[11]:=Map[Apply[Xor,#]&,bool2]
Out[11]={False,True,True,False}
In[12]:=Map[Apply[Implies,#]&,bool2]
```

Out[12]={`True,False,True,True`}

Обсудим несколько более тонких моментов, относящихся к употреблению логических связок. Для сравнения двух логических функций используется команда `LogicalExpand`, вносящая все отрицания внутрь и приводящая аргумент к дизъюнктивной нормальной форме.

• Функции `Nand[x,y]` и `Nor[x,y]` являются отрицаниями `And[x,y]` и `Or[x,y]`. Это очевидно из определения и/или из приведенных таблиц истинности, но автоматически в формы `Not[And[x,y]]` и `Not[Or[x,y]]` выражения `Nand[x,y]` и `Nor[x,y]` *не конвертируются*. С другой стороны, **формулы де Моргана** утверждают, что

`LogicalExpand[Nand[x,y]]===Or[Not[x],Not[y]]`,

`LogicalExpand[Nor[x,y]]===And[Not[x],Not[y]]`.

Поэтому в дальнейшем эти связки мы можем отдельно не обсуждать.

• Конечно, операции `Xor` и `Implies` тоже приводятся к нормальной форме. По определению `Xor[x,y]` равно `And[Or[x,y],Not[And[x,y]]]`. Вычисление `LogicalExpand[Xor[x,y]]` дает `Or[And[x,Not[y]],And[y,Not[x]]]`. С другой стороны `LogicalExpand[Implies[x,y]]` дает `Or[y,Not[x]]`. Обе эти формы гораздо менее интуитивны, чем `Xor[x,y]` и `Implies[x,y]`, поэтому отказываться от использования этих связок мы не будем.

• Операция `Not` всегда вызывается с *одним* аргументом, а операция `Implies` — с *двумя* аргументами. Все остальные логические связки могут вызываться с любым количеством аргументов — или вообще без аргументов. Легко видеть (тот, кто не видит, может проверить на компьютере!), что `And[]` принимает значение `True`, `Or[]` и `Xor[]` — значение `False` (напомним, что `Nand[]===Not[And[]]`, а `Nor[]===Not[Or[]]`).

| | |
|---|---|
| `ForAll[x,p]` | $p$ истинно для всех $x$ |
| `Exists[x,p]` | $p$ истинно для какого-то $x$ |
| `Resolve[p]` | элиминировать кванторы в $p$ |
| `Resolve[p,dom]` | элиминировать кванторы в $p$ в области `dom` |

## § 14. Реляционные операторы

Если $R$ — какое-то отношение на некотором множестве $X$, например на множестве $\mathbb{R}$ вещественных чисел, то функция `R[x,y]` проверяет, находятся ли $x$ и $y$ в отношении $R$ и возвращает `True`, если это так и `False` в противном случае. С математической точки зрения отношение является просто предикатом от двух аргументов. Однако отличие от ранее рассмотренных предикатов состоит в том, что аргументы отношений равноправны!

| | | |
|---|---|---|
| `x>y` | `Greater[x,y]` | больше |
| `x>=y` | `GreaterEqual[x,y]` | больше или равно |
| `x<y` | `Less[x,y]` | меньше |
| `x<=y` | `LessEqual[x,y]` | меньше или равно |

Вопрос `x>y` возвращает значение `True`, если система знает, что число $x$ *больше*, чем $y$, значение `False`, если система знает, что число $x$ *не больше*, чем $y$ и, наконец, остается неэвалюированным, если система не знает, какое из этих утверждений имеет место.

Стоит подчеркнуть, что хотя мы описали эти отношения как бинарные, все они могут вызываться с любым количеством аргументов — или даже вообще без аргументов. Кстати, какие значения истинности, по Вашему мнению, принимают `Greater[]` и `Greater[x]`? Выражение `x>y>z>w` или, что то же самое, `Greater[x,y,z,w]` истолковывается как `x>y&&y>z&&z>w`. Это значит, что, например, `Less[Sqrt[2],E,Pi]` принимает значение `True`.

Эти отношения могут фигурировать как в качестве упрощаемых выражений, так и в качестве условий в командах `Simplify`, `FullSimplify`, `Refine`, `Assuming` и т.д. Например, по умолчанию выражение

```
(x+y)/2>=Sqrt[x*y]
```

остается неэвалюированным. В то же время вычисление

```
In[13]:=Refine[(x+y)/2>=Sqrt[x*y],x>=0&& y>=0]
```

даст значение `True`.

Отношения $x > 0$, $x < 0$, $x >= 0$, $x <= 0$ имеют специальные названия:

| | |
|---|---|
| `Positive[x]` | $x$ положительно |
| `Negative[x]` | $x$ отрицательно |
| `NonPositive[x]` | $x$ неположительно |
| `NonNegative[x]` | $x$ неотрицательно |

## § 15. Числовые функции: экспонента и логарифм

В `Mathematica` имеется большое количество различных *типов* встроенных функций. Экспонента и логарифм относятся к так называемым **числовым функциям** (`NumericFunction`), использование которых имеет ряд особенностей. Подразумевается, что эти функции принимают численные значения, в случае, когда их аргументы имеют численные значения, в чем можно убедиться при помощи вопроса `NumericQ`. Кроме того, к этому классу относятся арифметические операции, тригонометрические и обратные тригонометрические функции и несколько десятков специальных функций. Проиллюстрируем на этих простых примерах несколько особенностей использования числовых функций:

• Все эти функции имеют атрибут `Listable`. Иными словами, применение такой функции к списку аргументов дает список значений:

```
In[14]:=Exp[{0,1,Pi,0.5}]
```

Out[14]=$\{1, e, e^{\pi}, 1.64872\}$

• Во всех случаях, *когда это возможно*, их результат интерпретируется не просто как вещественное, а как **точное** целое, рациональное или алгебраическое число.

• Во всех остальных случаях их значение может быть вычислено *с произвольной степенью точности* для произвольных комплексных аргументов. Однако (за исключением *целозначных* функций (таких как `Floor`, `Ceiling` или `Round`) эта точность, вообще говоря, не может быть выше той, с которой заданы аргументы.

• Во всех случаях, *когда это возможно*, производные, неопределенные интегралы, разложения в ряды и пр. для численных функций приведены *в табличном виде*, в терминах других встроенных функций. Это значит, что обращение к ним происходит практически мгновенно.

Обращение к экспоненте и логарифму происходит очевидным образом.

| | |
|---|---|
| `Exp[x]` | экспонента $x$ |
| `Log[x]` | логарифм $x$ |
| `Log[b,x]` | логарифм $x$ по основанию $b$ |

Проиллюстрируем на этом примере еще одно важное обстоятельство.

```
In[15]:=Exp[Log[x]]
Out[15]=x
In[16]:=FullSimplify[Log[Exp[x]]]
Out[16]=Log[e^x]
```

Понятно, что здесь произошло? Экспонента от логарифма всегда равна исходному числу, а вот логарифм от экспоненты только при каких-то дополнительных предположениях. В языке `Mathematica` имеется несколько функций, которые позволяют упрощать выражения при определенных предположениях такие, как `Assuming` и `Refine` и другие. Команда `Refine[x,y]` говорит, какую форму приняло бы выражение $x$, если заменить в нем $x$ на явное число, удовлетворяющее условию $y$. При этом условие может $y$ состоять из любого списка или логической комбинации включений, равенств и неравенств. Теперь понятно, как добиться того, чтобы `Log[Exp[x]]` все же равнялось `x`:

```
In[17]:=Refine[Log[Exp[x]],Element[x,Reals]]
Out[17]=x
In[18]:=Refine[Log[Exp[x]],Element[x,Complexes]]
Out[18]=Log[e^x]
```

Вот еще одна точно такая же ситуация, с возведением в квадрат и извлечением квадратного корня:

```
In[19]:=Sqrt[x]^2
Out[19]=x
In[20]:=FullSimplify[Sqrt[x^2]]
Out[20]=√(x^2)
In[21]:=Refine[Sqrt[x^2],Element[x,Reals]]
```

```
Out[21]=Abs[x]
 In[22]:=Refine[Sqrt[x^2],x>=0]
Out[22]=x
```

## § 16. Тригонометрические и гиперболические функции

Обращение к тригонометрическим функциям вещественного или комплексного аргумента не представляет никакого труда, нужно только иметь в виду, что традиционные английские имена этих функций *отличаются* от русских, например, тангенс $x$ обозначается $\tan(x)$, а не $\mathrm{tg}(x)$ котангенс — $\cot(x)$, а не $\mathrm{ctg}(x)$ и т.д.

| | |
|---|---|
| `Cos[x]` | косинус $x$ |
| `Sin[x]` | синус $x$ |
| `Tan[x]` | тангенс $x$ |
| `Cot[x]` | котангенс $x$ |
| `Sec[x]` | секанс $x$ |
| `Csc[x]` | косеканс $x$ |

По умолчанию аргумент тригонометрической функции измеряется **в радианах**. Во всех случаях, когда это возможно, значение тригонометрических функций вычисляется как *точное* целое, рациональное или алгебраическое число, в остальных случаях остается неэвалюированным:

```
 In[23]:=Cos[{0, 1, Pi/6, Pi/5, Pi/4}]
```

Out[23]=$\{1, \texttt{Cos}[1], \frac{\sqrt{3}}{2}, \frac{1}{4}(1+\sqrt{5}), \frac{1}{\sqrt{2}}\}$

*Численные* значения этих функций находится по обычным правилам для вещественных чисел:

```
 In[24]:=N[Cos[1]]
Out[24]=0.540302
```

Имена основных гиперболических функций получаются добавлением буковки ‘h’ (`hyperbolic`) к имени соответствующей круговой функции:

| | |
|---|---|
| `Cosh[x]` | гиперболический косинус $x$ |
| `Sinh[x]` | гиперболический синус $x$ |
| `Tanh[x]` | гиперболический тангенс $x$ |
| `Coth[x]` | гиперболический котангенс $x$ |
| `Sech[x]` | гиперболический секанс $x$ |
| `Csch[x]` | гиперболический косеканс $x$ |

Имена обратных тригонометрических функций получаются из имен со-

ответствующих прямых функций добавлением префикса `Arc`:

| | |
|---|---|
| `ArcCos[x]` | арккосинус $x$ |
| `ArcSin[x]` | арксинус $x$ |
| `ArcTan[x]` | арктангенс $x$ |
| `ArcCot[x]` | арккотангенс $x$ |
| `ArcSec[x]` | арксеканс $x$ |
| `ArcCsc[x]` | арккосеканс $x$ |

Имена обратных гиперболических функций образуются совершенно аналогично

| | |
|---|---|
| `ArcCosh[x]` | гиперболический арккосинус $x$ |
| `ArcSinh[x]` | гиперболический арксинус $x$ |
| `ArcTanh[x]` | гиперболический арктангенс $x$ |
| `ArcCoth[x]` | гиперболический арккотангенс $x$ |
| `ArcSech[x]` | гиперболический арксеканс $x$ |
| `ArcCsch[x]` | гиперболический арккосеканс $x$ |

Все знание находится повсюду.

Идрис Шах, *Знание как знать*

I believe that mathematical reality lies outside us, and that our function is to discover or observe it, and that the theorems which we prove, and which we describe grandiloquently as our 'creations' are simply our notes of our observations.

Godfrey Harold Hardy

Mathematics is an experimental science, and definitions do not come first, but later on.

Oliver Heavyside

In theory there is no difference between theory and practice. In practice there is.

Yogi Berra

Q: How many mathematicians does it take to screw in a light bulb?
A: None. It's left to the reader as an exercise.

But those who do not care about fanciful things have no reason to read about Ireland.

Lord Dunsany, *My Ireland*

"Dwarf-doors are not made to be seen when shut," said Gimli. "They are invisible, and their own masters cannot find them or open them, if their secret is forgotten."

J.R.R Tolkien, *The Lord of the Rings*

"Have you guessed the riddle yet?" the Hatter said, turning to Alice again.
"No, I give it up," Alice replied. "What's the answer?"
"I haven't the slightest idea," said the Hatter.

Lewis Carroll, *Alice's adventures in Wonderland*

Think, speak, cast, write, sing, number.

William Shakespeare, *Antony and Cleopatra*

По прихоти своей скитаться здесь и там,
Дивясь божественным природы красотам,
И пред созданьями искусств и вдохновенья
Трепеща радостно в восторгах умиленья.
— Вот счастье! вот права ...

Александр Пушкин, *Из Пиндемонти*

Модуль "Задачи" основан на курсе "Математика и компьютер", который мы вели несколько лет на экономическом факультете СПбГУ. Сюда

включено большинство задач, обсуждавшихся на занятиях, а также задачи, предлагавшиеся в качестве домашних заданий, на зачетах и экзаменах.

Задачи, которые рассматривались в рамках этого курса, относились преимущественно к теории чисел (различные способы и форматы представления чисел, простые числа, основы мультипликативной теории чисел, аддитивные задачи), дискретной математике (множества, наборы, списки, функции, отношения), комбинаторике (перестановки, простейшие задачи перечисления, биномиальные коэффициенты, числа Стирлинга и т.д.) и алгебре (многочлены и рациональные дроби, системы алгебраических уравнений, представление матриц и действия над ними, численные инварианты и канонические формы матриц, простейшие алгебраические системы). В качестве обоснования в пользу такого выбора мы можем сказать следующее.

• В этих областях имеется много легко формулируемых задач, решение которых не требует (почти) никаких специфических знаний.

• Эти темы традиционно являются центральными для самой математики, а в последние десятилетия играют огромную и постоянно возрастающую роль в ее приложениях, связанных с компьютерами и информационными технологиями.

• В то же время все эти важнейшие темы практически не представлены в действующих программах по математике для нематематиков!!!

Кроме того, обсуждалось совсем небольшое количество задач, связанных с основами анализа (графики функций, пределы, суммы и произведения, производные и интегралы), элементарной геометрией и элементарной теорией вероятностей (конечные вероятности, генерация случайных объектов).

Основное содержание модуля, представленное в главах 7-9, предваряют несколько замечаний о субстрате, содержании и основных установках курса, и предполагаемом уровне подготовки.

### 1. Соединить идеи с вычислениями[55]

Последние три века развитие математики происходило под действием следующей антитезы:

• слогана Лейбница заменить идеи вычислениями,

• слогана Дирихле заменить вычисления идеями.

Однако, как известно, прогрессивное человечество без труда обалдевает даже простейшими идеями[56].

К сожалению, в конце XIX начале XX в преподавании математики под громкие крики о сохранении традиций произошел *полный* разрыв с традицией. Поэтому все преподавание математики извратило тезис Лейбница настолько успешно, что не только школьный курс математики,

[55] Slick's Third Law of the Universe: There are two types of dirt: the dark kind, which is attracted to light objects, and the light kind, which is attracted to dark objects.

[56] humankind lives under perverted ideas.

но и традиционные вузовские курсы `высшей математики`, `математического анализа`, `аналитической геометрии`, `линейной алгебры` превратились в бессвязные наборы бессмысленных и бессодержательных калькулятивных экзерсисов.

С другой стороны, в XX веке многие математики извратили тезис Дирихле настолько успешно, что значительная часть математических исследований полностью утратила связь не только с вычислениями, не только с математическим естествознанием, но вообще с чем бы то ни было, включая саму математику, и полностью превратилась в артефакты для артефактов.

Понимая и *полностью* разделяя пафос обоих этих высказываний, мы все же считаем, что в истории математики все разумные люди, — включая, конечно самих Лейбница и Дирихле!! — всегда пытались соединить идеи с вычислениями.

В нашем учебнике мы хотим показать, что настоящая математика — как и настоящее программирование — основаны на игре и равновесии идей и вычислений. Мы хотим продемонстрировать, что *любую* идею можно превратить в вычисление и что правильно организованное вычисление может привести не только к результату, но и к пониманию. Мы хотим проиллюстрировать, как можно использовать вычисления для того, чтобы проверить — или опровергнуть! — математические утверждения. И, наоборот, как простые и естественные идеи можно использовать для того, чтобы резко упростить или даже вовсе элиминировать вычисления.

Как математики, мы полностью, безоговорочно, *без всяких резерваций* верим в могущество простых, но могущественных идей. Но мы верим также, что, в той мере, в которой математика является наукой, она, как и все остальные науки, основана на эксперименте. Прежде, чем доказывать результат, нужно убедиться в его справедливости. Закон распределения простых или теорема Дирихле о простых в арифметических прогрессиях являются *прежде всего* экспериментальными фактами. Доказательства сообщают этим фактам дополнительную глубину и объем, но мало меняют нашу уверенность в их истинности.

Так как основной задачей учебника является именно *полное* согласование математической и алгоритмической точек зрения, то характер изложения и предлагаемых задач радикально отличаются от подавляющего большинства как математических, так и компьютерных курсов. Наше изложение в гораздо большей степени основано на эксперименте, чем принято в математике, и, с другой стороны, в гораздо большей степени основано на чистом умозрении, чем обычно в Computer Science.

Задачи, которые мы предлагали на занятиях, на зачетах и экзаменах, являются *слишком* простыми как для опытного математика (который может решить почти все из них вообще не пользуясь компьютером), так и для опытного вычислителя (который для большинства из них может написать программу, не зная почти ничего об их математической сути). Однако, как нам кажется, для студентов первого курса, еще не страдающих профессиональной гипертрофией соответствующих отделов мозга, решение некото-

рых из этих задач может потребовать напряжения воображения и должно способствовать улучшению мозгового кровообращения и выработке полезных навыков.

С другой стороны, в дальнейшем мы постараемся объяснить свою точку зрения, состоящую в том, что на начальном этапе изучения программирования борьба за эффективность `is not an issue`. Именно поэтому мы практически не обсуждаем в нашем учебнике серьезные профессиональные алгоритмы сортировки и поиска, где недостаточно просто полиномиальности, а происходит борьба за степень и мультипликативные константы. Как нам кажется, это отвечает принципиально другому уровню алгоритмического мышления и возможно *только* после того, как студент усвоил основы алгоритмики и научился непринужденно писать коды, выражающие важнейшие математические операции и конструкции.

### 2. Субстрат[57,58]

То, что настоящий курс преподавался экономистам, накладывало заметные ограничения на характер и направленность изложения.

В частности, излагая арифметику мы ограничивались обсуждением исключительно следующих числовых систем, которые реализованы в качестве числовых доменов[59] *в ядре* системы `Mathematica`.

| | | |
|---|---|---|
| `Booleans` | {`True`,`False`} | значения истинности |
| `Integers` | $\mathbb{Z}$ | целые числа |
| `Primes` | $\mathbb{P}$ | простые числа |
| `Rationals` | $\mathbb{Q}$ | рациональные числа |
| `Algebraics` | $\overline{\mathbb{Q}}$ | алгебраические числа |
| `Reals` | $\mathbb{R}$ | вещественные числа |
| `Complexes` | $\mathbb{C}$ | комплексные числа |

В отличие от `Axiom` и `MuPAD` в `Mathematica` кольца классов вычетов $\mathbb{Z}/m\mathbb{Z}$ не оформлены внутренним образом как домены. Тем не менее, модулярная арифметика легко реализуется при помощи функций `Mod` и `PowerMod`, а также установки опции `Modulus->m`, поддерживаемой большинством алгебраических команд. Дефолтная установка этой опции `Modulus->0` соответствует обычной арифметике целых чисел.

---

[57] Мы имеем здесь дело с математикой, а не теологией. Пусть другие математики думают, что им доступно проникновение в мысли Бога об их любимом предмете; мне это всегда казалось пустым и бессмысленным занятием. ©Андре Вейль — А.Вейль. Основы теории чисел. — М.: Мир, 1972. 408с.

[58] Если что-нибудь может подорвать незаслуженный авторитет одних и упрочить справедливую славу других, так это только возможно большее распространение в массах теоретических сведений о музыке. ©Петр Чайковский

[59] В документации к системе домен $\overline{\mathbb{Q}}$ алгебраических чисел обозначается через $\mathbb{A}$, но профессиональные математики чаще обозначают через $\mathbb{A}$ множество *целых* алгебраических чисел.

Если бы мы читали аналогичный курс математикам или физикам, то, несомненно, рассматривали бы в нем наравне с перечисленными выше традиционными числовыми системами также по крайней мере следующие конечные, неархимедовы, некоммутативные и неассоциативные числовые системы:

| | | |
|---|---|---|
| `GaloisField[q]` | $\mathbb{F}_q$ | конечное поле из $q$ элементов |
| `PadicIntegers[p]` | $\mathbb{Z}_p$ | целые $p$-адические числа |
| `Padics[p]` | $\mathbb{Q}_p$ | $p$-адические числа |
| `Quaternions` | $\mathbb{H}$ | кватернионы |
| `Octonions` | $\mathbb{O}$ | октавы Кэли |
| `AlbertNumbers` | $\mathbb{J}$ | числа Алберта |

В действительности, почти все эти числовые структуры реализованы — именно под такими названиями — в стандартных пакетах, входящих в поставку системы. Эти новые типы чисел играют огромную — и все время растущую! — роль не только в самой математике, но и в ее приложениях, как во всех приложениях в Computer Science и передаче информации, так и в современных физических теориях.

• Конечные поля не только естественно возникают в теории чисел, алгебре, комбинаторике и геометрии, но и являются естественным контекстом для большинства результатов криптографии и кодирования.

• Что касается $p$-адических чисел, то, кроме их центральной роли в теории чисел, алгебре и анализе, они находят все более широкие приложения в математической физике. Кроме того, как оказалось, $p$-адические числа с конечным $p$ гораздо лучше, чем вещественные, приспособлены для безошибочных вычислений. Вещественные числа $\mathbb{R} = \mathbb{Q}_\infty$ при этом возникают просто как $p$-адические числа в *бесконечном* простом $p = \infty$.

• Кватернионы и октавы отвечают за существование всех исключительных объектов в математике и, являются адекватным инструментом для описания геометрии физического мира.

К сожалению, школьные и университетские курсы математики *на тысячелетия* отстают от потребностей математики и ее приложений, так что рассмотреть эти системы в рамках отведенного нам времени и предполагавшейся математической подготовки студентов не было никакой возможности. Вот еще два серьезных упущения.

• Ничего не говорится о системе ${}^*\mathbb{R}$ гипервещественных чисел и других неархимедовых системах, в которых вещественные числа расширяются посредством добавления актуально бесконечно малых и/или актуально бесконечно больших. Введение актуально бесконечно малых позволяет дать значительно более внятную и вычислительно эффективную трактовку *всех* вопросов математического анализа.

• Ничего не говорится об арифметике числовых полей, даже простейших квадратичных полей, таких как поле гауссовых чисел $\mathbb{Q}(i)$, поле эйзенштейновых чисел $\mathbb{Q}(\omega)$, пифагорово поле $\mathbb{Q}(\sqrt{2})$ и поле золотого сечения $\mathbb{Q}(\sqrt{5})$. Единственной причиной этого снова была чисто физическая невозможность хотя бы просто упомянуть алгебраическую теорию чисел при ограничениях, накладываемых отведенным временем и подготовкой студентов.

Однако, любой, кто серьезно овладел нашим курсом и знает необходимую математику, сможет легко проводить вычисления в этих и любых других числовых системах.

### 3. Стиль программирования[60,61]

Мы считаем, что требование в элементарных курсах программирования пользоваться исключительно "эффективными" алгоритмами является столь же лицемерным и абсурдным, как требование сопровождать все сообщаемые в элементарных математических курсах результаты "полными и подробными" доказательствами.

• Понятие доказательства не является абсолютным, а носит прагматический и психологический характер. Доказательство — это такое рассуждение, которое убеждает нас настолько, что мы готовы убеждать других.

• Точно так же и понятие эффективности алгоритма носит исключительно прагматический и психологический характер. Эффективный алгоритм — это такой алгоритм, который дает ответ на поставленную задачу на имеющемся оборудовании за приемлемое для пользователя время.

• Важнейшей задачей любого курса программирования на начальном этапе является развитие алгоритмического мышления. Для большинства начинающих серьезной трудностью является понимание алгоритма как такового, а также уточнение и перевод математических понятий на язык понятный компьютеру. С этой точки зрения чрезвычайно полезно сравнивать разные алгоритмы, в том числе даже *заведомо* плохие.

• Для небольших значений параметров – скажем, порядка нескольких сотен или нескольких тысяч – время вычисления пренебрежимо мало по сравнению со временем обмена с памятью и вывода на экран и борьба

---

[60] Ведь я, например, нисколько не удивлюсь, если ни с того ни с сего среди всеобщего будущего благоразумия возникнет какой-нибудь джентльмен с неблагородной или, лучше сказать, с ретроградной и насмешливою физиономией, упрет руки в боки и скажет нам всем: а что, господа, не столкнуть ли нам все это благоразумие с одного разу, ногой, прахом, единственно с той целью, чтоб все эти логарифмы отправились к черту и чтоб нам опять по своей глупой воле пожить! Это бы еще ничего, но обидно то, что ведь непременно последователей найдет. —©Федор Достоевский, *Записки из подполья*

[61] Efficiency is only an issue if the code fails to produce an answer. As long as the memory requirements of the code does not exceed the memory of the computer used and as long as the time required does not exceed the user's patience, the code is efficient enough.—©Joseph Sloan

за эффективность алгоритма вообще не имеет никакого смысла. Здесь часто можно использовать экспоненциальные алгоритмы, которые во многих случаях гораздо проще реализовать. В большинстве реально возникающих задач время, необходимое для реализации более эффективного алгоритма, никак не компенсируется выигрышем в скорости вычисления.

• Более того, для многих задач при таких значениях параметров, при которых они еще решаются в реальное время на бытовом компьютере, асимптотически более эффективные алгоритмы *проигрывают* в скорости менее эффективным за счет больших аддитивных и/или мультипликативных констант, необходимости предварительной обработки данных и других подобных обстоятельств.

• Выбор профессионального алгоритма для решения задачи, находящейся на пределе сегодняшних вычислительных возможностей, является чрезвычайно тонким делом. Например, в большинстве задач линейной алгебры выбор параметров алгоритма определяется абсолютно конкретными деталями используемой системы и зависит от организации конвейера, объема кэша и других подобных обстоятельств. Обсуждение подобных деталей на начальном этапе обучения программированию представляется совершенно абсурдным.

В настоящем пособии параметры подобраны так, чтобы на стандартном персональном компьютере вычисление производилось за секунды — или, в исключительных случаях, за одну-две минуты. Другим ограничением являлась величина вывода. Обычно параметры выбраны так, чтобы получающийся ответ полностью помещался на один экран. Продолжающееся несколько минут вычисление или непомерно длинный вывод в большинстве случаев следует рассматривать как явное указание на ошибку в программе.

## 4. Пререквизиты[62]

Предполагается, что читатель знаком с простейшими понятиями языка `Mathematica` и несколькими десятками основных внутренних функций примерно в объеме приводимого ниже силлабуса, полностью покрываемого содержанием первых двух модулей.

Все используемые в Модуле 3 математические определения и факты, выходящие за рамки школьного курса математики и излагаемые на 1-м курсе анализа и линейной алгебры, нами напоминаются.

---

[62] "You know, it's at times like this when I'm trapped in a Vogon airlock with a man from Betelgeuse and about to die of asphyxiation in deep space that I really wish I'd listened to what my mother told me when I was young!"

"Why, what did she tell you?"

"I don't know, I didn't listen!" ©Douglas Adams, *Hitchhiker's Guide to the Galaxy*

## 5. Силлабус[63,64]

### 5.1. Основы синтаксиса.

• Объекты, атомарные объекты, тип объекта, имя объекта, объекты типов `Integer`, `Rational`, `Real`, `Complex`, `Symbol`, `String`.

• Разделение аргументов функции или компонент списка `Comma ,` и разделение команд или частей аргумента `Semicolon ;`.

• Выражения, правильно составленные выражения. Заголовок, длина, часть, уровень: `FullForm`, `Head`, `Length`, `Part`, `Level`.

• Группировка и скобки: `Parenthesis ()`, `Braces {}`, `Brackets []`, `DoubleBrackets [[]]`.

• Спецификация уровня: `Level`, `Depth`, `n`, `-n`, {`n`}, {`m,n`}.

• Значения истинности `True`, `False` и простейшие тесты `TrueQ`, `SameQ ===`, `UnsameQ =!=`, `IntegerQ`, `PrimeQ`, `EvenQ`, `OddQ`, `NumberQ`, `NumericQ`.

• Проверка принадлежности списку или домену `MemberQ`, `Element`, `FreeQ`.

• Основные булевы операции `Not !`, `And &&`, `Or ||`, `Xor`, `Implies` и кванторы `ForAll`, `Exists`.

• Важнейшие отношения: `Equal ==`, `Unequal !=`, `Greater >`, `GreaterEqual >=`, `Less <`, `LessEqual <=`, `Order`.

• Переменные и их значения. Текущее значение. Немедленное и отложенное присваивание: `Set ==` и `SetDelayed :=`. Чистка: `ClearAll`.

• Немедленная и отложенная подстановка: `Rule ->` и `RuleDelayed :>`.

• Замены: `Replace /.`, `Replace All //.`, `ReplaceRepeated`.

• Внутренние итеративные конструкции: `Do`, `Sum`, `Product`, `Table`. Формы итератора {`n`}, {`i,n`}, {`i,m,n`}, {`i,m,n,d`}.

• Понятие о приоритете, таблица приоритетов.

### 5.2. Числа.

• Домен `Booleans` и числовые домены `Integers`, `Primes`, `Rationals`, `Algebraics`, `Reals`, `Complexes`,

• Задание целых и рациональных чисел. Команды работы с цифрами: `IntegerDigits`, `FromDigits`, `DigitCount`.

• Арифметические операции `Plus +`, `Minus -`, `Subtract -`, `Times *`, `Divide /`, `Power ^`, `Sqrt`.

---

[63] J'ai oublié l'orthographie aussi, et la moitié des mots. Cela n'a pas d'importance, paraît-il. (фр.) — Я также забыл орфографию и половину слов. Но это, скорее всего, не имеет никакого значения ©Samuel Becket, *Molloy*

[64] Забудьте, если знали, и никогда не вспоминайте, и даже не пытайтесь узнать, что означают слова: гаспаччо, буйабез, вишисуаз, минестроне, авголемоно. Не спрашивайте, из каких продуктов сделаны эти блюда, острые они или пресные, холодные или горячие. Вам этого знать не нужно. Да чего там гаспаччо: забудьте разницу между щами и борщом. ©Татьяна Толстая, *Сердца горестные заметы*

• Делимость целых чисел, наибольший общий делитель GCD и наименьшее общее кратное LCM. Факториал Factorial !.

• Деление целых чисел с остатком: Quotient, Mod, PowerMod.

• Простые числа: Prime, PrimeQ, PrimePi. Основная теорема арифметики: FactorInteger, IntegerExponent.

• Задание вещественных чисел и основные числовые форматы. Приближенное значение N и RealDigits. Основные константы: E, Pi, Infinity.

• Основные численные функции: Abs, Sign, Max, Min, Round, Floor, Ceiling.

• Задание комплексных чисел: I, Re, Im, Abs, Arg, Conjugate, ComplexExpand.

• Генерация (псевдо)случайных чисел при помощи функции Random, параметры функции Random, затравка генератора случайных чисел SeedRandom.

### 5.3. Элементарные функции.

• Функция, имя функции, аргумент, значение. Функциональная и операторная запись функции.

• Задание многочленов и их структура: Variables, Coefficient, CoefficientList, Exponent.

• Основные структурные манипуляции с многочленами: Expand, Factor, Collect, Decompose.

• Арифметика многочленов: PolynomialQuotient, PolynomialRemainder, PolynomialMod, PolynomialGCD, PolynomialLCM, PolynomialReduce.

• Рациональные дроби и структурные манипуляции с ними: Numerator, Denominator, Together, Apart, ExpandAll.

• Решение (алгебраических) уравнений: Solve, Roots, Root, LinearSolve, Reduce, Eliminate.

• Основные элементарные функции: Exp, Log, Cos, Sin, Tan, Cot.

• Символьное дифференцирование и интегрирование: Limit, Series, D, Derivative, Integrate, DSolve.

• Упрощение выражений: Simplify, FullSimplify, Refine, Function Expand. Упрощение с предположениями.

### 5.4. Функциональное программирование.

• Основные классы функций языка Mathematica: арифметические операции, символьные вычисления, структурные манипуляции, булевы функции, числовые функции, приближенные функции, функции работы со списками, команды и директивы процедурного и функционального программирования.

• Функции нескольких аргументов. Функциональная запись `f[x,y]`, префиксная операторная запись `f @@ {x,y}`, инфиксная операторная запись `x~f~y` и постфиксная операторная запись `Sequence[x,y] // f`

• Формат аргументов, список и последовательность аргументов, отличие `f[x][y]`, `f[x,y]` и `f[{x,y}]`.

• Типы аргументов. Явные аргументы (собственно аргументы) и неявные аргументы (параметры, опции, атрибуты). Дефолтные значения. Настройки опций `Option->Choice`.

• Фиктивные переменные, бланки `Blank _` и последоваетльности бланков `BlankSequence __`, `BlankNullSequence ___`.

• Задание функций с помощью конструкции `f[x_]:=rhs`. Опоределение с условием `Condition /;`.

• Индукция и рекурсия. База индукции и шаг индукции. Начальные условия и рекуррентные соотношения.

• Контроль времени `Timing`. Сравнение эффективности алгоритмов. Экспоненциальный, полиномиальный и логарифмический рост.

• Сравнение различных определений степени, факториала, биномиальных коэффицентов, чисел Фибоначчи и чисел Стирлинга с точки зрения вычислительной эффективности.

• Конструкция `Remember`, запоминающая таблицу значений функции

```
f[x_]:=f[x]=rhs].
```

• Чистая и анонимная функция `Function &`, слоты и последовательности слотов: `Slot #`, `SlotSequence ##`.

• Итерации функций: `Nest`, `NestList`, `NestWhile`, `FixedPoint`, `Fold`, `FoldList`.

• Применение функций к спискам: `Apply`, `Map`, `MapAt`, `MapAll`, `MapThread`, `MapIndexed`, `Scan`.

• Распределение и протаскивание действия функций: `Inner`, `Outer`, `Distribute`, `Thread`, `Through`, `Operate`.

### 5.5. Списочное программирование.

• Списки и последовательности: `List` и `Sequence`. Выделение частей списка: `Part`, `Extract`.

• Формирование списков: `Table`, `Array`, `Range`, `CharacterRange`.

• Вычеркивания: `Take`, `Drop`, `First`, `Last`, `Most`, `Rest`, `Delete`.

• Вставки и замены: `Insert`, `ReplacePart`, `Append`, `AppendTo`, `Prepend`, `PrependTo`.

• Выборки: `Select`, `Cases`, `DeleteCases`, `Count`, `Position`.

• Простейшие структурные манипуляции: `Join`, `Reverse`, `RotateLeft`, `RotateRight`, `PadLeft`, `PadRight`.

• Вложенные списки и изменение уровней вложенности: `Flatten`, `FlattenAt`, `Partition`, `Split`, `Transpose`.

• Сортировка списков: `Sort`, `Ordering`, `Permutations`, и т.д.

• Основные теоретико-множественные операции: `Union`, `Intersection`, `Complement`.

• Операции над матрицами. Матричное умножение `Dot .`, обратная матрица `Inverse`, функции от матриц `MatrixPower`, `MatrixExp`.

• Системы линейных уравнений: `LinearSolve`, `NullSpace`, `RowReduce`.

• Инварианты матриц: `Det`, `Tr`, `Minors`, `MatrixRank`.

• Собственные числа и векторы: `Eigenvectors`, `Eigenvalues`, `Eigensystem`

### 5.6. Процедурное программирование.

• Основной цикл: `In`, `Out`.

• Организация циклов `Do`, `For`, `While`, `Increment ++`, `Decrement --`.

• Условные операторы: `If`, `Which`, `Switch`.

• Передача управления: `Goto`, `Label`.

• Конструкции локализации переменных `Block`, `Module`.

• Удерживание в вбрасывание: `Hold`, `Return`, `Evaluate`.

• Контексты: `Begin`, `End`. Чтение файлов `Get <<`.

# Глава 7. АРИФМЕТИКА И ТЕОРИЯ ЧИСЕЛ

The genealogical trees at the end of the Red Book of Westmarch are a small book in themselves, and all but Hobbits would find them exceedingly dull. Hobbits delighted in such things, if they were accurate: they liked to have books filled with things that they already knew, set out fair and square and with no contradictions.

J.R.R.Tolkien, *The Lord of the Rings*

## § 1. Целые числа

Why are numbers beautiful? It's like asking why is Beethoven's Ninth Symphony beautiful. If you don't see why, someone can't tell you. I *know* numbers are beautiful. If they aren't beautiful, nothing is.

Paul Erdös

**1. Арифметика целых чисел.**[65,66]

Следующие тесты выясняют, имеет ли число $x$ *формат* целого числа и, в этом случае, является ли оно четным или нечетным.

| | |
|---|---|
| `IntegerQ[x]` | целочисленность $x$ |
| `EvenQ[x]` | четность $x$ |
| `OddQ[x]` | нечетность $x$ |

Стоит еще раз подчеркнуть, `IntegerQ[x]` возвращает `True`, только в том случае, когда число $x$ имеет *заголовок* `Integer`. Во всех остальных случаях, даже если $x$ задекларировано как целое, например, посредством указания паттерна, тест `IntegerQ[x]` вернет значение `False`. Тем самым, `IntegerQ[2]` возвращает значение `True`, в то время как `IntegerQ[2.]` — `False`. Дело в том, что с точки зрения системы 2. является вовсе не целым, а приближенным вещественным числом.

Все арифметические операции в `Mathematica` имеют обычные названия и подчиняются стандартным соглашениям.

| | | |
|---|---|---|
| `x+y+z` | `Plus[x,y,z]` | сложение |
| `-x` | `Minus[x]` | переход к противоположному |
| `x-y` | `Subtract[x,y]` | вычитание |
| `x*y*z` | `Times[x,y,z]` | умножение |
| `x/y` | `Divide[x,y]` | деление |

Единственное, на что следует обращать внимание, это вопросы приоритета. Например, выражение `a/b*c` будет интерпретировано как $ac/b$. Поэтому

[65] Seid umschlungen, Millionen! (нем.: Обнимитесь, миллионы!) ©Friedrich Schiller

[66] A billion here, a couple of billion there — first thing you know it adds up to be real money. ©Bill Gates

во всех сомнительных случаях следует использовать скобки. Выражение $ab/cd$ предпочтительно вводить в виде `(a*b)/(c*d)`.

**1.1.** Сколько среди выражений `a/b/c/d`, `a/(b/c)/d`, `a/b/(c/d)`, `a/(b/c/d)`, `a/(b/(c/d))` различных?

Суммы и произведения вычисляются при помощи внутренних команд `Sum` и `Product`, с итераторами обычного формата.

| | |
|---|---|
| `Sum[a,{i,m,n}]` | сумма $\sum_{i=m}^{n} a_i$ |
| `Product[a,{i,m,n}]` | произведение $\prod_{i=m}^{n} a_i$ |

**1.2.** Найдите сумму всех $n$-значных чисел

**Решение.** Породив таблицу первых 10 таких сумм

```
Table[Sum[i,{i,10^(n-1),10^n-1}],{n,1,10}]
```

мы увидим следующий ответ

45, 4905, 494550, 49495500, 4949955000, 494999550000, 49499995500000, 4949999955000000, 494999999550000000, 49499999995500000000,

где количество идущих подряд девяток на единицу меньше, чем количество нулей. *После* того как этот ответ написан, он становится очевидным. Если Вы предпочитаете видеть не всю таблицу сразу, а отдельные суммы, которые возникают на экране по мере их вычисления, можно напечатать, например, `For[n=1,n<=10,n++,Print[Sum[i,{i,10^(n-1),10^n-1}]]]`

Максимум и минимум конечной последовательности или конечного списка чисел ищутся при помощи команд `Max` и `Min`.

| | |
|---|---|
| `Max[m,n]` | максимум $m$ и $n$ |
| `Min[m,n]` | минимум $m$ и $n$ |

Пусть $2s$ натуральных чисел расположены в порядке возрастания:

$$0 < n_1 < n_2 \ldots < n_{2s-1} < n_{2s}.$$

**1.3.** Разбейте эти числа на пары так, чтобы сумма произведений пар была максимальна.

**1.4.** Разбейте эти числа на пары так, чтобы сумма произведений пар была минимальна.

**1.5.** Разбейте эти числа на пары так, чтобы произведение сумм пар было максимально.

**1.6.** Разбейте эти числа на пары так, чтобы произведение сумм пар было минимально.

**Указание.** Проведите небольшой компьютерный эксперимент. После того, как ответ известен, он легко доказывается индукцией.

**1.7.** Найдите все наборы из $m$ натуральных чисел $\leq n$, для которых их сумма равна изх произведению.

**2. Вычисление степеней.**[67,68]

Следующая функция служит для образования степеней.

| | | |
|---|---|---|
| `x^y` | `Power[x,y]` | потенцирование |

Обратите внимание, что потенцирование использует правую группировку!

**2.1.** Проверьте, как истолковывается выражение `l^m^n`. В связи высказыванием популярного музыканта, приведенным в сноске следующего раздела, вычислите `2^2^7` и `(2^2)^7`.

**2.2.** Дайте рекуррентное определение функции степени $x^n$.

**Решение.** Напрашивающееся определение для того, кто делал вид, что учился математике, но при этом никогда не интересовался программированием, и ничего не вычислял сам, такое:

```
power[x_,0]:=1;
power[x_,n_]:=power[x,n-1]*x
```

Будучи правильным с логической точки зрения в вычислительном смысле это определение *абсолютно чудовищно*. Почему? Подобное определение становится совершенно бесполезным при вычислении чего-нибудь столь крошечного, как $2^{100000}$. Для вычислении $x^n$ по этому определению требуется около $n-1$ умножений, в то время как еще древние египтяне знали, что при вычислении $x^n$ можно обойтись количеством умножений порядка $\log_2(n)$. Следующий способ вычисления степени называется **бинарным** или **египетским алгоритмом**:

$$x^n = \begin{cases} (x^{n/2})^2, & \text{если } n \text{ четно,} \\ x^{n-1}x, & \text{если } n \text{ нечетно.} \end{cases}$$

Непродолжительное размышление убедит вас в том, что 100000 умножений это чуть меньше, чем $2^{100000}$ умножений. Так, выполняя $10^9$ умножений в секунду, мы произведем 100000 умножений за время, меньшее зернистости контроля времени системой `Windows` (1 миллисекунда). Тогда как $2^{100000} \approx 10^{30103}$ умножений не может быть выполнено *никогда*.

**2.3.** А теперь определите функцию $x^n$ еще раз, при помощи египетского алгоритма.

---

[67] O ye POWERS! (for powers ye are, and great ones too) ©Laurence Sterne, *Tristram Shandy*

[68] If all computers of earth were to focus simply on writing down as many digits as possible for the next century, they would write down far less than $2^{2^7}$ digits. ©A.Granville, It is easy to determine whether a given integer is prime. — Bull. Amer. Math. Soc., 2004, vol.42, N.1, p.3–38.

**Решение.** Ну, *хотя бы*, так:

```
power[x_,0]:=1;
power[x_,n_]:=If[EvenQ[n],power[x,n/2]^2,power[x,n-1]*x]
```

Хотя, вероятно, в естественных условиях мы воспользовались бы не условным оператором, а определением с условием:

```
power[x_,0]:=x;
power[x_,n_]:=power[x,n/2]^2 /; EvenQ[n]
power[x_,n_]:=power[x,n-1]*x /; OddQ[n]
```

Конечно, по факту эти определения работают за одинаковое время (но все еще раз в пять медленнее, чем встроенная функция `Power` использующая профессиональные алгоритмы).

Вычисление степеней можно реализовывать и иначе. Еще один чрезвычайно популярный алгоритм вычисления степени, известный очень давно, но систематически исследованный только де Жонкьером в самом конце XIX века, это **метод простого множителя**:

$$x^n = \begin{cases} (x^{n-1})x, & n \text{ простое}, \\ (x^p)^m, & n = pm, \text{ где } p \text{ наименьший простой делитель } n. \end{cases}$$

**2.4.** Определите функцию $x^n$ при помощи метода простого множителя и сравните скорость ее работы со скоростью функции основанной на египетском алгоритме.

**2.5.** Найдите случаи, когда метод простого множителя требует меньше умножений, чем египетский алгоритм.

**Ответ.** Первый случай, когда метод простого множителя лучше, чем египетский алгоритм, это вычисление $x^{15}$, которое требует *шести* умножений по египетскому алгоритму

$$x, x^2, x^3, x^6, x^7, x^{14}, x^{15},$$

и только *пять* умножений по методу простого множителя:

$$x, x^2, x^3, x^6, x^{12}, x^{15}.$$

Впрочем, $x^n$ можно вычислять и многими другими способами, например, вычисление $x^{15}$ при помощи описанного в книге Кнута **дерева степеней** тоже дает *пять* умножений:

$$x, x^2, x^3, x^5, x^{10}, x^{15}.$$

Первый случай, для которого дерево степеней более эффективно, чем метод простого множителя, это вычисление $x^{23}$. При этом используется только *шесть* умножений:

$$x, x^2, x^3, x^5, x^{10}, x^{13}, x^{23},$$

в то время как метод простого множителя требует *семь* умножений:

$$x, x^2, x^4, x^5, x^{10}, x^{11}, x^{22}, x^{23}.$$

Хармс определил функцию, несколько первых значений которой таковы:

$$1,\ 2^2,\ 3^{3^3},\ 4^{4^{4^4}},\ 5^{5^{5^{5^5}}}, \ldots$$

**2.6.** Дайте рекуррентное определение функции Хармса. Сколько значений этой функции Вам удастся вычислить?

**3. Извлечение цифр.**[69] В настоящем параграфе мы начнем отрабатывать технику вычислений с цифрами. Это на непросвещенный взгляд чисто схоластическое занятие имеет по крайней мере два важных приложения. Во-первых, числовые последовательности обычно удобнее всего порождать именно через последовательности цифр. Во-вторых, большинство эффективных современных алгоритмов работы с многозначными числами имитирует вычисления с многочленами и основано на разбиении чисел на блоки цифр.

| | |
|---|---|
| `IntegerDigits[n]` | список цифр числа $n$ |
| `FromDigits[{a1,...,an}]` | преобразование списка цифр в число |
| `DigitCount[n,b,d]` | кратность цифры $d$ в числе $n$ |
| `IntegerExponent[n]` | количество нулей в конце $n$ |

**3.1.** Как узнать, сколько цифр содержит число $n$? Чему равна сумма его цифр? Какая цифра стоит у него в старшем/младшем разряде?

**Ответ.** Проще всего так:

`Length[IntegerDigits[n]]`, `Total[IntegerDigits[n]]`,

`First[IntegerDigits[n]]`, `Last[IntegerDigits[n]]`,

**3.2.** Задайте число $9\ldots9$, состоящее из $n$ девяток.

**3.3.** Задайте число $123\ldots123$, где группа цифр 123 повторяется $n$ раз.

**Решение.** По-хорошему это делается, например, так

```
FromDigits[Flatten[Table[{1,2,3},{n}]]]
```

Функция выравнивания `Flatten` убирает лишний уровень вложенности в получающемся при исполнении `Table` списке `{{1,2,3},...{1,2,3}}`.

Можно, конечно, и без затей

```
Sum[123*1000^(i-1),{i,1,n}]
```

[69] Учитель сказал, что я совсем не знаю математики и поставил мне в дневник какую-то цифру. ©Николай Фоменко

но для больших $n$ это гораздо менее эффективно с вычислительной точки зрения. Кроме того, возможности применения команды `FromDigits` гораздо шире.

**3.4.** Задайте число $10001\ldots10001$, состоящее из $n$ единиц, разделенных группами по три нуля.

**Указание.** Используйте `Join`, чтобы добавить в список последнюю единицу.

**3.5.** Задайте число $1234\ldots9991000$, состоящее из выписанных подряд чисел от 1 до 1000.

**Указание.** Обратите внимание, что наивное `FromDigits[Range[1000]]` дает неправильный результат! Используйте `IntegerDigits` и `Flatten`.

Еще одной внутренней командой является `DigitCount[n]]`, которая показывает *количество* цифр, входящих в запись числа $n$, в следующем порядке: 1,2,3,4,5,6,7,8,9,0.

**3.6.** Определите функцию, которая показывает количество цифр, входящих в разложение числа $n$, в обычном порядке 0,1,2,3,4,5,6,7,8,9.

**Решение.** Конечно, можно просто циклически переставить кратности, возвращаемые `DigitCount`:

```
dc1[n_]:=RotateRight[DigitCount[n]]
```

С другой стороны, такую функцию легко задать и обращаясь непосредственно к `IntegerDigits`:

```
dc2[n_]:=Table[Count[IntegerDigits[n],i],{i,0,9}]
```

**3.7.** Каких чисел среди натуральных чисел $\leq 10^n$ больше: тех в десятичной записи которых встречается 1 или тех, в записи которых она не встречается? А теперь определите функцию, вычисляющую количество тех чисел $\leq 10^n$, в записи которых не встречается 1, и посчитайте первые 10 ее значений.

**3.8.** Имеется $9! = 362880$ девятизначных чисел, состоящих из попарно различных цифр $1,\ldots,9$. Сколько из них являются полными квадратами?

**Указание.** Что легче выбрать, девятизначные числа, являющиеся полными квадратами, или квадраты пятизначных чисел, состоящие из различных цифр?

**Ответ.** Вот эти числа

| | | | | |
|---|---|---|---|---|
| 139854276 | 152843769 | 157326849 | 215384976 | 245893761 |
| 254817369 | 326597184 | 361874529 | 375468129 | 382945761 |
| 385297641 | 412739856 | 523814769 | 529874361 | 537219684 |
| 549386721 | 587432169 | 589324176 | 597362481 | 615387249 |
| 627953481 | 653927184 | 672935481 | 697435281 | 714653289 |
| 735982641 | 743816529 | 842973156 | 847159236 | 923187456 |

Поскольку литература по теории чисел и Computer Science никем не редактируется, большинство многозначных чисел в книгах по этим наукам

воспроизведено с опечатками. Таким образом, при ссылке на любые численные данные их приходится проверять заново. В предположении, что опечатка затронула лишь одну цифру, часто приходится генерировать списки чисел, отличающихся от исходного лишь в одной позиции.

**3.9.** Напишите программу, которая порождает список чисел той же разрядности, отличающихся от исходного в одной цифре.

**Решение.** Например, при помощи `Replace Part`:

```
replacedigit[n_]:=Map[FromDigits[
      ReplacePart[IntegerDigits[n],#[[2]],#[[1]]]]&,
      Complement[Flatten[Outer[List,
      Range[Length[IntegerDigits[n]]],Range[0,9]],1],{{1,0}}]]
```

**3.10.** Напишите программу, которая порождает список чисел той же разрядности, отличающихся от исходного в двух цифрах.

**3.11.** Проверьте, что заменяя в четырехзначном числе не более двух цифр, из него всегда можно получить простое число. Верно ли то же самое для пятизначных чисел?

**3.12.** Из цифр 1, 2, 3, 4, 5, 6, 7, 8, 9 составлены всевозможные числа, не содержащиеся повторяющихся цифр. Найти их сумму.

## 4. Манипуляция с цифрами.[70]

Билет с номером $abcdef$ называется счастливым по петербургски, если

$$a + b + c = d + e + f,$$

т.е. сумма первых трех цифр равна сумме трех последних цифр. Тот же билет называется счастливым по московски, если

$$a + c + e = b + d + f,$$

т.е. сумма цифр, расположенных на нечетных местах равна сумме цифр на четных местах.

**4.1.** Найдите количество счастливых билетов.

**4.2.** Найдите количество дважды счастливых билетов (т.е. таких, которые одновременно являются счастливыми по петербургски и по московски).

**Ответ.** Не пытайтесь вывести на экран *список* номеров счастливых билетов: количество счастливых билетов равно 55252, а дважды счастливых — 6700.

Следующие задачи взяты из классического сборника задач московских математических кружков[71]. Впрочем, его авторы вряд ли предполагали, что эти задачи будут решаться с использованием компьютера!

---

[70] Recreational Number Theory is that part of Number Theory that is too difficult to study seriously. ©H. W. Lenstra, Jr. — AMS-MAA Invited talk, 2002 Annual meeting, San Diego, Jan. 8, 2002.

[71] Д.О.Шклярский, Н.Н.Ченцов, И.М.Яглом, Избранные задачи и теоремы элементарной математики. Ч.I. Арифметика и алгебра. — М., ГИТТЛ, 1954, с.1–455.

**4.3.** Найдите все натуральные числа, которые при зачеркивании последней цифры уменьшаются в целое число раз.

**Ответ.** Очевидно, что число, заканчивающееся на 0 при отбрасывании последней цифры уменьшается в 10 раз. С другой стороны, если последняя цифра не равна нулю, то при ее отбрасывании число уменьшается больше, чем в 10 раз. Теперь вычисляя

```
Select[Range[10,10000],Last[IntegerDigits[#]]!= 0&&
            Mod[#,FromDigits[Most[IntegerDigits[#]]]]==0&]
```

мы получим следующие числа 11, 12, 13, 14, 15, 16, 17, 18, 19, 22, 24, 26, 28, 33, 36, 39, 44, 48, 55, 66, 77, 88, 99. Теперь секундное размышление убеждает нас, что каждое число, уменьшающееся при отбрасывании последней цифры в $m > 10$ раз, обязано быть двузначным, так что мы действительно нашли все такие числа.

**4.4.** Существует ли натуральное число $x$ такое, что его произведения на 2,3,4,5,6 записываются теми же цифрами, что само $x$, но в другом порядке?

**Ответ.** Условие, что $x$ и $y$ имеют один и тот же набор цифр выражается как

```
Sort[IntegerDigits[x]]==Sort[IntegerDigits[y]]
```

Теперь по аналогии с предыдущей задачей совсем легко написать код, выбирающий числа, удовлетворяющие этому условию. Вот наименьший такой пример

$$x = 142857, \quad 2x = 285714, \quad 3x = 428571,$$
$$4x = 571428, \quad 5x = 714285, \quad 6x = 857142.$$

Понятно, что пририсовывая к этому примеру любое количество нулей мы получим число, обладающее таким же свойством. Стоит отметить, что это свойство числа 142857 известно много тысячелетий. Каким образом? Дело в том, что это число представляет собой в точности период десятичной дроби, которой записывается $1/7$. Вычисляя `Table[N[i/7],{i,1,6}]` мы получим следующий знакомый ответ:

```
0.142857, 0.285714, 0.428571, 0.571429, 0.714286, 0.857143
```

Вас, конечно, не смущают последние цифры трех последних чисел, получающиеся в результате так называемого `округления`.

Поиск дальнейших примеров является весьма хлопотным делом, занимающим на бытовом компьютере несколько минут. Среди семизначных чисел кроме 1428570 возникает еще ровно один пример, а именно,

$$x = 1429857, \quad 2x = 2859714, \quad 3x = 4289571,$$
$$4x = 5719428, \quad 5x = 7149285, \quad 6x = 8579142.$$

**4.5.** Существуют ли числа, которые при перестановке первой цифры в конец увеличиваются в три раза?

**Ответ.** Два таких числа, а именно, 142857 и 285714 были найдены нами в предыдущей задаче. Выполняя следующий код,

```
Select[Range[10^6],
            FromDigits[RotateLeft[IntegerDigits[#]]]==3*#&]
```

мы видим, что это единственные шестизначные числа, обладающие таким свойством. С другой стороны, совершенно ясно, что повторяя ту же группу цифр при помощи команды

```
FromDigits[Flatten[Table[{1,4,2,8,5,7},{n}]]]
```

мы получим дальнейшие числа, обладающие тем же свойством.

**4.6.** Найдите все числа $n \leq 10^6$, которые при вычеркивании одной цифры уменьшаются в 9 раз.

**Ответ.** Исполнение кода

```
Select[Range[10^6],MemberQ[
            Table[9*FromDigits[Drop[IntegerDigits[#],{i}]],
                        {i,1,Length[IntegerDigits[#]]}],#]&]
```

показывает, что имеется 87 таких чисел 45, 135, 225, 315, 405, 450, 675, 1125, ...

**4.7.** Найдите все числа $n \leq 10^6$, которые при вычеркивании одной цифры уменьшаются в 9 раз и обладают тем дополнительным свойством, что получившееся число снова делится на 9.

**Ответ.** Нужно лишь внести в предыдущий код дополнительный тест `Mod[Total[IntegerDigits[#]],9]==0&`, выбирающий из списка чисел, получающихся из $x$ вычеркиванием одной цифры, те, сумма которых делится на 9. При этом найдется 18 таких чисел, из которых ровно 7 заканчиваются на 5: 405, 2025, 6075, 10125, 30375, 50625, 70875, а все остальные получаются из них дорисовыванием нулей в конце.

**4.8.** Для данного натурального $n$ найдите все пары натуральных чисел таких, что $l + m = n$ и $m$ получается из $l$ вычеркиванием одной цифры.

**4.9.** Найдите все числа, для которых квадрат заканчивается на само это число — последняя цифра $\neq 0$.

## 5. Палиндромы.[72]

В настоящем разделе мы обсудим операцию **реверсии**, состоящую в переворачивании списка цифр числа. Иными словами, если исходное число $x$ записывается как $a_1 \dots a_n$, то **реверсированное** число $\mathrm{rev}(x) = a_n \dots a_1$ состоит из тех же цифр в обратном порядке. Обратите внимание, что число реверсированное к реверсированному, вообще говоря, не обязано совпадать с исходным. Казалось бы, реверсированные числа представляют собой чистую игру ума, но в действительности в докомпьютерную эпоху они очень широко использовались для вычисления произведения не начиная с

[72] I'm guided by the beauty of our weapons. ©Leonard Cohen — AMS-MAA Invited talk, 2002 Annual meeting, San Diego, Jan. 8, 2002.

младшего разряда, как при обычном школьном “умножении столбиком”, а начиная со старшего разряда, как это принято в теории приближенных вычислений[73].

**5.1.** Определите функцию, которая сопоставляет числу реверсированное к нему число.

**Ответ.** Ну, конечно, это

```
rev[n_]:=FromDigits[Reverse[IntegerDigits[n]]]
```

**5.2.** Найдите все числа $\le 10^7$, которые в 4 раза меньше своего реверсированного.

**Ответ.** Вычисляя

```
Select[Range[10^7],FromDigits[Reverse[IntegerDigits[#]]]==4*#&]
```

мы видим, что таких чисел ровно 4, а именно 2178, 21978, 219978, 2199978, причем все они получаются врисовыванием девяток внутрь первого из них.

**5.3.** Найдите все числа $\le 10^7$, которые в 9 раз меньше своего реверсированного.

**Ответ.** В данном случае ответ таков: 1089, 10989, 109989, 1099989.

**5.4.** Может ли число быть в 2, 3, 5, 6, 7 или 8 раз своего реверсированного?

Число $n$, совпадающее со своим реверсированным, называется **палиндромическим** или, коротко, **палиндромом**. Иными словами, палиндромическое число $a_1 \dots a_n$ читается одинаково слева направо и справа налево.

**5.5.** Задайте тест, который проверяет, является ли число палиндромом.

**Ответ.** Функция `rev` уже определена, поэтому:

```
palinQ[n_]:=TrueQ[rev[n]==n]
```

**5.6.** Ясно, что $11^0 = 1$, $11^1 = 11$, $11^2 = 121$, $11^3 = 1331$, $11^4 = 14641$ (бином Ньютона). Дальше из-за переноса разрядов симметрия нарушается, например, $11^5 = 161051$. Есть ли среди степеней 11 еще палиндромы?

Следующие обобщения этой задачи объясняют, что происходит при решении уравнений $\mathrm{rev}(x^m) = \mathrm{rev}(x)^m$

**5.7.** Проверьте, что квадрат и куб числа 111 являются палиндромами, а остальные степени не являются.

**5.8.** Квадраты чисел 1111, 11111, 111111, 1111111, 11111111, 111111111 являются палиндромами, а остальные степени не являются.

**5.9.** Квадраты чисел $11\dots11$ состоящих более, чем из 10 единиц, не являются палиндромами.

**Ответ.** Они заканчиваются цифрами 987654321, а начинаются цифрами 12345679.

---

[73] Я.С.Безикович, Приближенные вычисления. — ГИТТЛ, Л.–М., 1941, 290с.; см. с.60–64.

Следующая задача предлагалась на вологодской областной олимпиаде 1994 года по программированию[74].

**5.10.** Найдите палиндромы $\leq 10^5$, квадраты которых тоже являются палиндромами.

**Ответ.** Кроме 1, 2 и 3, уже изученных нами чисел 11, 111, 1111, 11111, состоящих их одних 1, и еще двух очевидных вариаций на тему $11^2 = 121$, а именно, $22^2 = 484$ и $121^2 = 14641$, имеется еще ровно 12 примеров:

$$\begin{array}{ccc} 101^2 = 10201 & 202^2 = 40804 & 212^2 = 44944 \\ 1001^2 = 1002001 & 2002^2 = 4008004 & 10001^2 = 100020001 \\ 10101^2 = 102030201 & 10201^2 = 104060401 & 11011^2 = 121242121 \\ 11211^2 = 125686521 & 20002^2 = 400080004 & 20102^2 = 404090404 \end{array}$$

Этот ответ носит общий характер. Сумма квадратов цифр такого палиндрома не превосходит 9. Таким образом, палиндром с $m > 1$ разрядами может содержать только 0, 1 и 2.

• При четном $m$ палиндром либо соддержит не более 8 единиц, либо начинается и заканчивается двойкой, а все остальные цифры нули.

• При нечетном $m$ палиндром либо соддержит не более 9 единиц, либо в середине стоит двойка и он содержит не более 4 единиц, либо он начинается и заканчивается двойкой, все остальные цифры, кроме, быть может, средней, нули, а средняя цифра может равняться единице.

**5.11.** Найдите палиндромы $\leq 10^5$, кубы которых тоже являются палиндромами.

**Ответ.** Кроме 1, 2, $7^3 = 343$ и уже изученных нами чисел 11 и 111 имеется ровно пять примеров

$$101^3 = 1030301, \quad 1001^3 = 1003003001, \quad 10001^3 = 1000300030001,$$
$$10101^3 = 1030607060301, \quad 11011^3 = 1334996994331$$

Чуть позже мы объясним этот ответ.

**5.12.** Найдите палиндромические простые $p \leq 10000$.

**5.13.** Найдите первые двести пар $(p, \mathrm{rev}(p))$, где как число $p$, так и реверсированное к нему число $\mathrm{rev}(p)$ оба простые, причем $p < \mathrm{rev}(p)$.

**5.14.** Найдите все простые $< 10^6$ такие, что каждая циклическая перестановка цифр в них снова дает простое.

**Ответ.** Мы не будем приводить соответствующий код, а ограничимся частью ответа. Имеется четыре группы трехзначных простых с этим свойством:

$$113,\ 131,\ 311, \qquad 197,\ 719,\ 971, \qquad 199,\ 919,\ 991, \qquad 337,\ 373,\ 733,$$

[74] А.С.Сипин, А,И.Дунаев, Областные олимпиады по информатике. — Изд-во Русь, Вологда, 1995, с.1–95.

Обратите внимание, что те из них, которые имеют две одинаковых цифры, остаются простыми вообще при любой перестановке цифр! Имеется две группы четырехзначных

$$1193,\ 1931,\ 3119,\ 9311, \qquad 3779,\ 7937,\ 7793,\ 9377,$$

и для группы пятизначных чисел:

$$11939,\ 19391,\ 39119,\ 91193,\ 93911, \qquad 19937,\ 37199,\ 71993,\ 93719,\ 99371.$$

Авторы не хотят лишать читателей удовольствия самостоятельно найти шестизначные числа с этим свойством.

В следующей задаче предлагается решить уравнение $\mathrm{rev}(x^2) = \mathrm{rev}(x)^2$.

**5.15.** Пусть $(a_1 \dots a_m)^2 = b_1 \dots b_n$. Найдите все числа $\le 10000$, для которых $(a_m \dots a_1)^2 = b_n \dots b_1$.

**Указание.** Ясно, что дорисовывание нулей в конце числа не меняет этого свойства, поэтому рассматривайте только приведенные решения, последняя цифра которых $\neq 0$.

**5.16.** Решите уравнение $\mathrm{rev}(x^3) = \mathrm{rev}(x)^3$.

**Ответ.** Кроме 7 все приведенные решения этого уравнения являются решениями предыдущего и, кроме 2, все состоят их фрагментов вида 1 и 11, разделенных нулями и фрагментов вида 111 отделенных с каждой стороны по крайней мере двумя нулями. Приведем список всех приведенных решений меньших миллиона: 1, 2, 7, 11, 101, 111, 1001, 1011, 1101, 10001, 10011, 10101, 11001, 11011, 100001, 100011, 100101, 100111, 101001, 101011, 101101, 110001, 110011, 110101, 111001.

**5.17.** Решите уравнение $\mathrm{rev}(x^4) = \mathrm{rev}(x)^4$.

**Ответ.** Единственными приведенными решениями этого уравнения являются числа 1, 11, 101, 1001, 10001, ... Все остальные решения получаются из этих дописыванием нулей.

**5.18.** Решите уравнение $\mathrm{rev}(x^5) = \mathrm{rev}(x)^5$.

**Ответ.** Единственным приведенным решением этого уравнения является 1. Интересно, что множество решений каждого из этих уравнений содержится в множестве решений предыдущего.

Следующая простенькая задача предлагалась в 2000 году в качестве утешительной на уральском четвертьфинале студенческой олимпиады по программированию.

**5.19.** При повороте на $\pi$ цифры 0, 1 и 8 не меняются, цифры 6 и 9 переходят друг в друга, а все остальные цифры становятся бессмысленными. Среди двузначных чисел при вращении на $\pi$ не меняются 11, 69, 88 и 96. Найдите все $n$-значные числа, которые не меняются при таком вращении.

**5.20.** Существуют ли простые числа, кроме состоящих из одних единиц, которые не меняются при вращении на $\pi$?

**6. Закон старшего разряда.**[75]

Сейчас произойдет нечто совершенно удивительное. Казалось бы, у случайного числа все должно быть случайно, в том числе и первая цифра.

**6.1.** Исследуйте поведение первой цифры чисел $2^n$, $n \in \mathbb{N}$.

**Решение.** Следующий код вычисляет, сколько раз каждая из цифр 1,2, ... ,9 встречается как первая цифра среди первых $n$ степеней двойки:

```
firstdigit[n_]:=Table[Count[
            Table[First[IntegerDigits[2^m]],{m,1,n}],
                                                i],{i,1,9}]
```

Ответ потрясает воображение. То, что среди первых цифр первых 10 степеней двойки встречается три единицы, кажется случайностью. Однако, взгляд на первые цифры первых 100 степеней

30, 17, 13, 10, 7, 7, 6, 5, 5

делает закономерности в распределении цифр очевидными. При рассмотрении первых цифр первой 1000 степеней

301, 176, 125, 97, 79, 69, 56, 52, 45

подозрения переходят в уверенность. Оказывается, дальше распределение первых цифр практически не меняется. Вот как расположены первые цифры в нескольких последовательных отрезках по 10000 степеней:

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $1-10000$ | 3010 | 1761 | 1249 | 970 | 791 | 670 | 579 | 512 | 458 |
| $10001-20000$ | 3010 | 1762 | 1249 | 969 | 793 | 669 | 579 | 512 | 457 |
| $20001-30000$ | 3010 | 1760 | 1250 | 969 | 791 | 670 | 581 | 511 | 458 |
| $30001-40000$ | 3011 | 1761 | 1249 | 969 | 792 | 668 | 581 | 511 | 458 |
| $40001-50000$ | 3010 | 1760 | 1251 | 969 | 791 | 670 | 580 | 512 | 457 |
| $50001-60000$ | 3010 | 1762 | 1248 | 969 | 793 | 670 | 579 | 512 | 457 |
| $60001-70000$ | 3011 | 1760 | 1250 | 969 | 791 | 670 | 580 | 511 | 458 |
| $70001-80000$ | 3010 | 1763 | 1248 | 969 | 793 | 668 | 580 | 512 | 457 |
| $80001-90000$ | 3010 | 1760 | 1250 | 969 | 792 | 671 | 579 | 512 | 457 |
| $90001-100000$ | 3010 | 1762 | 1248 | 970 | 792 | 669 | 579 | 511 | 459 |

Не пытайтесь повторить этот "смертельный" номер! Вычисление этой таблицы на стандартном ноутбуке может занять значительное время.

Если Вы думаете, что это характерно именно для степеней 2, Вас ждет следующий сюрприз.

**6.2.** Исследуйте поведение первой цифры чисел $3^n$, $n \in \mathbb{N}$.

**Ответ.** Среди первых цифр первых 10 степеней тройки цифра 2 встречается три раза, $3^3 = 27$, $3^5 = 243$, $3^7 = 2187$, а цифра 1 — всего один

[75] Но, желаньем подстрекаем

Их сюрпризом удивить,

Не давай, подлец, быка им

В виде опыта доить! ©Алексей Константинович Толстой, *Мудрость жизни*

раз, $3^9 = 19683$. Однако, дальше все постепенно становится на свои места. Вот распределение первых цифр среди первых 100 степеней, первой 1000 степеней и первых 10000 степеней:

28, 19, 12, 8, 9, 7, 7, 5, 5;

300, 177, 123, 98, 79, 66, 59, 52, 46;

3007, 1764, 1247, 968, 792, 669, 582, 513, 458.

Обратите внимание, что из-за аномального поведения в первом десятке цифра 1 все еще встречается чуть реже, а 2 — чуть чаще, чем должны.

**6.3.** Исследуйте поведение первой цифры чисел $m^n$, где $n \in \mathbb{N}$, а $m$ не является степенью 10.

Теперь Вы, конечно, больше не думаете, что этот закон применим только к степеням?

**6.4.** Исследуйте поведение первой цифры чисел Фибоначчи $F_n$, где $n \in \mathbb{N}$.

**Ответ.** Да в точности то же самое. Вот распределение первых 100, 1000 и 10000 чисел Фибоначчи по первой цифре:

30, 18, 13, 9, 8, 6, 5, 7, 4

301, 177, 125, 96, 80, 67, 56, 53, 45

3011, 1762, 1250, 968, 792, 668, 580, 513, 456

**6.5.** Исследуйте поведение первой цифры факториала $n!$, где $n \in \mathbb{N}$.

Конечно, такое точное совпадение ответов для разных классов чисел не может быть случайным. И действительно, **закон Ньюкомба**, известный также как **закон старшего разряда**, утверждает, что цифра $d$ встречается в качестве ведущей с вероятностью $\log_{10}(1 + 1/d)$. Вот значения логарифма с точностью до 6 значащих цифр:

$$0.301030,\quad 0.176091,\quad 0.124939,\quad 0.0969100,\quad 0.0791812,$$
$$0.0669468,\quad 0.0579919,\quad 0.0511525,\quad 0.0457575$$

Как отмечает В.И. Арнольд, это объясняет многие реально наблюдаемые явления, связанные с экспоненциальным ростом, например, почему численность населения примерно одной страны из трех начинается с 1. То же относится к основным физическим постоянным — в любой системе единиц![76]

[76] Ну, как не воскликнуть: “I do not know as much as God, but I know as much as God did at my age!” ©Milton Shulman

## § 2. Рациональные числа

> Уровень научного образования во всех странах мира неуклонно снижается, а Россия в этом общемировом процессе, как и в других, отстает. Например, некоторые наши школьники до сих пор свободно складывают дроби, тогда как все американские *студенты* и 80% школьных учителей давно уже думают, что $1/2+1/3=2/5$.
>
> В.И.Арнольд, *Что такое математика*?

В данном параграфе мы обсуждаем основы вычислений с рациональными числами. Поле $\mathbb{Q}$ рациональных чисел является полем частных кольца $\mathbb{Z}$ целых чисел и, таким образом, с одной стороны вычисления в нем сводятся к вычислениям с целыми числами, а, с другой стороны, интерпретация в терминах дробей позволяет гораздо проще проводить многие вычисления с целыми числами.

**1. Числитель и знаменатель.**[77]

Рациональное число $x$ представляется как частное двух целых чисел $m/n$, $n \neq 0$, при этом автоматически производятся все сокращения, которые система в состоянии найти.

| | |
|---|---|
| `Numerator[x]` | числитель $x$ |
| `Denominator[x]` | знаменатель $x$ |

Таким образом, `Numerator[m/n]` и `Denominator[m/n]` возвращают не $m$ и $n$, а $m/\gcd(m,n)$ и $n/\gcd(m,n)$, причем знак относится к числителю.

**1.1.** Как известно, многие школьники сокращают правильные дроби, зачеркивая одинаковую цифру в числителе и знаменателе. Например, зачеркивая 6 в $26/65$ мы получаем $2/5$. Найдите все правильные дроби со знаменателями $< 1000$, которые можно сокращать таким образом.

**Ответ.** Вот все такие дроби с двузначными числителями и знаменателями:

$$\frac{16}{64}=\frac{1}{4},\qquad \frac{19}{95}=\frac{1}{5},\qquad \frac{26}{65}=\frac{2}{5},\qquad \frac{49}{98}=\frac{4}{8}.$$

Вот такие дроби с двузначным числителем и трехзначным знаменателем:

$$\frac{13}{325}=\frac{1}{25}\quad \frac{27}{756}=\frac{2}{56}\quad \frac{34}{136}=\frac{4}{16}\quad \frac{34}{238}=\frac{4}{28}\quad \frac{39}{195}=\frac{3}{15}$$
$$\frac{39}{975}=\frac{3}{75}\quad \frac{49}{196}=\frac{4}{16}\quad \frac{49}{294}=\frac{4}{14}\quad \frac{49}{392}=\frac{4}{32}\quad \frac{59}{295}=\frac{5}{25}$$
$$\frac{67}{268}=\frac{7}{28}\quad \frac{67}{469}=\frac{7}{49}\quad \frac{79}{395}=\frac{7}{35}\quad \frac{83}{332}=\frac{8}{32}\quad \frac{96}{192}=\frac{6}{12}$$
$$\frac{97}{194}=\frac{7}{14}\quad \frac{97}{291}=\frac{7}{21}\quad \frac{98}{196}=\frac{8}{16}\quad \frac{98}{294}=\frac{8}{24}\quad \frac{98}{392}=\frac{8}{32}$$

[77] Can you do Division? Divide a loaf by a knife — what's the answer to that? ©Lewis Carroll, Through the looking glass

Что касается дробей с трехзначными числителями и знаменателями, то их уже довольно много. Кроме тривиальных примеров 101/202, 101/303, 102/204, 103/206, встречается и несколько более интересные примеры, скажем, $133/931 = 13/91$.

Как известно, многие школьники складывают дроби складывая их числители и знаменатели. Однако, *хороший* школьник знает, что так можно складывать только *несократимые* дроби. А именно, дробь

$$\frac{a}{b} \dot{+} \frac{c}{d} = \frac{a+c}{b+d}$$

называется **медиантой** дробей $a/b$ и $c/d$.

**1.2.** Задайте медианту двух дробей.

**Решение.** Если ровно двух, то проще всего так:

```
medianta[x_,y_]:=(Numerator[x]+Numerator[y])/
                        (Denominator[x]+Denominator[y])
```

Однако, следует иметь в виду, что из-за возможных сокращений медианта не ассоциативна. Поэтому в природных условиях мы бы, скорее всего, определили медианту так:

```
medianta[x__]:=Total[Numerator[{x}]]/Total[Denominator[{x}]]
```

В 1816 году английский геолог Джон Фарей расположил все (несократимые) дроби $0 \le l/m \le 1$ со знаменателем $m \le n$ в порядке возрастания и высказал предположение, что каждый член этой последовательности является медиантой двух соседних. Получающаяся так последовательность дробей называется **последовательностью Фарея** порядка $n$.

**1.3.** Напишите команду, порождающую последовательность Фарея и убедитесь в справедливости гипотезы Фарея.

**Решение.** Ну, например, так:

```
farey[n_]:=Union[Flatten[Table[i/j,{j,1,n},{i,0,j}],1]]
```

Теперь вычисление

```
Do[Print[farey[n]],{n,1,6}]
```

Вернет нам ответ

```
{0,1}
{0,1/2,1}
{0,1/3,1/2,2/3,1}
{0,1/4,1/3,1/2,2/3,3/4,1}
{0,1/5,1/4,1/3,2/5,1/2,3/5,2/3,3/4,4/5,1}
{0,1/6,1/5,1/4,1/3,2/5,1/2,3/5,2/3,3/4,4/5,5/6,1}
```

Разумеется, 0 истолковывается как 0/1.

**2. Гармонические числа.**

Из курса анализа хорошо известен гармонический ряд:

$$1+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\frac{1}{5}+\dots$$

Его частичные суммы называются **гармоническими числами**:

$$H_n = 1+\frac{1}{2}+\frac{1}{3}+\dots+\frac{1}{n}.$$

Вот несколько первых гармонических чисел:

$$H_1 = 1,\ H_2 = \frac{3}{2},\ H_3 = \frac{11}{6},\ H_4 = \frac{25}{12},\ H_5 = \frac{137}{60},\ H_6 = \frac{49}{20},$$
$$H_7 = \frac{363}{140},\ H_8 = \frac{761}{280},\ H_9 = \frac{7129}{2520},\ H_{10} = \frac{7381}{2520}.$$

По самому определению $n$-е гармоническое число рационально. Являясь дискретным аналогом логарифма, гармонические числа настолько часто возникают в комбинаторных задачах и при анализе алгоритмов, что в систему встроена специальная функция `HarmonicNumber` для их быстрого вычисления.

| | |
|---|---|
| `HarmonicNumber[n]` | $n$-е гармоническое число |
| `EulerGamma` | константа Эйлера |

Следующее упражнение основано на том, что знаменатель $H_n$ является делителем $\mathrm{lcm}(2,\dots,n)$ — и, тем более, $n!$. Таким образом, все числа $\mathrm{lcm}(2,\dots,n)H_n$ — и, тем более, $n!H_n$ — целые.

**2.1.** Убедитесь, что при $n > 1$ число $H_n$ не может быть целым.

**Указание.** Посмотрите на числа $\mathrm{lcm}(2,\dots,n)H_n$, все они нечетны. Это значит, что входящая в знаменатель $H_n$ степень 2 никогда не сокращается.

**2.2.** Вычислите таблицу первых 100 гармонических чисел. Найдите те простые, на которые в них происходит сокращение.

Посмотрим теперь на числители гармонических чисел. Числитель $H_2$ делится на 3, но не на $3^2$. Зато числитель $H_4$ делится на $5^2$, числитель $H_6$ — на $7^2$, а числитель $H_{10}$ — на $11^2$. Вообще, **теорема Вольстенхольма** утверждает, что для любого простого $p > 3$ числитель $H_{p-1}$ делится на $p^2$.

**2.3.** Проверьте теорему Вольстенхольма для первой тысячи простых.

**Решение.** Можно так.

```
Apply[And,Table[TrueQ[
    Mod[Numerator[HarmonicNumber[Prime[i]-1]],Prime[i]^2]==0],
                                                {i,3,1002}]]
```

Если мы интересуемся только целыми частями, $n$-е гармоническое число довольно точно аппроксимирует $\ln(n)$ — или, в зависимости от точки зрения, аппроксимируется $\ln(n)$.

**2.4.** Оцените разницу $\ln(n)$ и $H_n$ для $n \leq 1000$. Убедитесь, что всегда

$$\ln(n) < H_n < \ln(n) + 1.$$

В действительности существует предел $H_n - \ln(n)$ при $n \longrightarrow \infty$. Этот предел обозначается $\gamma$ и называется **константой Эйлера**. Вычисление `N[EulerGamma,50]` дает первые 50 знаков константы Эйлера:

$$0.57721\,56649\,01532\,86060\,65120\,90082\,40243\,10421\,59335\,93992.$$

Однако, если нас интересуют знаки после запятой, то для небольших значений $n$ (порядка миллионов или миллиардов) даже $\ln(n)+\gamma$ все еще является не очень хорошим приближением к $H_n$. Следующая формула для $H_n$

$$H_n = \ln(n) + \gamma + \frac{1}{2n} - \frac{1}{12n^2} + \frac{\varepsilon}{120n^4}$$

где $0 \leq \varepsilon \leq n$, позволяет получить гораздо лучшее приближение.

Как доказал Эйлер, ряд

$$\sum_{i=1}^{\infty} \frac{1}{p_i},$$

где $p_i$ обозначает $i$-е простое число, расходится.

**2.5.** Сравните два следующих кода:

```
N[Sum[1/Prime[i],{i,1,PrimePi[10^n]}]]
Sum[N[1/Prime[i]],{i,1,PrimePi[10^n]}]
```

В чем их отличие? Какой из них дает более надежный результат? Какой дольше вычисляется? А теперь проведите эксперимент для небольших значений $n$, скажем, $n = 5, 6, 7$.

Если гармонический ряд расходится логарифмически, то этот ряд расходится еще гораздо медленнее, примерно как $\ln(\ln(n))$! Иными словами, никто никогда не видел — и в течение ближайшего столетия скорее всего и не увидит — никаких его значений, больших чем 7.

**2.6.** Пронаблюдайте динамику частичных сумм ряда $\sum_{i=1}^{\infty} \frac{1}{p}$ при небольших $n$, скажем $n = m10^4$, $1 \leq m \leq 10$. При каком $n$ эта сумма превзойдет 3?

### 3. Десятичные дроби.[78]

В данном разделе мы изучим запись рациональных чисел так называемыми `бесконечными десятичными дробями`. Как хорошо известно, никаких бесконечных десятичных дробей не существует. То есть, конечно, они существуют как последовательности цифр, но вот только ни складывать, ни

[78] In the Middle Age, in Germany, if you wanted to learn addition and multiplication, you could go to any university. But if you wanted to learn division, you could only do it in one place, Heidelberg. ©Israel Gelfand

умножать их никто не умеет, поэтому вся излагаемая в школьном курсе математики "теория" вещественных чисел, основанная на использовании бесконечных десятичных дробей, абсолютно бессмысленна.

Что, однако, существует, это *конечные* десятичные дроби и *периодические* десятичные дроби, которыми выражаются рациональные числа. Операции над конечными дестяичными дробями сразу сводятся к операциям над целыми числами. В настоящем параграфе мы постараемся придать смысл операциям над периодическими дробями. Дело это называется **теорией сравнений** и обычно изучается не в школьном курсе арифметики, а в университетском курсе теории чисел. Например, в школьном курсе никогда не доказывается, что рациональное число записывается *периодической* дробью. В действительности этот факт составляет содержание **теоремы Эйлера**, которая, кроме того, утверждает, что период несократимой дроби со знаменателем $n$ не превосходит $\varphi(n)$, где $\varphi$ некоторая арифметическая функция, называемая **функцией Эйлера**.

В следующем параграфе мы обсудим, как работают команды `RealDigits` и `FromDigits` для приближенных вещественных чисел. Однако для рациональных чисел их использование имеет некоторую специфику.

| | |
|---|---|
| `RealDigits[x]` | предпериод и период рационального числа $x$ |
| `FromDigits[{list,m}]` | восстановление числа по списку цифр |

Говорят, что число $x$ изображается **периодической** десятичной дробью, если найдется такое натуральное $l$, что $10^l x - x$ выражается конечной десятичной дробью. Это значит, что начиная с некоторого места эта дробь состоит из бесконечно повторяющегося блока цифр длины $l$. Наименьшее такое $l$ называется длиной периода, а сам повторяющийся фрагмент длины $l$ — **периодом** этой дроби. Однако десятичная запись числа не обязана начинаться с периода, ему может предшествовать **предпериод**. Дробь, начинающаяся сразу с периода, называется **чисто периодической**.

Например, дробь $5/12 = 0.416666666\ldots$ имеет предпериод длины 2, и период — длины 1. Как мы уже знаем из предыдущей главы, дробь $1/7 = 0.142857142857142857\ldots$ является чисто периодической и имеет период 142857 длины 6.

Разложение рационального числа $x$ в конечную/периодическую десятичную дробь получается применением к нему команды `RealDigits[x]`. Ответ имеет следующий формат.

• Для числа $x = m/n$, в знаменатель которого входят только степени 2 и 5, ответ имеет вид `{{x1, ... ,xk},r}`, где $x_1, \ldots, x_k$ предпериод (кроме начальных нулей), а $r$ — десятичная экспонента. Иными словами, $10^r \cdot 0.x_1 \ldots x_k$.

• Для любого другого рационального числа $x = m/n$ ответ имеет вид `{{x1, ... ,xk,{y1, ... ,yl}},r}`, где $x_1, \ldots, x_k$ и $r$ имеют тот же смысл, что и раньше, а $y_1, \ldots, y_l$ период десятичной записи $x$.

**3.1.** Убедитесь, что длина периода несократимой дроби $m/n$ равна наименьшему $l$, для которого найдется такое натуральное $k$, что знаменатель $n$ этой дроби делит $10^k(10^l-1)$.

**3.2.** В условиях предыдущей задачи убедитесь, что длина предпериода $m/n$ равна наименьшему $k$ такому, что $n$ делит $10^k(10^l-1)$.

**3.3.** Убедитесь, что длина предпериода и периода правильной дроби $m/n$ не превосходит $\varphi(n)$. Когда здесь достигается равенство?

**4. Египетские дроби.**[79] Древние египтяне представляли рациональное число как сумму целого и нескольких *различных* дробей с числителем равным 1. Такие выражения принято называть **египетскими дробями**. Вот первые интересные примеры египетского разложения правильных дробей:

$$\frac{3}{7}=\frac{1}{3}+\frac{1}{11}+\frac{1}{231},\qquad \frac{5}{7}=\frac{1}{2}+\frac{1}{5}+\frac{1}{70},\qquad \frac{6}{7}=\frac{1}{2}+\frac{1}{3}+\frac{1}{42}.$$

Следующая задача содержится в знаменитом папирусе Райнда[80], хранящемся в Британском музее.

**4.1.** Найдите натуральное $n$ такое, что

$$\frac{2}{73}=\frac{1}{60}+\frac{1}{219}+\frac{1}{292}+\frac{1}{n}.$$

Каждое рациональное число может быть бесконечным числом способов записано как египетская дробь, но для любого $m$ среди этих записей лишь конечное число состоит ровно из $m$ слагаемых[81]. Сейчас мы попробуем описать несколько алгоритмов разложения рационального числа в египетскую дробь.

Проще всего реализовать **жадный алгоритм**[82]. Так называется алгоритм, выбирающий на каждом шаге *наибольшее* слагаемое, которое может войти в египетскую дробь.

Для реализации этого алгоритма нам понадобится важная вспомогательная функция, сопоставляющая списку его **список разностей**.

**4.2.** Определите функцию, которая по списку длины $n$ порождает список длины $n-1$, состоящий из попарных разностей соседних членов исходного списка.

**Решение.** Вот совсем топорное решение:

```
differences[x_]:=Table[x[[i]]-x[[i+1]],{i,1,Length[x]-1}]
```

---

[79] Человек не хуже муравья может переносить тяжести в 20 раз больше собственного веса. Но за большее количество раз, и страшно матерясь. ©Николай Фоменко

[80] A.B.Chace, H.P.Manning, R.C.Archibald, The Rhind mathematical papyrus. vol.I. — Berlin, 1929.

[81] I.Stewart. The riddle of the vanishing camel. Scientific American, June 1992, p.122–124.

[82] S.Wagon. `Mathematica` in Action — Springer-Verlag, 1999.

Зная, что арифметические операции распределяются по спискам, можно дать гораздо более изящную конструкцию:

```
differences[x_]:=Most[x]-Rest[x]
```

Из общих соображений кажется, что этот код улучшить невозможно, но в действительности следующее определение еще лучше и исполняется чуть быстрее:

```
differences[x_]:=Apply[Subtract,Partition[x,2,1],{1}]
```

**4.3.** Определите функцию, которая сопоставляет рациональному числу $x$ его разность с наибольшим египетским слагаемым.

**Решение.** Если число целое или его числитель равен 1, то остаток равен 0, если оно рациональное вне отрезка $[0,1]$, то нужно вычесть целую часть. Остается понять, что в случае, когда $x \in (0,1)$ из него нужно вычесть $1/\lceil 1/x \rceil$. Все это можно резюмировать так:

```
greedypart[x_]:=Which[IntegerQ[x],0,Numerator[x]==1,0,
            x<0||x>1,x-Floor[x],True,x-1/Ceiling[1/x]]
```

При предыдущих условиях `Ceiling[1/x]` совпадает с `1+Floor[1/x]` или, что то же самое с `1+Quotient[1,x]`

**4.4.** Закончите программу разложения рационального числа в египетскую дробь по жадному алгоритму.

**Решение.** Ну, например, так:

```
greedylist[x_]:=Most[FixedPointList[greedypart,x]]
greedyegypt[x_]:=differences[greedylist[x]]
```

К сожалению, даже в случае небольших числителей и знаменателей жадный алгоритм часто дает *очень* плохие разложения. Так, уже

$$\frac{17}{19} = \frac{1}{2} + \frac{1}{3} + \frac{1}{17} + \frac{1}{388} + \frac{1}{375972}$$

выглядит подозрительно, а при разложении $31/311$ получаются знаменатели, содержащие больше 500 цифр! Хватать все, что подвернется под руку, не всегда является лучшей стратегией. Имеется много других алгоритмов, которые дают более короткие разложения с гораздо меньшими знаменателями.

Часто выгоднее предполагать, что знаменатели с самого начала быстро растут. Вот два классических алгоритма такого рода, гарантирующие, кроме того, единственность представления[83] .

**4.5. Алгоритм Фибоначчи**: любое рациональное число $0 < x < 1$ единственным образом представляется в виде

$$x = \frac{1}{n_1} + \frac{1}{n_2} + \ldots + \frac{1}{n_s},$$

[83]по поводу доказательства см. С.Б.Гашков, В.Н.Чубариков. Арифметика, алгоритмы, сложность вычислений, 2-е изд. – М: "Высшая школа", 2000. 320 с.

где $n_1 \geq 2$ и $n_{i+1} > n_i^2 - n_i$ для всех $i$. Реализуйте алгоритм Фибоначчи.

**4.6. Алгоритм Остроградского**: любое рациональное число $0 < x < 1$ единственным образом представляется в виде

$$x = \frac{1}{n_1} + \frac{1}{n_1 n_2} + \ldots + \frac{1}{n_1 n_2 \ldots n_s},$$

где $n_1 \geq 2$ и $n_{i+1} \geq n_i$ для всех $i$. Реализуйте алгоритм Остроградского.

**4.7.** Всегда ли алгоритм Фибоначчи и алгоритм Остроградского дают один и тот же результат?

**4.8.** Найдите сумму $s$ египетских дробей

$$x = \frac{1}{n_1} + \frac{1}{n_2} + \ldots + \frac{1}{n_s} < 1$$

такую, что между $x$ и 1 нет никаких других сумм $s$ египетских дробей.

**Ответ.** Это сумма первых $s$ членов ряда

$$\frac{1}{2} + \frac{1}{3} + \frac{1}{7} + \frac{1}{43} + \frac{1}{1807} + \ldots,$$

в котором знаменатель каждого члена равен произведению знаменателей всех предыдущих членов $+1$.

**4.9.** Ясно, что

$$\frac{1}{n} = \frac{1}{2n} + \frac{1}{2n},$$

Утверждается, что при $n \geq 2$ египетская дробь $1/n$ представляется как сумма двух *различных* египетских дробей:

$$\frac{1}{n} = \frac{1}{x} + \frac{1}{y}, \qquad x < y.$$

Например,

$$\frac{1}{2} = \frac{1}{3} + \frac{1}{6}, \qquad \frac{1}{3} = \frac{1}{4} + \frac{1}{12}, \qquad \frac{1}{4} = \frac{1}{5} + \frac{1}{20} = \frac{1}{6} + \frac{1}{12}, \qquad \frac{1}{5} = \frac{1}{6} + \frac{1}{30},$$

и так далее. Ясно, что почти все эти разложения являются частными случаями тождества

$$\frac{1}{n} = \frac{1}{n+1} + \frac{1}{n(n+1)},$$

но вот второе разложение для $1/4$ так не получается. Найдите все такие разложения.

Следующая задача, предложенная Эрдешем и Штраусом, обсуждается в классической книге Морделла[84].

[84] L.J.Mordell, Diophantine equations. — Acad. Press, London et al., 1969.

**4.10.** Верно ли, что для любого $n > 1$ уравнение

$$\frac{4}{n} = \frac{1}{x} + \frac{1}{y} + \frac{1}{z},$$

имеет решение в натуральных числах?

**Ответ.** Это верно, по крайней мере для всех $n \leq 10^7$. Кроме того, в цитированной книге Морделла доказано, что это вообще всегда так, за исключением, возможно, случая, когда $n$ простое число сравнимое с 1, 121, 169, 289, 361 или 529 по модулю 840.

**4.11.** Проверьте справедливость утверждения предыдущей задачи для первых нескольких тысяч простых, удовлетворяющих этим сравнениям.

**Указание.** Составляя список при помощи команды `Select` следует ли выбирать числа, удовлетворяющие сравнениям, из списка простых или простые из списка чисел, удовлетворяющих сравнениям?

**4.12.** Верно ли, что для любого $n > 1$ уравнение

$$\frac{5}{n} = \frac{1}{x} + \frac{1}{y} + \frac{1}{z},$$

имеет решение в натуральных числах?

**4.13.** Верно ли, что для любого $m$ существует такое $n(m)$, что для любого $n > n(m)$ уравнение

$$\frac{m}{n} = \frac{1}{x} + \frac{1}{y} + \frac{1}{z},$$

имеет решение в натуральных числах?

**4.14.** Можно ли в предыдущей задаче утверждать, что всегда $n(m) \leq m$?

**5. Числа Бернулли.**

Числа Бернулли $B_n$ естественно возникают во многих теоретико-числовых, алгебраических и комбинаторных результатах, а также в классическом анализе. Вот одно из их простейших определений:

$$\frac{t}{e^t - 1} = \sum_{n=0}^{\infty} B_n \frac{t^n}{n!}$$

Числа $B_n$ были независимо введены Яковом Бернулли и Такакадзу Секи Кова для вычисления сумм $1^m + 2^m + \ldots + n^m$. Сегодня большинство начинающих впервые видят числа Бернулли при вычислении коэффициентов рядов Тэйлора тригонометрических и гиперболических функций. Кроме того, многие алгоритмы используют численные значения $B_n$. Поэтому в системе встроена функция `BernoulliB`, возвращающая числа Бернулли.

| `BernoulliB[n]` | число Бернулли $B_n$ |
|---|---|

**5.1.** Составьте таблицу первых 31 чисел Бернулли. Что Вам сразу бросается в глаза?

**Решение.** Простое вычисление `Table[BernoulliB[n],{n,0,10}]` дает

$$B_0 = 1, \quad B_1 = -\frac{1}{2}, \quad B_2 = \frac{1}{6}, \quad B_3 = 0, \quad B_4 = -\frac{1}{30}, \quad B_5 = 0$$
$$B_6 = \frac{1}{42}, \quad B_7 = 0, \quad B_8 = -\frac{1}{30}, \quad B_9 = 0, \quad B_{10} = \frac{5}{66}$$

Все дальнейшие числа Бернулли $B_{2n+1}$ с нечетными номерами тоже равны 0, поэтому ограничимся значениями чисел $B_{2n}$ с четными номерами.

$$B_{12} = -\frac{691}{2730}, \quad B_{14} = \frac{7}{6}, \quad B_{16} = -\frac{3617}{510}, \quad B_{18} = \frac{43867}{798},$$
$$B_{20} = -\frac{174611}{330}, \quad B_{22} = \frac{854513}{138}, \quad B_{24} = -\frac{236364091}{2730},$$
$$B_{26} = \frac{8553103}{6}, \quad B_{28} = -\frac{23749461029}{870}, \quad B_{30} = \frac{8615841276005}{14322}$$

Трудно не заметить, что знаки чисел Бернулли $B_{2n}$ чередуются в зависимости от четности $n$: числа Бернулли $B_{4n}$ отрицательны, а $B_{4n+2}$ — положительны.

Знаменитая **теорема фон Штаудта** утверждает, что дробная часть чисел $(-1)^n B_{2n}$ чрезвычайно естественно выражается в терминах суммы египетских дробей, зависящих только от $n$. В частности, числа Бернулли рациональны.

**5.2.** Убедитесь, что число $B_{2n}$ является разностью некоторого целого числа и суммы всех египетских дробей вида $1/p$, где $p$ пробегает те простые числа, которые на 1 больше какого-то делителя числа $2n$.

**Ответ.** Ограничимся несколькими небольшими примерами[85]:

$$B_2 = \frac{1}{6} = 1 - \frac{1}{2} - \frac{1}{3}$$

$$B_4 = -\frac{1}{30} = 1 - \frac{1}{2} - \frac{1}{3} - \frac{1}{5}$$

$$B_6 = \frac{1}{42} = 1 - \frac{1}{2} - \frac{1}{3} - \frac{1}{7}$$

$$B_8 = -\frac{1}{30} = 1 - \frac{1}{2} - \frac{1}{3} - \frac{1}{5}$$

$$B_{10} = \frac{5}{66} = 1 - \frac{1}{2} - \frac{1}{3} - \frac{1}{11}$$

$$B_{12} = -\frac{691}{2730} = 1 - \frac{1}{2} - \frac{1}{3} - \frac{1}{5} - \frac{1}{7} - \frac{1}{13}$$

$$B_{14} = \frac{7}{6} = 2 - \frac{1}{2} - \frac{1}{3}$$

$$B_{16} = -\frac{3617}{510} = -6 - \frac{1}{2} - \frac{1}{3} - \frac{1}{5} - \frac{1}{17}$$

$$B_{18} = \frac{43867}{798} = 56 - \frac{1}{2} - \frac{1}{3} - \frac{1}{7} - \frac{1}{19}$$

$$B_{20} = -\frac{174611}{330} = -528 - \frac{1}{2} - \frac{1}{3} - \frac{1}{5} - \frac{1}{11}$$

$$B_{22} = \frac{854513}{138} = 6193 - \frac{1}{2} - \frac{1}{3} - \frac{1}{23}$$

$$B_{24} = -\frac{236364091}{2730} = -86579 - \frac{1}{2} - \frac{1}{3} - \frac{1}{5} - \frac{1}{7} - \frac{1}{11}$$

$$B_{26} = \frac{8553103}{6} = 1425518 - \frac{1}{2} - \frac{1}{3}$$

$$B_{28} = -\frac{23749461029}{870} = -27298230 - \frac{1}{2} - \frac{1}{3} - \frac{1}{5} - \frac{1}{19}$$

$$B_{30} = \frac{8615841276005}{14322} = 601580875 - \frac{1}{2} - \frac{1}{3} - \frac{1}{7} - \frac{1}{11} - \frac{1}{31}$$

O dear Ophelia, I am ill at these numbers.

William Shakespeare, *Hamlet*

[85]Смотри по этому поводу Г.Полиа, Г.Сеге, Задачи и теоремы из анализа, том II. — М., Наука, 1978, с.161.

## § 3. Вещественные числа

Допущение, что над вещественными числами можно производить все операции согласно обычным формальным законам арифметических действий, считалось само собой разумеющимся вплоть до второй половины XIX века. Поэтому мы будем рассматривать тот факт, что обычные правила вычислений приложимы к вещественным числам, как аксиому.

Рихард Курант[86]

В школе математическими понятиями пользуются, не сомневаясь в их законности и очень часто не задаваясь даже вопросом, что же они, собственно, означают. Не зная, что такое вещественные числа, мы, тем не менее умеем с ними обращаться, т.е. складывать их, умножать и сравнивать по величине.

Виктор Петрович Хавин[87]

Что уж говорить о мучениях посредством $\varepsilon/2$ и $\sqrt{\delta}$, которым преподаватели математического анализа подвергают студентов первого курса из чистого садизма, без всякой необходимости, могущей сыграть роль смягчающего обстоятельства.

Анри Лебег[88]

Из несчетности множества самих вещественных чисел следует даже, что НЕ СУЩЕСТВУЕТ ЯЗЫКА, В КОТОРОМ КАЖДОЕ ВЕЩЕСТВЕННОЕ ЧИСЛО ИМЕЛО БЫ ИМЯ. Такая вещь, как, например, бесконечное десятичное разложение, не может, конечно, рассматриваться как *имя* соответствующего вещественного числа, поскольку бесконечное десятичное разложение не может даже быть полностью выписано или включено как часть в какое-нибудь фактически выписанное или произнесенное суждение.

Алонзо Черч[89]

Когда во время **партсобрания** за окном трижды каркает ворона и **члены бюро** незаметно сплевывают через левое плечо или крестят под столом животы, это не проявление суеверия, временно омрачающего высшую форму человеческой деятельности, а искаженное переплетение древних психических феноменов, из которых самым поздним является крестное знамение.

Виктор Пелевин, *Зомбификация*

В настоящем параграфе мы обсуждаем основы вычислений с вещественными числами. Среди прочих тем мы рассматриваем алгебраические и трансцендентные числа, десятичное представление вещественного числа,

---

[86] Р.Курант, Курс дифференциального и интегрального исчисления. — М., Наука, 1967, 704с.

[87] В.П.Хавин, Дифференциальное и интегральное исчисление функций одной вещественной переменной. — Лань, СПб, 1998, 446с.

[88] А.Лебег, Об измерении величин, Изд.2. — Учпедгиз, М., 1960, с.1–204; стр.41.

[89] А.Черч, Введение в математическую логику. т.I. – ИИЛ, М.,1960, с.1–484, стр.354.

непрерывные дроби, основные константы и элементарные функции. Основной пафос систем компьютерной алгебры состоит именно в том, что при помощи них можно производить вычисления бесконечной точности. Однако, в некоторых приложениях, а также при дискретизации вывода (графики или звука), приходится пользоваться приближенными вычислениями, так что мы затрагиваем основные связанные с ними понятия.

**1. Точные вещественные числа.**[90]

В настоящей главе мы не будем пытаться объяснять, что такое вещественное число. Тем более, что как математики мы знаем, что никакого другого определения вещественного числа, кроме как элемента поля $\mathbb{R}$ вещественных чисел, не существует. *Определить* $\mathbb{R}$ совсем просто, это (единственное с точностью до изоморфизма) полное архимедово линейно упорядоченное поле — *построить* $\mathbb{R}$ несколько сложнее. Любое *индивидуальное* трансцендентное число является манифестацией актуальной бесконечности и — в строгом техническом смысле — столь же бесконечно, как множество *всех* рациональных чисел. Никакая математически последовательная трактовка вещественных чисел без признания этого основополагающего факта невозможна.

Излагаемая в школе "конструкция" $\mathbb{R}$ при помощи `бесконечных десятичных дробей` является, в действительности, чистой **профанацией**, так как она не позволяет осмысленным образом определить алгебраические операции над вещественными числами. Попробуйте, например, при помощи десятичных дробей *доказать*, что $2/7+5/7=1$ — не говоря уже про гораздо более сложное равенство $\sqrt{2}\sqrt{3}=\sqrt{6}$. Тот факт, что *невозможно* вычислить десятичное разложение суммы двух чисел по десятичным разложениям слагаемых, был замечен еще Дедекиндом, который считал доказательство равенства $\sqrt{2}\sqrt{3}=\sqrt{6}$ *как вещественных чисел* одним из своих главных математических достижений.

Любая попытка определить сумму двух вещественных чисел $x$ и $y$ по их десятичным разложениям сводится к тому, что эти разложения обрезаются до какого-то порядка, берутся суммы получившихся рациональных приближений, и, наконец, $x+y$ определяется как верхняя грань этих сумм. Ясно, что ЭТА ПРОЦЕДУРА НЕ ЯВЛЯЕТСЯ АЛГОРИТМОМ и ничем не отличается от определения $x+y$ по Дедекинду, как верхней грани множества

$$\{a+b \mid a,b\in\mathbb{Q}, a\le x, b\le y\}.$$

Тьюринг обратил внимание на то, что при этом, вообще говоря, нельзя найти *ни одной* десятичной цифры суммы/произведения, не зная *всех* цифр слагаемых/сомножителей. Иными словами, невозможность использовать бесконечные десятичные дроби для реальных вычислений состоит

[90]Вредно распространение между людьми мыслей о том, что наша жизнь есть произведение вещественных сил и находится в зависимости от этих сил. Но когда такие ложные мысли называются науками и выдаются за святую мудрость человечества, то вред, производимый таким учением, ужасен. ©Лев Толстой, *Путь жизни*

не в том, что нахождение *всех* цифр результата требует бесконечного количества операций, а в том, что уже нахождение *первой* цифры результата может потребовать бесконечного количества операций! В ниже в пункте 3 мы приводим пример, показывающий, что увеличение точности приближенного вычисления на *один* разряд может изменить *любое* количество предшествующих цифр.

На самом деле бесконечными десятичными дробями можно описать $\mathbb{R}$ лишь как упорядоченное множество, но не как поле. Внушаемая школьным курсом уверенность, что вещественное число *является* бесконечной десятичной дробью, представляет собой опасную иллюзию. Любая попытка осмысленным образом ввести операции над бесконечными десятичными дробями приводит к какой-то актуально бесконечной конструкции (Вейерштрасса, Кантора, Дедекинда), в которой иррациональные числа истолковываются как бесконечные множества рациональных чисел, классы таких множеств, или что-то в таком духе. Можно, конечно, пользоваться записью $\pi = 3,1415\ldots$, если понимать, что многоточие здесь не означает *ничего* сверх того, что $3,1415 < \pi < 3,1416$. Запись $\pi \approx 3,1416$ не означает даже этого.

Тем не менее, здесь мы встанем на наивную точку зрения существования каких-то вещественных чисел имеющих какие-то приближения, какие-то `десятичные цифры` — и даже будем производить над этими `десятичными цифрами` какие-то арифметические операции. Разумеется, фактически это значит, что в таких случаях мы будем производить вычисления в **кольце десятичных дробей**

$$\mathbb{Z} < \mathbb{Z}\Big[\frac{1}{10}\Big] = \mathbb{Z}\Big[\frac{1}{2}, \frac{1}{5}\Big] < \mathbb{Q}$$

— иными словами, вычисления с рациональными числами — но **вычисления неограниченной точности**. Конечно, вместо десятичных дробей, мы могли бы с тем же успехом пользоваться двоичными дробями $\mathbb{Z}\Big[\frac{1}{2}\Big]$, как большинство программистов, троичными дробями $\mathbb{Z}\Big[\frac{1}{3}\Big]$, как некоторые математики, или шестидесятиричными дробями $\mathbb{Z}\Big[\frac{1}{60}\Big] = \mathbb{Z}\Big[\frac{1}{2}, \frac{1}{3}, \frac{1}{5}\Big]$ (градусы, минуты, секунды, терции, ... ), как астрономы.

Однако, принципиальным отличием систем компьютерной алгебры от *всех* традиционных математических пакетов является не использование вычислений неограниченной точности, а возможность проводить в них **безошибочные вычисления**, известные также под народным названием **вычислений бесконечной точности**. Так принято называть вычисления, в которых наряду с целыми и рациональными числами используются *точные* вещественные или комплексные числа, и при этом не производится *никаких* округлений и приближений. При этом алгебраические числа трактуются как корни многочленов с целыми коэффициентами, трансцендентные числа — как полиномиальные переменные, а при вычис-

лении значений трансцендентных функций используются только точные функциональные соотношения.

**Точное вещественное число** характеризуется тем, что для него — точно так же, как для целого или рационального числа — как **абсолютная точность** `Accuracy`, так и **относительная точность** `Precision` обе *бесконечны*. Так, вычисление

```
{Accuracy[Pi],Precision[Pi]}
```

даст ответ `{Infinity,Infinity}`.

Например, ввод `Log[3]^Cos[1]` будет интерпретирован как *точное* вещественное число $\log(3)^{\cos(1)}$. Точно так же `Pi`, `E` и `Sqrt[2]` представляют *точные* вещественные числа $\pi$, $e$ и $\sqrt{2}$. По умолчанию *все* вычисления с ними — арифметические, логические, вычисления значений функций, etc. — производятся только с бесконечной точностью.

**1.1.** Что больше, $e^\pi$ или $\pi^e$? Что больше, $\sqrt{2}^{\sqrt{3}}$ или $\sqrt{3}^{\sqrt{2}}$? Больше ли $\log(2)^{\cos(1)}$ и $\log(3)^{\cos(1)}$, чем 1?

**Ответ.** Конечно, можно посмотреть на какое-то количество десятичных цифр, как мы это делаем в следующем параграфе, но еще проще спросить прямо. Ну, скажем, `Log[3]^Cos[1]>1`.

## 2. Приближенные вещественные числа: теория.[91]

В настоящем пункте мы обсудим основные понятия, связанные с **приближенными значениями** вещественных чисел. Стоит иметь в виду, что в системах компьютерных вычислений многие понятия и соглашения, связанные с приближенными вычислениями, радикально отличаются от традиционного численного анализа. Вещественное число $y$ называется **приближением** вещественного числа $x$ с **абсолютной погрешностью**[92] $\varepsilon > 0$, если $|x-y| \le \varepsilon$. Если $y$ — приближение $x$ с абсолютной погрешностью $\varepsilon$, то $y-\varepsilon \le x \le y+\varepsilon$, таким образом, неопределенность $x$ равна $2\varepsilon$. Таким образом, приближение тем лучше, чем меньше $\varepsilon$.

Вещественные числа можно приближать любыми другими вещественными числами, но чаще всего рассматривается приближение рациональными числами. В численном анализе, в отличие от классического анализа и теории чисел, при этом как правило рассматривают не все $\mathbb{Q}$, а лишь кольцо $\mathbb{Z}\left[\frac{1}{10}\right]$ десятичных дробей, которые в этом контексте называются **приближенными вещественными числами**. Каждая такая дробь представля-

---

[91] Q: How many numerical analysts does it take to replace a lightbulb??

A: 3.9967: (after six iterations).

[92] В посконных курсах *методов вычислений* абсолютной погрешностью называлась $|x-y|$, а $\varepsilon$ — *предельной* абсолютной погрешностью. Однако, если $x$ и $y$ являются точными числами, то никакой необходимости вводить еще одно название для абсолютной величины их разности нет. Если же они не являются точными числами, то выражение $|x-y|$ само по себе просто не имеет никакого смысла, смысл можно придать только неравенству $|x-y| \le \varepsilon$.

ется в виде $x = m \cdot 10^{-n}$, где $m$ и $n \geq 0$ целые числа, причем если дополнительно требовать, чтобы $m$ не делилось на 10, то такое представление единственно. Множество

$$\frac{1}{10^n}\mathbb{Z} = \left\{ \frac{m}{10^n} \mid m \in \mathbb{Z} \right\}$$

называется множеством приближенных чисел **абсолютной точности** (`Accuracy`) $n$. Традиционно числа абсолютной точности $n$ записываются с $n$ знаками после запятой — которая в Computer Science называется `десятичной точкой` — даже если эти знаки нули, однако `Mathematica` использует совершенно другое соглашение.

Однако, в большинстве вычислений важна не абсолютная, а **относительная точность** (`Precision`) числа $x$, равная абсолютной точности не самого числа $x$, а его **мантиссы**, т.е. числа $x/10^l$, где $l$ подобрано так, чтобы $1/10 \leq x/10^l < 1$.

Как правило для вещественного числа $x$ существует *единственное* приближенное число $y$ абсолютной точности $n$ такое, что абсолютная погрешность приближения $x$ при помощи $y$ равна $10^{-n}/2$. Отображение $x \mapsto y$ называется **округлением** вещественного числа до $n$ разрядов после запятой. Единственными исключениями являются приближенные числа $x$ абсолютной точности $n+1$, заканчивающиеся на 5. Например, число 0.5 одинаково хорошо приближается как 0, так и 1. В этом случае, чтобы избежать накопления систематической ошибки, используется следующее правило: в качестве округления числа $x$ берется то из чисел точности $n$, последняя цифра которого четна.

Произведение/частное двух чисел фиксированной *абсолютной* точности не обязательно имеют ту же абсолютную точность. Для чисел фиксированной *относительной* точности то же самое верно уже для суммы/разности. Поэтому вместо обычных арифметических операций для приближенных чисел используются **приближенные арифметические операции**, с математической (но не с вычислительной!!!) точки зрения состоящие в следующем:

- выполняется обычная арифметическая операция,
- получившийся результат округляется до нужной точности.

Обозначим так определенные **приближенное сложение** и **приближенное умножение** через $\oplus$ и $\odot$, соответственно. Строго говоря, следовало бы указывать еще и используемую при округлении абсолютную или относительную точность, но уже введение специальных символов $\oplus$ и $\odot$ для приближенных операций является *огромным* прогрессом по сравнению с традиционными текстами по *методам вычислений*, где приближенные операции обозначались теми же знаками $+$ и $\cdot$, что и обычные операции, так что после нескольких страниц становилось вообще непонятно, о чем идет речь.

Математически трудности приближенных вычислений объясняются тем, что операции приближенного сложения и умножения не обладают *никакими* из обычных свойств сложения и умножения. В книге Кнута подробнейшим обсуждаются связанные с этим неожиданности, например, почему при (приближенном) сложении чисел столбиком сверху вниз и снизу вверх *всегда* получаются разные результаты! Отметим еще несколько необычных явлений.

• Для чисел относительной точности 2 выполяется *точное* равенство $10 \oplus 0.1 = 10$. Таким образом, приближенное сложение не обладает свойством сокращения.

• Для чисел абсолютной точности 1 выполняются следующие соотношения:

$$0.1 \cdot 0.1 = 0.2 \cdot 0.2 = 0, \qquad 0.3 \cdot 0.3 = 0.1, \qquad 0.4 \cdot 0.4 = 0.5 \cdot 0.5 = 0.2$$

В частности, произведение двух ненулевых чисел может равняться нулю. Более того, относительно операции приближенного умножения все числа $-1 < x < 1$ фиксированной абсолютной точности нильпотентны, т.е. дают в некоторой степени 0.

• Ассоциативность сложения нарушается — что еще хуже, она может нарушаться сколь угодно сильно!

Подгрузка пакета `ComputerArithmetic` позволяет эмулировать арифметику чисел с плавающей запятой с точностью до 8 значащих цифр, обычную для большинства микрокалькуляторов:

```
<<ComputerArithmetic‘; SetArithmetic[8]
```

— впрочем, следующую задачу можно решить и на обычном микрокалькуляторе.

**2.1.** Приведите примеры нарушения обычных свойств арифметических операций:

• нарушения ассоциативности приближенного умножения для чисел абсолютной точности 8;

• нарушения ассоциативности приближенного сложения для чисел относительной точности 8;

• нарушения дистрибутивности приближенного умножения относительно приближенного сложения для чисел точности 8.

В связи с этим еще раз подчеркнем, что использование приближенных вычислений без контроля погрешностей и, в особенности, использование приближенных вычислений в итеративных процедурах, является грубейшей методологической ошибкой. В подавляющем большинстве случаев использование приближенных вычислений представляет собой атавизм, пагубную привычку, злостный пережиток, закрепившиеся в то время, когда все вычисления производились вручную. При сегодняшнем

уровне вычислительных возможностей почти все практически возникающие вычислительные задачи, в которых фигурируют вещественные числа, могут быть решены точно. За исключением задач, связанных с генерацией графика и звука, правильная стратегия состоит в том, чтобы проводить точные вычисления, не округляя никаких промежуточных результатов — и, тем более, не делая этого многократно!!! Округлять — в случае необходимости — можно только окончательный результат.

**3. Приближенные вещественные числа: практика.**[93]

Посмотрим теперь, как описанные в предыдущем параграфе понятия выражаются в системе `Mathematica`.

| | |
|---|---|
| `Accuracy[x]` | абсолютная точность $x$ |
| `Precision[x]` | относительная точность $x$ |

В первом приближении можно считать, что абсолютная точность `Accuracy` возвращает количество десятичных цифр справа от запятой, в то время как относительная точность `Precision` — общее количество явно заданных десятичных цифр. В действительности, конечно, дело обстоит *гораздо* сложнее, так как внутреннее представление чисел в системе двоичное, так что для двух чисел одинаковой (десятичной) разрядности команды `Accuracy` и `Precision` могут возвращать различные результаты, притом не обязательно даже целые!

Абсолютная точность может быть как больше, так и меньше относительной, в зависимости от того, больше или меньше чем 1 абсолютная величина рассматриваемого числа. Перечислим основные соглашения, связанные с точностью приближенных чисел.

• Любое рациональное число, а также *точные* иррациональные числа, такие как $e$, $\pi$, $\ln(2)$, $\cos(1)$ рассматриваются как имеющие *бесконечную* точность, `infinite precision`.

• Любое число, имеющее явную десятичную точку, и любое *численное* выражение, содержащее хотя бы одно приближенное вещественное число, рассматривается как *приближенное* вещественное число и имеет тип `Real`.

• Приближенное вещественное число может быть либо **числом машинной точности** = `machine precision number`, либо **числом произвольной точности** = `arbitrary precision number`, `variable precision number`. Машинное число имеет 6 отображаемых в ответе десятичных разрядов и еще *около* 10 разрядов дополнительной точности, используемых в промежуточных вычислениях. Таким образом, количество значащих разрядов машинного числа, `MachinePrecision`, приблизительно равно 16, в то же время разрядность числа произвольной точности может быть любой и ограничена только конечностью физического мира (о чем подробнее в следующем параграфе).

---

[93] Finagle's Third Law: In any collection of data, the figure most obviously correct, beyond all need of checking, is the mistake.

Следующие тесты позволяют узнать, рассматривает ли система $x$ как число, как явно заданное число (рациональное или приближенное вещественное), и, наконец, как машинное число.

| | |
|---|---|
| `NumericQ[x]` | $x$ является числом |
| `NumberQ[x]` | $x$ является *явно заданным* числом |
| `MachineNumberQ[x]` | $x$ является машинным числом |

• В силу особенностей архитектуры процессора и шины, а также организации памяти, вычисления с машинными числами часто значительно эффективнее, чем с числами произвольной точности. С другой стороны, получающаяся при этом точность достаточна для большинства обычных приложений, в частности, для генерации графики. Поэтому по умолчанию используемая системой рабочая точность равна машинной точности, `WorkingPrecision->MachinePrecision`.

• Приближенное число с общим количеством знаков, *меньшим* машинной точности `MachinePrecision`, по умолчанию рассматривается как число машинной точности. Иными словами, 0.1 значит то же самое, что 0.10, 0.100, 0.1000 и т.д., вплоть до машинной точности, что находится в кричащем противоречии с практикой классической вычислительной математики! Например, вычисление

`{Accuracy[10^6.+10^-6],Precision[10^6.+10^-6]}`

даст ответ `{9.95459, MachinePrecision}`

• Численные команды — такие как `N`, `NSolve`, `NRoots`, `NSum`, `NProduct`, `NIntegrate`, `NDSolve` и т.д. — имеют параметры или опции, явно декларирующие точность результата и/или промежуточных значений. Кроме того, точность всех чисел, входящих в некоторое выражение, может быть явно задекларирована посредством команды `SetPrecision`, установкой значения опции `WorkingPrecision` и т.д.

Перечислим простейшие способы декларирования точности.

| | |
|---|---|
| `AccuracyGoal` | ожидаемая абсолютная точность результата |
| `PrecisionGoal` | ожидаемая относительная точность результата |
| `SetAccuracy` | декларирование абсолютной точности |
| `SetPrecision` | декларирование относительной точности |
| `WorkingPrecision` | количество цифр, сохраняемых во всех промежуточных результатах |
| `$MaxExtraPrecision` | количество дополнительные разрядов, используемых в промежуточных вычислениях, по умолчанию 50 |

• Точность приближенного числа с количеством знаков, *большим* машинной точности, равна его *декларированной* точности, либо, по умолчанию, количеству явно указанных знаков. При обычных конфигурациях

ядра точность приближенного числа произвольной точности ограничена только объемом оперативной памяти используемого компьютера.

Основная команда округления в системе `Mathematica` это `N`.

| | |
|---|---|
| `N[x]` | приближенное значение $x$ с машинной точностью |
| `N[x,m]` | приближенное значение $x$ с точностью $m$ цифр |

По умолчанию `N` возвращает машинные числа, иными словами, она отображает на экране первые 6 десятичных знаков числа, но при этом все внутренние вычисления производятся примерно с 16 знаками. Скажем, вычисление

```
{N[Sqrt[2]],N[E],N[Pi]}
```

дает

```
{1.41421,2.71828,3.14159}
```

В то же время вычисление `N[Pi,100]` дает первые сто знаков $\pi$.

**3.1.** Сколько времени занимает вычисление первого миллиона знаков $\pi$, $e$, $\sqrt{2}$? Сколько времени занимает вычисление первых ста тысяч знаков $\ln(2)$, $\cos(1)$, $e^{\pi}$, $\pi^2$, $\sqrt{2}^{\sqrt{3}}$?

**Ответ.** Как говорится в `Implementation Notes`, $\pi$ вычисляется по формуле Чудновских, но все равно гораздо медленнее, чем $e$ — ряд Тэйлора для экспоненты хорошо сходится! Но $e$, в свою очередь, вычисляется медленнее, чем $\sqrt{2}$, вычисление которого использует быстро сходящиеся итерационные алгоритмы. Вычисление значений экспонент, логарифмов и тригонометрических функций основано непосредственно на рядах Тэйлора + функциональные соотношения + итерационные процедуры. Для таких значений аргументов скорость вычисления определяется главным образом скоростью сходимости соответствующего ряда.

Следующий курьезный пример, который мы уже упоминали в нашем курсе, показывает, что в приближенных вычислениях *ни одна* из цифр ничего не значит! В данном случае, конечно, причина этого явления хорошо известна специалистам и связана с тем, что фигурирующие здесь числа 43, 67 и 163 возникают как часть составленного Гауссом списка одноклассных мнимых квадратичных полей

$$\mathbb{Q}(\sqrt{d}), \qquad d = -1, -2, -3, -7, -11, -19, -43, -67, -163.$$

Просто для маленьких дискриминантов интересующий нас эффект не так заметен.

**3.2.** Вычислите $e^{\pi\sqrt{43}}$ с точностью до 12 и 13 знаков, $e^{\pi\sqrt{67}}$ с точностью до 17, 18 и 19 знаков, и $e^{\pi\sqrt{163}}$ с точностью до 29, 30 и 31 знака.

**Ответ.** Ограничимся наиболее эффектной частью ответа. Вычисление

```
NumberForm[N[Exp[Pi*Sqrt[163]],29],ExponentFunction->(Null&)]
```

дает

$$262537412640768744.00000000000$$

Напомним, что опция `ExponentFunction->(Null&)` введена здесь только для того, чтобы система не пыталась выразить это число в научной форме, с разделением мантиссы и порядка. То же вычисление с точностью до 30 знаков дает

$$262537412640768743.999999999999$$

где все еще неясно, является ли это число целым на самом деле, или всего лишь с точностью до 12 знаков после запятой. Вычисление следующего разряда снимает все сомнения:

$$262537412640768743.9999999999993$$

Как показывает следующая задача, все ссылки на использование вещественных чисел в естествознании и на их роль в `измерении величин`, о которой говорит Лебег, являются чистой фикцией.

**3.3.** Каждый кристаллограф знает[94], что $\pi/6 = 0,52$, $\sqrt{3}\pi/8 = 0,68$ и $\sqrt{2}\pi/6 = 0,74$. Найдите $\pi$, $\sqrt{3}$ и $\sqrt{2}$.

**Ответ.** Из первого равенства находим $\pi = 3.12$, что хорошо согласуется с теоретическим значением. Подставляя это значение во второе равенство, получаем, что в кристаллографии $\sqrt{3} = 1.74359$, а $\sqrt{2} = 1.42308$.

Следующие две задачи взяты из замечательной книги Гуго Штейнгауза[95].

**3.4.** Выяснить, верно ли, что

$$x = \sqrt{5 + \sqrt{3 + \sqrt{5 + \sqrt{3 + \ldots}}}} < 3?$$

**3.5.** Выяснить, при любом ли $n$ выполняется неравенство

$$x = \sqrt{1 + \sqrt{2 + \sqrt{3 + \ldots + \sqrt{n}}}} < 2?$$

---

[94]Смотри, например, А.Анималу, Квантовая теория кристаллических твердых тел. — М., Мир, 1981, 574с., где все эти формулы *именно в таком виде* приведены на страницах 15–16 как плотности простой кубической, объемноцентрированной кубической и гранецентрированной кубической упаковок одинаковых шаров!

[95]Г.Штейнгауз, Задачи и размышления. — М., Мир, 1974, 400с.

**4. Машинные числа.**[96]

В нашем учебнике мы почти не упоминаем системные команды. Тем не менее, при некоторых практических вычислениях полезно понимать ограничения, накладываемые используемой операционной системой, а также архитектурой процессора, памятью компьютера и другими подобными внешними по отношению к вычислению обстоятельствами. Перечислим системные команды, которые позволяют узнать параметры приближенных вычислений. Приводимые ниже численные значения системных параметров относятся к системе `Windows` и бытовым компьютерам 2005–2006 годов.

| | |
|---|---|
| `MachinePrecision` | машинная точность, по умолчанию $53\log_{10}(2)$ |
| `$MachinePrecision` | численное значение машинной точности, 15.9546 |
| `$MachineEpsilon` | машинный эпсилон $2^{-52}$ |
| `$MaxMachineNumber` | наибольшее машинное число $2^{1024}$ |
| `$MinMachineNumber` | наименьшее положительное машинное число $2^{-1022}$ |

**Машинный эпсилон** это наименьшее положительное *машинное* число $\varepsilon$ такое, что $1+\varepsilon \neq 1.000000000000000$, где равенство понимается как равенство машинных чисел. Поскольку вычисления в системе производятся в двоичной системе, то фактически `$MachineEpsilon` имеет следующее значение:

$$\texttt{\$MachineEpsilon} = 2^{-52} \approx 2.22045 \cdot 10^{-16}.$$

Машинный эпсилон является абсолютной версией применяемого в теоретической Computer Science **ульпа** = `unit in the last place`. А именно, машинный эпсилон $\varepsilon$ — это в точности ульп между 1 и 2. Ульп между 2 и 4 равен $2\varepsilon$, etc. Теоретически приближенные вычисления могут быть организованы так, чтобы ошибка одной операции не превышала половины ульпа, а ошибка при выполнении нескольких операций — одного ульпа. Однако, по причинам упомянутым в § 2, практически при вычислениях с числами разного масштаба уже выполнение *двух* операций может давать ошибку в миллионы ульпов. Более того, в обычной машинной арифметике контроль ошибки не производится!

Наибольшее машинное число `$MaxMachineNumber` равно

$$\begin{aligned} 2^{1024} = 17976931348623159077293051907890247336179769789423 \\ 06572734300811577326758055009631327084773224075360 \\ 21120113879871393357658789768814416622492847430639 \\ 47412437776789342486548527630221960124609411945308 \\ 29520850057688381506823424628814739131105408272371 \\ 63350510684586298239947245938479716304835356329624 \\ 224137216 \approx 1.79769 \cdot 10^{308}. \end{aligned}$$

[96] Sometimes it is useful to know how large your zero is. — The Tao of Real Programming

С другой стороны, наименьшее положительное машинное `$MinMachineNumber` число равно

$$2^{-1022} \approx 2.22507 \cdot 10^{-308}.$$

Таким образом, операции над машинными числами такие, как сложение и умножение, не только не удовлетворяют обычным свойствам наподобие ассоциативности и коммутативности, но и вообще не являются всюду определенными алгебраическими операциями!

Фактически, однако, в силу физических ограничений, накладываемых размером оперативной памяти используемого компьютера, среди чисел произвольной точности тоже есть наибольшее и наименьшее число. Узнать, чему именно они равны для Вашей системы, можно при помощи системных команд `$MaxNumber` и `$MinNumber`.

| | |
|---|---|
| `$MaxNumber` | наибольшее число произвольное точности |
| `$MinNumber` | наименьшее положительное число произвольное точности |

Вот типичные значения наибольшего и наименьшего чисел произвольной точности:

$$\texttt{\$MaxNumber} = 1.605216761933662 \cdot 10^{1355718576299609}$$

$$\texttt{\$MinNumber} = 6.229688249675322 \cdot 10^{-1355718576299610}$$

Это значит, что выполнение алгебраических операций над числами произвольной точности тоже может привести к переполнению. Например, $2^{10^9}$ можно вычислить точно, в то же время попытка вычислить $2^{2 \cdot 10^9}$ приводит к переполнению.

**5. Десятичные цифры.**[97]

Основными функциями для работы с приближенными вещественными числами являются `RealDigits` и обратная к ней функция `FromDigits`. В предыдущей главе мы уже обсуждали, как эти функции работают для рациональных чисел.

| | |
|---|---|
| `RealDigits[x]` | список цифр вещественного числа $x$ |
| `FromDigits[{list,m}]` | восстановление числа по списку цифр |

Для приближенных вещественных чисел (=десятичных дробей) эти команды в целом действуют без всяких неожиданностей. Однако, непосредственная попытка вычислить `RealDigits` от *точного* иррационального числа – как алгебраического `RealDigits[Sqrt[2]]`, так и трансцендетного `RealDigits[Pi]` – приведет к сообщению об ошибке:

```
RealDigits: The number of digits to return cannot be determined.
```

[97] Так, поле “Номер паспорта”, 6 цифр. Это до запятой или после? ©Николай Фоменко

В отличие от команды `N` нельзя и спросить `RealDigits[Pi,20]`, так как это будет истолковано не как пожелание определить 20 десятичных знаков числа $\pi$, а как пожелание найти *все* знаки числа $\pi$ в двадцатиричной системе.

Правильное обращение к этой команде таково: нужно указать число, десятичные цифры которого мы хотим найти, основание системы счисления и затем количество цифр, которые мы хотим найти. Так, вычисление

```
RealDigits[Pi,10,25]
```

даст первые 25 десятичных цифр числа $\pi$ и позицию запятой в этом числе:

```
{{3,1,4,1,5,9,2,6,5,3,5,8,9,7,9,3,2,3,8,4,6,2,6,4,3},1}
```

Следует, впрочем, иметь в виду, что если мы требуем больше десятичных цифр, чем точность числа $x$, то недостающие цифры заменяются символом `Indeterminate`. Например, вычисление

```
RealDigits[N[Pi,15],10,18]
```

вернет

```
{{3,1,4,1,5,9,2,6,5,3,5,8,9,7,9,3,Indeterminate,Indeterminate},1}
```

**5.1.** Найдите, сколько раз каждая из цифр $0, 1, 2, \ldots, 9$ встречается среди первой тысячи десятичных знаков $\pi$.

**Решение.** Например, так

```
Map[Count[First[RealDigits[Pi,10,1000]],#]&,Range[0,9]]
```

**5.2.** Какая цифра чаще всего встречается среди первых 10000 десятичных знаков $e$?

**5.3.** Найдите наименьшее $n$ такое, что среди первых $n$ знаков $\pi$ встречается каждая из цифр $0, 1, 2, \ldots, 9$.

**5.4.** Найдите первое вхождение в десятичное разложение $\pi$ двух цифр 3 подряд, трех цифр 3 подряд, четырех цифр 3 подряд.

**5.5.** Найдите цифру, встречающуюся среди первых 100000 десятичных знаков $\pi$ ровно шесть раз подряд.

**5.6.** Существует ли цифра, которая встречается среди первых 1000000 десятичных знаков $\pi$ ровно семь раз подряд?

Следующая задача состоит главным образом в том, чтобы понять, что именно в ней спрашивается.

**5.7.** Если написать десятичные цифры чисел $e$ и $\pi$ в обратном порядке, какое из получившихся чисел будет больше?

**6. Алгебраические числа.**

На наивном уровне вещественные числа делятся на рациональные и иррациональные, При этом иррациональность $\sqrt{2}$ выдвигается в качестве мотивации для введения вещественных чисел. Это *полная ерунда*, так как этим мотивируется лишь необходимость введения *алгебраических* чисел. С точки зрения профессионального алгебраиста вычисления с $\sqrt{2}$ ничуть не сложнее, чем с $1/3$. Дело в том, что число $\sqrt{2}$ алгебраическое, даже *целое*

алгебраическое, притом степени 2, так что вычисления с ним сводятся к вычислениям с *целочисленными* матрицами степени 2.

Как с принципиальной, так и с вычислительной точки зрения действительно судьбоносное различие проходит не между рациональными и иррациональными числами, а между алгебраическими и трансцендентными. Напомним, что (комплексное) число $x$ называется **алгебраическим**, если оно является корнем алгебраического уравнения

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0 = 0, \qquad a_i \in \mathbb{Z},$$

для некоторого многолена $f$ с целыми коэффициентами. Если существует такое уравнение со старшим коэффициентом 1, то число $x$ называется **целым алгебраическим** — или, как говорят профессионалы, просто **целым** (при этом числа целые в обычном смысле называются *целыми рациональными*). Целочисленный многочлен $f$ наименьшей возможной степени такой, что $f(x) = 0$, называется **минимальным многочленом** $x$, а его степень — степенью алгебраического числа $x$. Вычисления с индивидуальным целым алгебраическим числом степени $d$ не сложнее, чем вычисления с $d$ обычными целыми числами.

Множество всех алгебраических чисел образует поле $\overline{\mathbb{Q}}$, а множество *целых* алгебраических чисел — его подкольцо $\mathbb{A}$. Заметим, впрочем, что в документации к системе `Mathematica` через $\mathbb{A}$ обозначается домен *всех* алгебраических чисел. Гаусс заметил, что число, которое одновременно является целым и рациональным, действительно целое рациональное: $\mathbb{A} \cap \mathbb{Q} = \mathbb{Z}$.

Число $x$, которое не является алгебраическим, называется **трансцендентным**. Трансцендентное число не удовлетворяет вообще никаким нетривиальным алгебраическим уравнениям и, таким образом, может быть принято за независимую полиномиальную переменную. Конечно, фактически системы компьютерной алгебры именно так и поступают, это значит, что вычисления с индивидуальным трансцендентным числом имеют ту же сложность, что вычисления с целочисленными многочленами.

Долгое время не было известно, существуют ли трансцендентные числа, и многие математики верили, что все вещественные числа являются алгебраическими. Лишь в 1851 году Лиувилль построил первые примеры трансцендентных чисел. Ясно, что до этих примеров не было никакой необходимости в формальном введении вещественного числа, и действительно первые конструкции вещественных чисел начали появляться лишь в 1860-е годы — ровно через три века после того, как были введены комплексные числа и через полвека после того, как они были формально построены! Одновременно с этим была доказана трансцендентность двух самых знаменитых констант: в 1873 Эрмит доказал трансцендентность $e$, а в 1882 году фон Линдеманн доказал трансцендентность $\pi$, полностью решив, тем самым, классическую проблему “квадратуры круга”. С другой стороны, при помощи совершенно других соображений Кантор доказал, что алгебраические числа встречаются крайне редко, *почти все* вещественные числа

являются трансцендентными. В то же время даже сегодня доказательство трансцендентности индивидуального числа часто остается чрезвычайно сложной задачей.

Упомянем некоторые определенные в ядре системы команды, позволяющие работать с алгебраическими числами.

| | |
|---|---|
| `Algebraics` | домен алгебраических чисел |
| `Root[f,m]` | $m$-й корень алгебраического уравнения $f(x)=0$ |
| `RootReduce[x]` | минимальный многочлен $x$ |
| `ToRadicals[x]` | преобразование $x$ к радикалам |

Имя домена `Algebraics` используется обычным образом, вопрос в формате `Element[x,Algebraics]` дает ответ `True`, если число $x$ алгебраическое и `False` в противном случае. Вычисление

```
Map[Element[#,Algebraics]&,{Sqrt[2],2^Sqrt[2],Sqrt[2]^Sqrt[3]}]
```

показывает, что $\sqrt{2}$ алгебраическое число, в то время как $2^{\sqrt{2}}$ и $\sqrt{2}^{sqrt3}$ — нет. Система *знает*, что $e$, $\pi$ и $e^\pi$ не являются алгебраическими, но вот вопрос о том, будут ли $e+\pi$ и $e\pi$ алгебраическими, ставит ее в тупик.

Ясно, что любое число, скомбинированное из рациональных чисел при помощи арифметических операций и извлечения корней, является алгебраическим. Однако, далеко не любое алгебраическое число имеет такой вид. Обычно, самым простым описанием алгебраического числа является его описание в терминах минимального многочлена. В ядре `Mathematica` для этого используются объекты типа `Root`, дополнительные возможности описаны в пакетах расширений.

В простейших случаях команда `RootReduce` пытается преобразовать выражение к явной комбинации радикалов. Если это невозможно, либо если ответ в терминах радикалов слишком сложен, вычисление `RootReduce[x]` преобразует $x$ к единственному объекту формата `Root` — иными словами, ищет минимальный многочлен $x$. Например, вычисление

```
RootReduce[Sqrt[2]+Sqrt[3]]
```

дает

```
Root[1-10#1^2+#1^4&,4]
```

иными словами, минимальный многочлен $\sqrt{2}+\sqrt{3}$ равен $x^4-10x^2+1$. Интересно, что решение уравнения `Solve[x^4-10x^2+1==0,x]` возвращает ответ в форме

$$-\sqrt{5-2\sqrt{6}},\quad \sqrt{5-2\sqrt{6}},\quad -\sqrt{5+2\sqrt{6}},\quad \sqrt{5+2\sqrt{6}},$$

и только применение `FullSimplify` возвращает эти корни к виду $\pm\sqrt{2}\pm\sqrt{3}$.

**6.1.** Найдите минимальный многочлен $\sqrt{2}+\sqrt{3}+\sqrt{5}$.

**6.2.** Найдите минимальный многочлен $\sqrt{1+\sqrt{1+\sqrt{2}}}$.

**6.3.** Найдите минимальный многочлен $\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{2}}}}$.

**6.4.** Найдите минимальный многочлен $\sqrt{2+\sqrt{3+\sqrt{5}}}$.

**6.5.** Найдите минимальный многочлен $\sqrt[3]{2}+\sqrt{3}$.

**6.6.** Найдите минимальный многочлен $\sqrt[3]{2}+\sqrt[3]{3}$.

Команда `ToRadicals` пытается — в тех случаях, когда это возможно — преобразовать выражение алгебраического числа как комбинации корней алгебраических уравнений к явному выражению в радикалах даже если она считает, что его описание как корня алгебраического уравнения проще. Как мы уже упоминали, в большинстве случаев сама по себе система предпочитает описание в терминах минимальных многочленов. Дело в том, что в общем случае непосредственное преобразование вложенных радикалов и даже проверка того, что два выражения, содержащие вложенные радикалы, равны, представляет собой совсем непростую задачу. С другой стороны, совпадение минимальных многочленов проверяется очень легко.

**7. Основные константы.**[98]

В системе имплементированы основные математические константы. Вот наибоее известные из них.

| | | |
|---|---|---|
| `E` | $e$ | основание натурального логарифма |
| `Pi` | $\pi$ | длина окружности диаметра 1 |
| `Degree` | $1^0$ | $\pi/180$, градус |
| `GoldenRatio` | $\phi = \dfrac{1+\sqrt{5}}{2}$ | золотое сечение |
| `EulerGamma` | $\gamma$ | константа Эйлера |
| `Catalan` | $C$ | константа Каталана |

**7.1.** В 1674 году Лейбниц открыл следующую формулу:

$$1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\ldots=\frac{\pi}{4}.$$

Проверьте ее.

**Ответ.** Для этого можно просто беззастенчиво набрать

```
Sum[(-1)^i*1/(2*i+1),{i,0,Infinity}]
```

---

[98] Mathematician: $\pi$ is the ratio of the circumference of a circle to its diameter.

Engineer: $\pi$ is about 22/7.

Physicist: $\pi$ is 3.14159 plus or minus 0.000005

Computer Programmer: $\pi$ is 3.141592653589 in double precision.

Nutritionist: $\pi$ is a healthy and delicious dessert!

но обычно при суммировании по арифметической последовательности удобнее не пересчитывать вид слагаемых, а надлежащим образом задавать вид итератора:

```
Sum[(-1)^(i/2-1/2)/i,{i,1,Infinity,2}]
```

**7.2.** В 1748 году Эйлер открыл следующую формулу:

$$1+\frac{1}{2^2}+\frac{1}{3^2}+\frac{1}{4^2}+\ldots=\frac{\pi^2}{6}$$

и много других подобных формул, в частности,

$$1-\frac{1}{2^2}+\frac{1}{3^2}-\frac{1}{4^2}+\ldots=\frac{\pi^2}{12},\qquad 1+\frac{1}{3^2}+\frac{1}{5^2}+\frac{1}{7^2}+\ldots=\frac{\pi^2}{8}.$$

Проверьте эти формулы и найдите еще несколько таких же.

**Указание.** Что можно менять в этих формулах? Заметим, что чередование знака в последней формуле даст нам формулу для одной важнейшей константы — константы Каталана:

$$1-\frac{1}{3^2}+\frac{1}{5^2}-\frac{1}{7^2}+\ldots=-C.$$

**7.3.** Стали ли бы Вы использовать какую-либо из формул, полученных в двух предыдущих задачах, для фактического вычисления $\pi$. Если да, то какую и почему?

В качестве ответа предложим следующее уточнение предыдущей задачи.

**7.4.** Проверьте, сколько верных знаков $\pi$ получается при суммировании первых $10000\cdot n$, $1\le n\le 10$ членов ряда Лейбница и сколько времени это занимает.

**Ответ.** Вот $\pi$ с точностью до 50 знаков *после запятой*, полученное с помощью `N[Pi,51]`:

3.14159 26535 89793 23846 26433 83279 50288 41971 69399 37511

С другой стороны, выполнение команды

```
Table[N[Sum[4*(-1)^(i-1)/(2i-1),{i,1,10000*n}],51],{n,1,10}]
```

возвращает следующую таблицу:

3.14149 26535 90043 23845 95183 83374 81537 87870 13642 74418
3.14154 26535 89824 48846 25457 27030 24751 30928 52688 19023
3.14155 93202 56469 16438 85564 49123 16786 47638 81274 79068
3.14156 76535 89797 14471 26403 31521 69620 16104 78839 39197
3.14157 26535 89795 23846 26423 83279 50410 41971 66629 37512
3.14157 59869 23127 72920 33837 22142 67194 89566 13878 53421
3.14157 83678 75508 25303 99026 72563 72981 01894 97534 74709
3.14158 01535 89793 72674 38932 87912 07128 90207 11000 30165
3.14158 15424 78682 47028 70603 39959 54829 01599 71987 72723
3.14158 26535 89793 48846 26433 52029 50289 37284 19393 96495

Обратите внимание, что суммирование первых 10000 членов ряда все еще дает ошибку в четвертом знаке после запятой, и даже суммирование первых 100000 членов все еще дает неправильный пятый знак! Ясно, что использовать такую формулу для вычислений невозможно. Что, однако, гораздо интереснее, после ошибки в четвертом знаке сумма дает 6 верных знаков, а последняя — целых 10 верных знаков! Некоторые из остальных сумм дают 7, 8 или 9 верных знаков после одного, двух или трех неверных. Концептуальное объяснение этого удивительного явления предложено в[99].

**7.5.** Проверьте формулу Валлиса

$$\frac{2}{1}\cdot\frac{2}{3}\cdot\frac{4}{3}\cdot\frac{4}{5}\cdot\frac{6}{5}\cdot\frac{6}{7}\cdot\ldots=\frac{\pi}{2}$$

**Решение.** Нужно просто понять, как ввести это произведение. Классическая запись такова:

```
Product[(2*i)*(2*i)/((2*i-1)*(2*i+1)),i,1,Infinity]
```

**7.6.** В 1593 году Франсуа Виет открыл следующую замечательную формулу

$$\frac{2}{\pi}=\frac{\sqrt{2}}{2}\cdot\frac{\sqrt{2+\sqrt{2}}}{2}\cdot\frac{\sqrt{2+\sqrt{2+\sqrt{2}}}}{2}\cdot\ldots$$

Удастся ли Вам проверить ее тем же способом, что предыдущие?

**7.7.** Чтобы понять, откуда берется формула Виета, вычислите $\cos(\pi/2^n)$.

**8. Элементарные функции.**

Как известно, в `Mathematica` все основные элементарные функции имеют обычные английские имена.

| | |
|---|---|
| `Exp[x]` | экспонента $x$ |
| `Log[x]` | логарифм $x$ |
| `Log[b,x]` | логарифм $x$ по основанию $b$ |

Экспонента и логарифм, также как тригонометрические и гиперболические функции, обратные тригонометрические и гиперболические функции и специальные функции, трактуются как **числовые функции** и мы рекомендуем по этому поводу еще раз перечитать соответствующие параграфы Главы 6.

**8.1.** (Р.В. Хэмминг)[100] Путем численного эксперимента — либо построения графиков — убедитесь, что *для всех практических целей*

$$\log_2(x)=\ln(x)+\log_{10}(x).$$

[99] J.M.Borwein, P.B.Borwein, K.Dolcher, Pi, Euler numbers, and asymptotic expansions. — Amer. Math. Monthly, 1989, vol.96, p.681–687.

[100] см. 1.2.2.22 в книге Кнут Д. Э. Искусство программирования. Том 1: Основные алгоритмы пер. с англ., ред. Козаченко ЛФ, Тертышного ВТ, Красикова ИВ-3-е изд. – 2000.

А теперь объясните это явление.

Также и основные тригонометрические функции имеют обычные в английской математической литературе имена.

| | |
|---|---|
| `Cos[x]` | косинус $x$ |
| `Sin[x]` | синус $x$ |
| `Tan[x]` | тангенс $x$ |
| `Cot[x]` | котангенс $x$ |
| `Sec[x]` | секанс $x$ |
| `Csc[x]` | косеканс $x$ |

Мы не будем упражняться в выполнении тригонометрических преобразований, эта тема упоминается в Главе 6. Ограничимся лишь несколькими забавными примерами *точных* вычислений со значениями тригонометрических функций. В качестве разминки начнем с совсем простеньких задачек.

**8.2.** Из школьного курса тригонометрии известно, что $2\cos(\pi/4) = \sqrt{2}$. Вычислите $2\cos(\pi/8)$, $2\cos(\pi/16)$, $2\cos(\pi/32)$ и $2\cos(\pi/64)$.

**Решение.** Да чего уж там,

```
Map[FunctionExpand,Table[2*Cos[Pi/2^n],{n,3,6}]]
```

Применение `FunctionExpand` необходимо, так как система сама по себе, разумеется не будет преобразовывать точное значение $\cos(\pi/2^n)$ в радикалы. Ответ выглядит примерно так:

$$\sqrt{2+\sqrt{2}},\quad \sqrt{2+\sqrt{2+\sqrt{2}}},\quad \sqrt{2+\sqrt{2+\sqrt{2+\sqrt{2}}}},$$
$$\sqrt{2+\sqrt{2+\sqrt{2+\sqrt{2+\sqrt{2}}}}}.$$

**8.3.** Упростите $\dfrac{1}{2\sin(10^o)} - 2\sin(70^o)$.

**Решение.** Это выражение равно 1. Вычисление

```
TrigReduce[1/(2*Sin[10*Degree])-2*Sin[70*Degree]]
```

дает результат

```
(1/2)*(-4*Cos[20*Degree] + Csc[10*Degree])
```

не более простой, чем чем исходное выражение. Но вот вычисление

```
Simplify[1/(2*Sin[10*Degree])-2*Sin[70*Degree]]
```

приводит к полному успеху.

Следующие три задачи взяты из книги[101].

[101] Ч.Тригг, Задачи без изюминки. — М., Мир, 2000, с.1–276.

**8.4.** Вычислите сумму косинусов

$$\cos(5^o) + \cos(77^o) + \cos(149^o) + \cos(221^o) + \cos(293^o).$$

**8.5.** Вычислите разность

$$\operatorname{tg}(117^o) + \operatorname{tg}(118^o) + \operatorname{tg}(125^o) - \operatorname{tg}(117^o)\operatorname{tg}(118^o)\operatorname{tg}(125^o).$$

**8.6.** Вычислите произведение косинусов

$$\cos\left(\frac{\pi}{15}\right)\cos\left(\frac{2\pi}{15}\right)\cos\left(\frac{3\pi}{15}\right)\cos\left(\frac{4\pi}{15}\right)\cos\left(\frac{5\pi}{15}\right)\cos\left(\frac{6\pi}{15}\right)\cos\left(\frac{7\pi}{15}\right)$$

## § 4. Комплексные числа

It is the complex case that is easier to deal with.

Cambridge Mathematical Quotes

I met a man once who told me that far from believing in the square root of minus one, he didn't believe in minus one. This is at any rate a consistent attitude[102].

Edward Titchmarsh

Ни один факт вещественного анализа невозможно понять, оставаясь в области вещественных чисел. В настоящей главе мы обсудим основы вычислений с комплексными числами. Комплексные числа, которые были введены итальянскими алгебраистами в XVI веке как игрушка для математических турниров, без всякой видимой практической цели, в XIX–XX веках превратились в основной инструмент математического естествознания. Комплексные числа играют такую же роль в квантовой механике, как вещественные числа в классической. Вся их история подтверждает мысль Харди о том, что чистая математика *ощутимо* полезнее прикладной.

### 1. Комплексные числа.[103]

Напомним, что алгебраической формой комплексного числа называется его запись в виде $a + bi$, где $a, b \in \mathbb{R}$ — вещественные числа, а $i$ — мнимая единица, $i^2 = -1$. При этом $a$ обычно называется **вещественной частью** $z$, и обозначается $\operatorname{re}(z)$, а $b$ — **мнимой частью** $z$ и обозначается $\operatorname{im}(z)$. Числа, у которых $\operatorname{im}(z) = 0$, называются **вещественными**, а те, у которых $\operatorname{im}(z) \neq 0$ — **мнимыми**. Числа, у которых $\operatorname{re}(z) = 0$ называются **чисто**

[102] В русском переводе книге Г.С.М.Коксетера *Введение в геометрию* эта фраза передана следующим образом: "Я недавно встретил человека, который сказал мне, что он не верит даже в существование $-1$, так как из этого следует существование квадратного корня из нее". Мы предоставляем читателю самостоятельно оценить модальности и еще раз задуматься над *трудностями перевода*.

[103] Life is complex. It has real and imaginary components.©Tom Potter

**мнимыми**. При сложении комплексных чисел отдельно складываются их вещественные и мнимые части,

$$\mathrm{re}(z+w) = \mathrm{re}(z) + \mathrm{re}(w), \qquad \mathrm{im}(z+w) = \mathrm{im}(z) + \mathrm{im}(w),$$

а умножение производится по следующей формуле:

$$(a+bi)(c+di) = (ac-bd) + (ad+bc)i.$$

Комплексное число $\overline{z} = a - bi$ называется **сопряженным** к $z = a + bi$. При этом $z + \overline{z} = 2a$ и $z\overline{z} = a^2 + b^2$ оба вещественные. Любое ненулевое комплексное число $z$ обратимо, при этом $z^{-1} = \overline{z}/(z\overline{z})$.

Использование функций, связанных с комплексными числами, ясно из названия.

| | | |
|---|---|---|
| `I` | $i$ | мнимая единица $i$ |
| `z=x+I*y` | $z = x + iy$ | комплексное число $z$, где $x, y \in \mathbb{R}$ |
| `Re[z]` | $\mathrm{re}(z)$ | вещественная часть $z$ |
| `Im[z]` | $\mathrm{im}(z)$ | мнимая часть $z$ |
| `Conjugate[z]` | $\overline{z} = x - iy$ | сопряженное к $z$ |
| `ComplexExpand[z]` | | выделение $\mathrm{re}(z)$ и $\mathrm{im}(z)$ |

Единственный момент, на который стоит обратить внимание, состоит в том, что автоматически операции над комплексными числами, содержащими символы, не исполняются. Чтобы фактически провести вычисление, нужно применять команду `ComplexExpand`, которая по умолчанию исходит из того, что *все* неспецифицированные переменные вещественные. Зададим два комплексных числа $z = a + bi$ и $w = c + di$ в алгебраической форме `z=a+b*I`, `w=c+d*I`. Вычисление `z*w` не дает ничего интересного, так как скобки автоматически не раскрываются. Но вот вычисление `ComplexExpand[z*w]` дает

```
(a*c-b*d)+I*(b*c+a*d),
```

так что система действительно считает $a, b, c, d$ вещественными. Применение `ComplexExpand` необходимо и в том случае, когда коэффициенты являются *точными* вещественными числами, наподобие $e$, $\pi$ или $\cos(\pi/7)$.

В ядре `Mathematica` описан домен `Complexes` комплексных чисел, элементы которого имеют тип `Complex`.

| | | |
|---|---|---|
| `Complex` | | тип комплексного числа |
| `Complexes` | $\mathbb{C}$ | домен комплексных чисел |

Так, например, вычисление `Head[1+I]` даст `Complex`, а тест

```
Element[1+I,Complexes]
```

даст результат `True`.

Комплексное число $a+bi$, $a,b\in\mathbb{R}$, можно отождествить с **антициркулянтом** $\begin{pmatrix} a & b \\ -b & a \end{pmatrix}$. Легко проверить, что при этом сложению и умножению комплексных чисел соответствуют обычное сложение и умножение матриц. Для сложения это совсем очевидно, а для умножения получается вот что:

$$\begin{pmatrix} a & b \\ -b & a \end{pmatrix}\begin{pmatrix} c & d \\ -d & c \end{pmatrix}=\begin{pmatrix} ac-bd & ad+bc \\ -ad-bc & ac-bd \end{pmatrix}.$$

**1.1.** Постройте поле $\mathbb{C}$ комплексных чисел при помощи этой конструкции. Что при этом отвечает сопряженному и обратному числам?

При написании программ полезно понимать, что `Mathematica` упорядочивает комплексные числа следующим несколько необычным образом:

• В первую очередь сравниваются вещественные части.

• При равенстве вещественных частей сравниваются *абсолютные значения* мнимых частей.

• Из пары сопряженных комплексных чисел первым указывается то, мнимая часть которого отрицательна.

**1.2.** Не включая компьютера расположите числа

$$1,-1,i,-i,1+i,1-i,-1+i,-1-i,-1+2i,-1-2i,1+2i,1+2i,0$$

в используемом системой порядке.

В следующей задаче

$$\omega=-\frac{1}{2}+i\frac{\sqrt{3}}{2},\qquad \omega^2=\overline{\omega}=-\frac{1}{2}-i\frac{\sqrt{3}}{2}.$$

После чтения § 3 она моментально решается устно, пока же мы предлагаем провести непосредственные вычисления.

**1.3.** Вычислите

• $(a+b)(a+b\omega)(a+b\omega^2)$,

• $(a\omega+b\omega^2)(b\omega+a\omega^2)$,

• $(a+b\omega+c\omega^2)(a+b\omega^2+c\omega)$.

• $(a+b+c)(a+b\omega+c\omega^2)(a+c\omega^2+b\omega)$,

• $(a+b\omega+c\omega^2)^3+(a+b\omega^2+c\omega)^3$.

На самом деле следующая задача предполагает решение в терминах тригонометрической, а не алгебраической формы, но она служит хорошей иллюстрацией того, что многократное альтернированное применение `ComplexExpand` и `Simplify` или `FullSimplify` может несколько раз приводить к новой, все более простой форме.

**1.4.** Вычислите

$$z=\frac{\sqrt{2-\sqrt{3}}+i\sqrt{2+\sqrt{3}}}{\sqrt{\sqrt{2}+1}+i\sqrt{\sqrt{2}-1}}.$$

**Решение.** Устное вычисление модуля и аргумента числителя и знаменателя $z$ показывает, что

$$z = \sqrt[4]{2}\left(\cos\left(\frac{7\pi}{24}\right) + i\sin\left(\frac{7\pi}{24}\right)\right).$$

Разумеется, предлагая подобную задачу в контрольной работе мы подразумеваем именно такое решение. Интересно, однако, сколько времени понадобится, чтобы придти к тому же результату в алгебраической форме. Вычисление `FullSimplify[z]` дает представление $z$ как корня многочлена с рациональными коэффициентами:

```
Root[4-4#1^2+2#1^4-2#1^6+#1^8&,6]
```

Иными словами, утверждается, что $z$ есть корень уравнения $x^8 - 2x^6 + 2x^4 - 4x^2 + 4 = 0$.

Попробуем вначале выделить вещественную и мнимую часть $z$, и только потом упростить, `FullSimplify[ComplexExpand[z]]`. При этом получится уже чуть более явный ответ

$$\sqrt{\frac{1}{2}\left((1+i) - (1-i)\sqrt{3}\right)}$$

Кажется, что не будет никакого вреда, если теперь уже в этом выражении выделить вещественную и мнимую часть, но при этом получается достаточно жуткое выражение, которое мы не решаемся здесь воспроизвести без промежуточного упрощения,

```
Simplify[ComplexExpand[FullSimplify[ComplexExpand[z]]]]
```

дает

$$z = \sqrt[4]{2}\left(i * \cos\left(\frac{5\pi}{24}\right) + \sin\left(\frac{5\pi}{24}\right)\right).$$

Но вот применение `FullSimplify` приводит к полному успеху, а именно, вычисляя

```
FullSimplify[ComplexExpand[FullSimplify[ComplexExpand[z]]]]
```

мы получим $(-1)^{7/24}2^{1/4}$, так что теперь еще одно применение `ComplexExpand` как раз и даст ответ, указанный в самом начале!

До сих пор мы обсуждали вычисления с *точными* комплексными числами. Однако, как и вещественные числа, комплексные числа бывают точными и приближенными. А именно, вещественная и/или мнимая части комплексного числа могут быть *приближенными* вещественными числами — по умолчанию машинными. Важнейший нюанс, который следует иметь в виду, проводя приближенные вычисления, состоит в том, что точность/приближенность вещественной и мнимой части оцениваются отдельно! Так, например, вычисление `Head[1+0*I]` дает `Integer`, а вычисление

`Head[1.+0*I]` — `Real`. Иными словами, система считает, что мнимая часть этих чисел точно равна 0, так что первое из них целое, а второе — приближенное вещественное число. Чтобы указать, что мнимая часть лишь *приближенно* равна 0, нужно явным образом поставить там десятичную точку. Числа `1+0.*I` и `1.+0.*I` оба имеют заголовок `Complex`

**2. Тригонометрическая запись комплексного числа.**[104]

В предыдущем пункте комплексное число $z = a+bi$ рассматривалось как точка вещественной плоскости $(a, b)$, проекции которой на вещественную и мнимую оси равны $a$ и $b$, соответственно. Сложение таким образом истолкованных комплексных чисел совпадает с обычным сложением векторов.

Оказывается, однако, что для умножения комплексных чисел *намного* удобнее пользоваться другой системой координат — полярной. В этой системе положение любой точки на плоскости определяется двумя числами: расстоянием $r$ от начала координат до этой точки и углом $\phi$ между полярной осью и радиусом-вектором данной точки, отсчитываемым в положительном направлении. Положительное вещественное число $r = \sqrt{a^2+b^2}$ называется **модулем** комплексного числа $z = a+bi$ и обозначается $|z|$. Полярный угол $\phi$ точки $(a, b)$, изображающей $z = a+bi$, называется **аргументом** комплексного числа $z$ и обозначается $\arg(z)$. Отметим, что аргумент 0 не определен. Пусть поэтому $|z| \neq 0$. В этом случае $\phi = \arg(z)$ — это такой угол, что

$$\cos(\phi) = \mathrm{re}(z)/|z|, \qquad \sin(\phi) = \mathrm{im}(z)/|z|.$$

Эти равенства определяют угол $\phi$ неоднозначно, лишь с точностью до целого кратного $2\pi$. Среди всех $\phi$ удовлетворяющих этим равенствам найдется единственное в промежутке $[0, 2\pi)$. Традиционно это значение $\phi$ называется **главным значением аргумента** и обозначается $\mathrm{Arg}(z)$.

В системе имплементированы функции, вычисляющие по комплексному числу его модулю и аргумент. Стоит, однако иметь в виде, что возвращаемый функцией `Arg` результат находится между $\pi$ и $-\pi$.

| | | |
|---|---|---|
| `Abs[z]` | $\|z\|$ | модуль $x$ |
| `Arg[z]` | $\arg(z)$ | аргумент $z$ |

**2.1.** Скажите - без помощи компьютера! - чему равен аргумент 0? А потом проверьте.

**Ответ.** Правильно, `Interval[{-Pi,Pi}]`.

**2.2.** Напишите команду, возвращающую главное значение аргумента в промежутке $[0, 2\pi)$.

**Решение.** Ну, хотя бы, так:

[104] If you stick your fingers in the mains, it's not the imaginary component which you will feel.— Cambridge Mathematical Quotes

```
arg[x_]:=If[Arg[x]>=0,Arg[x],Arg[x]+2*Pi]
```

если, конечно, Вы не боитесь сообщения `Possible spelling error`.

Если $r = |z|$ — модуль комплексного числа $z$, а $\phi = \arg(z)$ — его аргумент, то $z$ можно записать в виде:

$$z = a + bi = r \cdot \cos(\phi) + ir \cdot \sin(\phi) = r(\cos(\phi) + i \sin(\phi)),$$

называемом **тригонометрической формой** $z$. Тригонометрическая форма *идеально* согласована с умножением комплексных чисел, а именно, при умножении комплексных чисел их модули перемножаются, а аргументы складываются, $|zw| = |z||w|$, $\arg(zw) = \arg(z) + \arg(w)$. Таким образом, для

$$z = r(\cos(\phi) + i\ \sin(\phi)), \qquad w = s(\cos(\psi) + i\ \sin(\psi)),$$

имеем

$$zw = r(\cos(\phi) + i\ \sin(\phi)) \cdot s(\cos(\psi) + i\ \sin(\psi)) = rs(\cos(\phi + \psi) + i\ \sin(\phi + \psi)),$$

Эта формула, известная под названием **формулы де Муавра**, является одной из самых полезных формул в математике и, вместе с теоремой Пифагора, содержит **всю** школьную тригонометрию. Особенно часто используется такое ее следствие

$$z^n = r^n(\cos(n\phi) + i\ \sin(n\phi)).$$

**2.3.** Докажите формулу де Муавра.

**Решение.** Ну, конечно, просто

```
Simplify[(Cos[x]+I*Sin[x])*(Cos[y]+I*Sin[y])]
```

**2.4.** Докажите, что

$$\left(\frac{1 + i\ \mathrm{tg}(\phi)}{1 - i\ \mathrm{tg}(\phi)}\right)^n = \frac{1 + i\ \mathrm{tg}(n\phi)}{1 - i\ \mathrm{tg}(n\phi)}.$$

Формула же для сложения комплексных чисел в тригонометрической форме чуть сложнее и известна специалистам по оптике и электричеству под народным названием “суперпозиция синхронных скалярных гармонических колебаний”.

**2.5.** Найдите формулу для суммы $z = z_1 + z_2$ двух комплексных чисел

$$z_1 = r_1(\cos(\phi_1) + i\ \sin(\phi_1)), \qquad z_2 = r_2(\cos(\phi_2) + i\ \sin(\phi_2))$$

в тригонометрической форме.

**Ответ.** Модуль $r$ и аргумент $\phi$ числа $z$ определяются формулами:

$$r^2 = r_1^2 + r_2^2 + 2r_1r_2\cos(\phi_2 - \phi_1),$$

$$\operatorname{tg}(\phi) = \frac{r_1 \sin(\phi_1) + r_2 \sin(\phi_2)}{r_1 \cos(\phi_1) + r_2 \cos(\phi_2)}.$$

**2.6.** Обобщите результат предыдущей задачи на любое количество слагаемых.

**3. Корни из 1.**[105]

Важнейшую роль в строении комплексных чисел играют корни из 1. Напомним, что корнем $n$-й степени из 1 называется решение уравнения $x^n = 1$. В тригонометрической форме корни $n$-й степени из 1 выражаются как

$$\varepsilon_k = \cos\left(\frac{2\pi k}{n}\right) + i \sin\left(\frac{2\pi k}{n}\right), \qquad k = 0, \ldots, n-1.$$

Обычно множество всех корней степени $n$ из 1 обозначается $\mu_n$ и называется *группой* корней из 1 степени $n$.

**3.1.** Напишите команду, которая возвращает список корней из 1 степени $n$ в порядке возрастания аргумента.

**Решение.** Чтобы можно было реально показать возникающие ответы, проиллюстрируем это на примере $n = 7$. Наивная попытка написать

```
Solve[x^7==1,x]
```

возвращает следующий ответ:

```
{{x->1},{x->-(-1)^1/7},{x->(-1)^2/7},{x->-(-1)^3/7},
            {x->(-1)^4/7},{x->(-1)^5/7},{x->(-1)^6/7}}
```

Иными словами, команда `Solve` возвращает не корни, а правила замены! Ну, с этим справиться легко, нужно просто применить эти правила к $x$, например, так:

```
ReplaceAll[x,Solve[x^7==1,x]]
```

Теперь вычисление дает нам список корней:

```
{1,-(-1)^1/7},(-1)^2/7,-(-1)^3/7,(-1)^4/7,(-1)^5/7,(-1)^6/7}
```

Однако, эти корни по-прежнему не вычислены! Попробуем выделить в них вещественную и мнимую часть:

```
ComplexExpand[ReplaceAll[x,Solve[x^7==1,x]]]
```

Получающийся при этом ответ уже больше похож на то, что хотелось:

```
{1,-Cos[Pi/7]-I*Sin[Pi/7],Cos[3*Pi/14]+I*Sin[3*Pi/14],
      -Cos[Pi/14]-I*Sin[Pi/14],Cos[Pi/14]-I*Sin[Pi/14],
      -Cos[3*Pi/14]+I*Sin[3*Pi/14],-Cos[Pi/7]+I*Sin[Pi/7]}
```

Однако при этом используется описанный в пункте 1 внутренний порядок на комплексных числах, в то время как гораздо естественнее упорядочивать

---

[105] В одном отношении формула для $\cos(2\pi/17)$ не оставляет сомнения. Прийти к ней в рамках традиционных геометрических идей времени Эвклида невозможно. ©Семен Григорьевич Гиндикин — С.Г.Гиндикин, Рассказы о физиках и математиках. — Наука, М., 1981, с.1–191.

корни из 1 по возрастанию аргумента. Это можно попытаться сделать в терминах определенной в предыдущем пункте функции `arg`:

```
Sort[ComplexExpand[ReplaceAll[x,Solve[x^7==1,x]]],
                                    arg[#1]<arg[#2]&]
```

Мы оставляем читателю возможность побороться с системой, чтобы записать аргументы в привычном виде.

**3.2.** Найдите сумму корней степени $n \geq 2$ из 1.

**Ответ.** Вычислив эту сумму для нескольких первых случаев, мы видим, что она равна 0. Как только ответ сформулирован, он становится очевидным, так как корни из 1 являются корнями многочлена $x^n - 1$, а формула Виета утверждает, что сумма корней этого многочлена равна коэффициенту при $x^{n-1}$.

**3.3.** Найдите сумму попарных произведений корней степени $n \geq 3$ из 1.

Корень из единицы $\varepsilon$ называется **первообразным** или, как теперь принято говорить, **примитивным** корнем степени $n$, если он не является корнем никакой меньшей, чем $n$ степени. Если $\varepsilon$ — первообразный корень из 1 степени $n$, то $\varepsilon^k \neq 1$ для всех $k = 1, \ldots, n-1$ и поэтому $\varepsilon^k \neq \varepsilon^l$ для всех $0 \leq k \neq l \leq n-1$. Таким образом, все числа $\varepsilon^k = 1, \varepsilon, \varepsilon^2, \ldots, \varepsilon^{n-1}$ попарно различны. Это значит, что любой первообразный корень степени $n$ порождает группу $\mu_n$. Корни же степени $n$, не являющиеся первообразными содержатся в какой-то строго меньшей подгруппе $\mu_m$. Корень $\varepsilon_k$ в том и только том случае является первообразным корнем степени $n$, когда $\gcd(k, n) = 1$.

**3.4.** Напишите команду, которая возвращает список *первообразных* корней из 1 степени $n$ в порядке возрастания аргумента.

Решение двух следующих задач предполагает знакомство с арифметическими функциями $\phi$ и $\mu$ - так обозначаются функция Эйлера и функция Мёбиуса.[106]

**3.5.** Найдите сумму первообразных корней степени $n \geq 2$ из 1.

**Ответ.** Эта сумма равна значению функции Мебиуса $\mu(n)$.

**3.6.** Найдите сумму $m$-х степеней первообразных корней степени $n \geq 2$ из 1.

**Ответ.** Положим $d = \gcd(m, n)$. Тогда

$$\sum \varepsilon^m = \frac{\phi(n)}{\phi(n/d)}\mu(n/d),$$

Перейдем теперь к явному вычислению первообразных корней небольших степеней. При этом мы получим несколько формул для значений основных тригонометрических функций, которые почему-то не предлагают заучивать в школьном курсе тригонометрии.

---

[106] Описания и примеры использования этих функций можно найти в разделе Help системы `Mathematica`

**3.7.** Явно найдите все (первообразные) корни из 1 небольших степеней, скажем, $n \leq 30$. В каких случаях эти корни можно найти решая квадратные уравнения?

**Решение.** Прежде всего, чтобы превратить $\cos(2\pi/n)$ и $\sin(2\pi/n)$ в комбинацию радикалов или что-то в таком духе, нужно применить `FunctionExpand`. Однако, получающийся при этом ответ совершенно неудобочитаем, поэтому обычно поверх `FunctionExpand` мы применяем `Simplify`. Следует быть чрезвычайно осторожным с применением `FullSimplify`, так как упрощение чего-нибудь совсем невинного — как $\cos(\pi/7)$ или $\cos(\pi/13)$ — может потребовать времени и либо приводит к столь же нечитаемому ответу, либо возвращает исходное выражение.

Вот те степени, для которых такое вычисление дает явные формулы, использующие только квадратные радикалы:

$$1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 8,\ 10,\ 12,\ 15,\ 16,\ 17,\ 20,\ 24,\ 30.$$

Так как все эти степени являются делителями последних пяти из них, то достаточно привести ответ для 16, 17, 20, 24, 30.

Вот корни степени 16:

$$\begin{array}{cccc}
1 & \dfrac{\sqrt{2+\sqrt{2}}+i\sqrt{2-\sqrt{2}}}{2} & \dfrac{1+i}{\sqrt{2}} & \dfrac{\sqrt{2-\sqrt{2}}+i\sqrt{2+\sqrt{2}}}{2} \\
i & \dfrac{-\sqrt{2-\sqrt{2}}+i\sqrt{2+\sqrt{2}}}{2} & \dfrac{-1+i}{\sqrt{2}} & \dfrac{-\sqrt{2+\sqrt{2}}+i\sqrt{2-\sqrt{2}}}{2} \\
-1 & \dfrac{-\sqrt{2+\sqrt{2}}-i\sqrt{2-\sqrt{2}}}{2} & \dfrac{-1-i}{\sqrt{2}} & \dfrac{-\sqrt{2-\sqrt{2}}-i\sqrt{2+\sqrt{2}}}{2} \\
-i & \dfrac{\sqrt{2-\sqrt{2}}-i\sqrt{2+\sqrt{2}}}{2} & \dfrac{1-i}{\sqrt{2}} & \dfrac{\sqrt{2+\sqrt{2}}-i\sqrt{2-\sqrt{2}}}{2}
\end{array}$$

Чтобы получить корни степени 8, нужно взять каждый второй из них, начиная, естественно, с 1, которая является *нулевым*, а не первым по порядку корнем!

Вот корни степени 20:

$$\begin{array}{cccc}
1 & \dfrac{\sqrt{10+2\sqrt{5}}+i(-1+\sqrt{5})}{4} & \dfrac{1+\sqrt{5}+i\sqrt{10-2\sqrt{5}}}{4} \\
 & \dfrac{\sqrt{10-2\sqrt{5}}+i(1+\sqrt{5})}{4} & \dfrac{-1+\sqrt{5}+i\sqrt{10+2\sqrt{5}}}{4} \\
 & & \\
i & \dfrac{1-\sqrt{5}+i\sqrt{10+2\sqrt{5}}}{4} & \dfrac{-\sqrt{10+2\sqrt{5}}+i(1+\sqrt{5})}{4} \\
 & \dfrac{-1-\sqrt{5}+i\sqrt{10+2\sqrt{5}}}{4} & \dfrac{-\sqrt{10+2\sqrt{5}}+i(-1+\sqrt{5})}{4}
\end{array}$$

$$-1 \quad \frac{-\sqrt{10+2\sqrt5}+i(1-\sqrt5)}{4} \quad \frac{-1+\sqrt5-i\sqrt{10+2\sqrt5}}{4}$$
$$\frac{-\sqrt{10+2\sqrt5}+i(-1-\sqrt5)}{4} \quad \frac{-1+\sqrt5-i\sqrt{10+2\sqrt5}}{4}$$

$$-i \quad \frac{-1+\sqrt5-i\sqrt{10+2\sqrt5}}{4} \quad \frac{\sqrt{10+2\sqrt5}+i(-1-\sqrt5)}{4}$$
$$\frac{1+\sqrt5-i\sqrt{10+2\sqrt5}}{4} \quad \frac{\sqrt{10+2\sqrt5}+i(1-\sqrt5)}{4}$$

Чтобы получить корни степени 10 или 5, нужно просто взять каждый второй или, соответственно, каждый четвертый из них, снова, естественно, начиная с 1.

Вот, наконец, корни степени 24:

$$1 \quad \frac{1+\sqrt3+i(-1+\sqrt3)}{2\sqrt2} \quad \frac{\sqrt3+i}{2}$$
$$\frac{1+i}{\sqrt2} \quad \frac{1+i\sqrt3}{2} \quad \frac{-1+\sqrt3+i(1+\sqrt3)}{2\sqrt2}$$
$$i \quad \frac{1-\sqrt3+i(1+\sqrt3)}{2\sqrt2} \quad \frac{-1+i\sqrt3}{2}$$
$$\frac{-1+i}{\sqrt2} \quad \frac{-\sqrt3+i}{2} \quad \frac{-1-\sqrt3+i(-1+\sqrt3)}{2\sqrt2}$$

$$-1 \quad \frac{-1-\sqrt3+i(1-\sqrt3)}{2\sqrt2} \quad \frac{-\sqrt3-i}{2}$$
$$\frac{-1-i}{\sqrt2} \quad \frac{-1-i\sqrt3}{2} \quad \frac{1-\sqrt3+i(-1-\sqrt3)}{2\sqrt2}$$
$$-i \quad \frac{-1+\sqrt3+i(-1-\sqrt3)}{2\sqrt2} \quad \frac{+1-i\sqrt3}{2}$$
$$\frac{1-i}{\sqrt2} \quad \frac{\sqrt3-i}{2} \quad \frac{1+\sqrt3+i(1-\sqrt3)}{2\sqrt2}$$

Чтобы получить корни степени 12, 6 или 3 нужно просто взять каждый второй, соответственно, каждый четвертый или каждый восьмой из них.

Мы не будем приводить ответы для $n = 17$ и $n = 30$ ввиду их громоздкости. Однако, для ценителей конкретности приведем явную формулу, полученную с использованием ранних версий `Mathematica`:

$$\cos\Big(\frac{2\pi}{17}\Big) = \frac{1}{16}\Big(\sqrt{17}-1+\sqrt{34-2\sqrt{17}}+ \\ 2\sqrt{17+3\sqrt{17}-\sqrt{170+38\sqrt{17}}}\,\Big),$$

Заметим, что это не очень сложное вычисление было проведено семнадцатилетним Гауссом при построении правильного 17-угольника! Отсюда уже совсем легко вывести, что

$$\cos\left(\frac{\pi}{17}\right) = \frac{1}{16}\sqrt{15+\sqrt{17}+\sqrt{34-2\sqrt{17}}\ +} \\ \sqrt{2\left(34+6\sqrt{17}+\sqrt{578-34\sqrt{17}}-\sqrt{34-\sqrt{17}}-8\sqrt{2\left(17+\sqrt{17}\right)}\right)},$$

детали приведены в цитированной книге С.Г.Гиндикина. Оставляем заинтересованному читателю возможность убедиться в справедливости приведенных формул с использованием современной `Mathematica`.

**3.8.** В каких случаях первообразные корни из 1 степеней $n \leq 30$ можно найти решая кубические уравнения?

**Ответ.** Кроме степеней, приведенных в ответе к предыдущей задаче, это степени 7, 9, 13, 18, 19, 27. Однако, мы воздержимся от того, чтобы приводить явные формулы, ввиду их громоздкости.

If you don't care where you are, then you ain't lost.

## § 5. Простые числа

Die Mathematik ist die Königin der Wissenschaften, und die Zahlentheorie ist die Königin der Mathematik[107].

Карл Фридрих Гаусс

Mathematician: 3 is a prime, 5 is a prime, 7 is a prime,
Physicist: 3 is a prime, 5 is a prime, 7 is a prime, 9 is an experimental error, 11 is a prime,...
Engineer: 3 is a prime, 5 is a prime, 7 is a prime, 9 is a prime, 11 is a prime,...
Programmer: 3 is a prime, 5 is a prime, 7 is a prime, 7 is a prime, 7 is a prime,...
Salesperson: 3 is a prime, 5 is a prime, 7 is a prime, 9 – we'll do for you the best we can,...
Computer Software Salesperson: 3 is a prime, 5 is a prime, 7 is a prime, 9 will be prime in the next release,...
Biologist: 3 is a prime, 5 is a prime, 7 is a prime, 9 – results have not arrived yet,...
Advertiser: 3 is a prime, 5 is a prime, 7 is a prime, 11 is a prime,...
Lawyer: 3 is a prime, 5 is a prime, 7 is a prime, 9 — there is not enough evidence to prove that it is not a prime,...
Accountant: 3 is a prime, 5 is a prime, 7 is a prime, 9 is a prime, deducing 10% tax and 5% other obligations.
Statistician: Let's try several randomly chosen numbers: 17 is a prime, 23 is a prime, 11 is a prime...
Professor: 3 is a prime, 5 is a prime, 7 is a prime, and the rest are left as an exercise for the student.
Computational linguist: 3 is an odd prime, 5 is an odd prime, 7 is an odd prime, 9 is a very odd prime,...
Psychologist: 3 is a prime, 5 is a prime, 7 is a prime, 9 is a prime but tries to suppress it,...

Pure mathematics, on the other hand, seems to me a rock on which all idealism founders: 317 is a prime, not because we think so, or because our minds are shaped in one way rather than another, but *because it is so*, because mathematical reality is built that way.

Godfrey Harold Hardy[108]

Содержание этого параграфа основано на книге Рибенбойма (имеются также слегка переработанный португальский текст и более короткая по-

[107] Математика королева наук, а теория чисел королева математики.

[108] Обычно мы не приводим переводов с английского. В данном случае такой перевод необходим. "С другой стороны, чистая математика представляется мне скалой, о которую разбивается всякий идеализм: число 317 простое не потому, что нам так кажется, и не потому, что так а не иначе создан наш разум, а *потому, что это так*, потому что так устроена математическая реальность." — Г.Г.Харди, Апология математика. — RCD, Ижевск, 2000, с.1–102. Интересно, что в русском переводе смысл фразы заменен на *прямо противоположный*: "скалой, на которой *зиждется* всякий идеализм". Мало того, что переводчик не отличает `founders` от `is founded`, он к тому же не понимает смысл и пафос всего окружающего текста!

пулярная версия этой книги[109,110], а также цитированный в предисловии сокращенный русский перевод предварительного варианта книги). Многие исторические детали взяты из книг Наркевича и Серпиньского, а вычислительные аспекты всех рассматриваемых здесь проблем в общих чертах обсуждаются в книгах Ризеля и Крандалла—Померанса. Однако, конкретные детали меняются так быстро, что за ними можно следить только по интернет-сайтам.

**1. Простые числа.**[111]

Натуральное число $p \neq 1$ называется **простым**, если у него нет собственных делителей, т.е. делителей, отличных от самого $p$ и 1. В системе Mathematica имеется несколько важных и удобных функций для работы с простыми числами.

| | |
|---|---|
| `Prime[n]` | $n$-е простое число |
| `PrimePi[n]` | количество простых $\leq n$ |
| `PrimeQ[n]` | тест простоты $n$ |
| `ProvablePrimeQ[n]` | детерминистический тест простоты $n$ |
| `Primes` | домен простых чисел |

Использование этих функций ясно само по себе. Стоит, однако, подчеркнуть, что имплементация теста `PrimesQ` в системе вероятностная. *Известно*, что этот тест дает *достоверный* ответ для всех простых $\leq 10^{16}$ — и, тем самым во всех рассматриваемых в настоящей главе примеров! Для получения достоверного, а не просто в высшей степени правдоподобного ответа в общем случае необходимо использовать функцию `ProvablePrimeQ` из пакета `NumberTheory‘PrimeQ‘`.

**1.1.** Породите список первых $n$ простых.

**Ответ.** Проще всего так: `Table[Prime[i],{i,1,n}]`. Конечно, для фактического вывода такого списка нужно подставить вместо $n$ конкретное значение.

**1.2.** Определите номер простого числа $p$.

**1.3.** Породите список всех простых, не превосходящих $n$.

**Ответ.** Можно, например, так: `Table[Prime[i],{i,1,PrimePi[n]}]`.

**1.4.** Найдите наибольшее простое строго меньшее, чем $n$.

**Ответ.** Например, так: `Prime[PrimePi[n-1]]`.

**1.5.** Найдите $m$ наибольших простых строго меньших, чем $n$.

**Решение.** Можно, конечно, взять $m$ последних элементов из списка всех простых, меньших, чем $n$, но это плохая идея. Конечно, скорость работы

---

[109] P.Ribenboim, Números primos: mistérios e records. Ass. Inst. Nac. Math. Pura e Appl. Rio de Janeiro, 2001.

[110] P.Ribenboim, The little book of big primes, Springer, N.Y. et al, 1991.

[111] Enter any 11-digit prime number to continue...

следующей программы тоже уменьшается с ростом $n$, но совсем не так быстро.

```
prevprimes[n_,m_]:=
Sort[NestList[Prime[PrimePi[#-1]]&,Prime[PrimePi[n-1]],m-1]]
```

Для чисел порядка $10^9$ команда `prevprimes` для небольших значений $m$ работает сотые доли секунды, в то время как порождение всего списка простых меньших $m$ требует нескольких минут.

**1.6.** Найдите наименьшее простое строго большее, чем $n$.

**Решение.** Вот простая процедурная программа из уже упоминавшейся в данном модуле книги Стена Вагона[112], решающая эту задачу:

```
PrimeAfter[1]:=2
PrimeAfter[n_]:=Module[{p=n+1+Mod[n,2]},
                       While[Not[PrimeQ[p]],p+=2];p]
```

Поскольку сами мы редко организуем циклы, поясним, что именно здесь происходит. Так как первая строка в комментарии не нуждается, объясним вторую. Команда `Module` используется для локализации переменной $p$, которой придается первое *нечетное* значение большее $n$. После этого проверяется, будет ли это $p$ простым и, если нет, его значение инкрементируется на два (`p+=2` это просто программистское сокращение для `p=p+2`), пока получившееся $p$ не станет простым. После чего возвращается получившееся значение $p$.

**1.7.** Найдите первые четыре простых числа вида $1\ldots1$.

**Предостережение.** У начинающего возникнет искушение задать число, состоящее из $n$ единиц, посредством

```
ones[n_]:=Sum[10^i,{i,0,n-1}]
```

Следует, однако, иметь в виду, что это один из многочисленных случаев, когда гораздо естественнее воспользоваться внутренними командами работы со списками, в данном случае списком цифр

```
ones[n_]:=FromDigits[Table[1,{n}]
```

Конечно, для совсем небольших чисел, имеющих лишь несколько сотен разрядов, разница во времени работы этих конструкций малозаметна. Однако, с ростом $n$ время выполнения первой конструкции растет как $n^3$, а второй как $2n$. Поэтому для чисел с несколькими тысячами разрядов конструкции, использующие списки, работают в сотни раз быстрее, чем явные арифметические вычисления. С другой стороны, для чисел с сотнями миллионов разрядов реальным ограничением второй констукции становится используемая память. Поэтому если Вы хотите работать с *по-настоящему* большими числами, правильнее всего задать число, состоящее из $n$ единиц, как `(10^n-1)/9`. К сожалению, раскладывать числа такого порядка на множители мы все равно не умеем.

---

[112] S.Wagon. `Mathematica` in Action — Springer-Verlag, 1999.

**1.8.** Найдите первые 100 простых чисел, состоящих из

- единиц и троек.
- единиц и пятерок.
- единиц и семерок.
- единиц и девяток.

Какие из них встречаются реже всего и почему?

Вот еще одна классическая задача. Простые числа $p$ и $p+2$, разность которых равна 2, называются **близнецами**.

**1.9.** Найдите первые 1000 пар близнецов. Как меняется отношение количества близнецов $\leq n$ к общему количеству простых чисел $\leq n$?

**Ответ.** Например, при помощи следующей конструкции:

```
Select[Table[{Prime[i],Prime[i]+2},{i,1,n}],PrimeQ[#[[2]]]&]
```

Варварское, но верное решение состоит теперь в том, чтобы взять такое $n$, для которого этот список заведомо содержит больше 100 пар, например, $n = 1000$, и выбрать из него первые 100 посредством `Take[list,100]`.

За исключением случая 3, 5, 7 числа вида $p, p+2, p+4$ не могут быть все три простыми (кстати, почему?) — но вот числа $p, p+2, p+6$ и $p, p+4, p+6$ могут. Еще интереснее искать **четверки** простых чисел $p, p+2, p+6, p+8$. Первая такая четверка — это 5, 7, 11, 13, следующая — 11, 13, 17, 19.

**1.10.** Найдите первые 100 четверок. Есть ли среди них еще хотя бы одна такая, для которой последние цифры не есть 1, 3, 7, 9?

**Ответ.** Перечислим лишь те четверки, которые меньше 10000. Кроме двух уже упомянутых, их еще всего 10 штук:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 101 | 103 | 107 | 109 | 191 | 193 | 197 | 199 |
| 821 | 823 | 827 | 829 | 1481 | 1483 | 1487 | 1489 |
| 1871 | 1873 | 1877 | 1879 | 2081 | 2083 | 2087 | 2089 |
| 3251 | 3253 | 3257 | 3259 | 3461 | 3463 | 3467 | 3469 |
| 5651 | 5653 | 5657 | 5659 | 9431 | 9433 | 9437 | 9439 |

**1.11.** Убедитесь, что четверки встречаются очень часто.

**Указание.** Искать большие четверки лучше, конечно, не посредством `Select`, а организуя цикл начиная с некоторого места.

**Ответ.** Набрав что-нибудь в духе

```
Timing[Block[{i=10000000000000000001},While[
      Not[PrimeQ[i]&&PrimeQ[i+2]&&PrimeQ[i+6]&&PrimeQ[i+8]],
            i=i+10];{i,i+2,i+6,i+8}]]
```

мы в течение секунд увидим что-нибудь в духе

10000000000000000081721 10000000000000000081723
10000000000000000081727 10000000000000000081729

Для чисел небольшого порядка обычно это требует не больше миллиона итераций. Это значит, что четверок очень много.

**1.12.** Найдите первые 100 простых вида $n^2 + 1$.

**1.13.** Найдите первые 100 простых вида $n^2 + n + 1$.

**2. Теорема Эвклида.**[113]

В десятой книге "Элементов" Эвклида содержится следующее утверждение, известное как **теорема Эвклида**: множество рациональных простых чисел бесконечно. Основная идея доказательства теоремы Эвклида состоит в том, чтобы построить бесконечную последовательность попарно взаимно простых чисел $\neq 1$.

Особенно выпукло использованный для этого прием можно выразить следующим образом. Пусть $n_1, \ldots, n_s$ — любые натуральные числа. Тогда число $n_1 \ldots n_s + 1$ взаимно просто с каждым из $n_1, \ldots, n_s$. Определим теперь рекуррентно последовательность чисел $E_i$, $i \in \mathbb{N}$, полагая $E_1 = 2$ и $E_i = E_1 \ldots E_{i-1} + 1$ для всех $i \geq 2$. В силу только что сказанного все $E_i$ представляют собой попарно взаимно простые числа $> 1$. Пусть теперь $q_i$ — наименьший простой делитель числа $E_i$. Тогда все $q_i$, $i \in \mathbb{N}$, попарно различные простые числа.

Кнут предложил называть $E_i$ **числами Эвклида**. Таким образом, они имеют бесконечно много различных простых делителей. С другой стороны, Одони[114] доказал, что имеется бесконечно много простых чисел, которые *не* делят ни одно из чисел Эвклида.

Ясно, что

$$E_2 = 2 + 1 = 3, \qquad E_3 = 2 \cdot 3 + 1 = 7, \qquad E_4 = 2 \cdot 3 \cdot 7 + 1 = 43.$$

Обратите внимание, что все это простые числа!

**2.1.** Напишите рекуррентную программу для вычисления чисел Эвклида. Верно ли что все они простые?

**Ответ.** Нет, уже следующее число Эвклида $E_5 = 2 \cdot 3 \cdot 7 \cdot 43 + 1 = 1807$ не простое: $1807 = 13 \cdot 139$. Число $E_6 = 3263443$ снова простое, но, насколько нам известно, среди $E_n$, $n \geq 7$, не удалось найти ни одного простого числа. Большинство специалистов верят, что все они составные.

**2.2.** Найдите разложения нескольких следующих чисел Эвклида $E_n$ на простые. В тех случаях, когда это невозможно, найдите какие-то их простые делители.

**Указание.** Попробуйте использовать функцию `FactorInteger`.

**Ответ.** Так как при переходе от числа Эвклида $E_n$ к числу Эвклида $E_{n+1}$ количество цифр почти удваивается, то в $E_{30}$ уже больше 109 миллионов

---

[113] I was interviewed in the Israeli Radio for five minutes and I said that more than 2000 years ago, Euclid proved that there are infinitely many primes. Immediately the host interuppted me and asked: "Are there still infinitely many primes?" ©Noga Alon

[114] R.W.K.Odoni, On the prime divisors of the sequence $w_{n+1} = 1 + w_1 w_2 \ldots w_n$. — J. London Math. Soc. 1985, vol.32, p.1–11.

цифр. Поскольку числа Эвклида — и их наименьшие простые делители! — настолько быстро растут, уже при $n \geq 11$ поиск их явных разложений на простые множители на бытовом компьютере при помощи элементарных методов чрезвычайно затруднителен или прямо невозможен. Вот разложения нескольких первых из них:

$$E_7 = 10650056950807 = 547 \cdot 607 \cdot 1033 \cdot 31051,$$
$$E_8 = 113423713055421844361000443 = 29881 \cdot 67003 \cdot 9119521 \cdot 6212157481,$$
$$E_9 = 128649386832786717405371459983609615466532594851958 07 =$$
$$5295435634831 \cdot 31401519357481261 \cdot 77366930214021991992277,$$

Уже число $E_{10}$ содержит 105 знаков, поэтому ограничимся указанием тех простых делителей нескольких следующих чисел, которые ищутся относительно быстро:

| | | | |
|---|---|---|---|
| $E_{10}$ | 181 | 1987 | 112374829138729 |
| $E_{11}$ | 2287 | | |
| $E_{12}$ | 73 | | |
| $E_{13}$ | 2589377038614498251653 | | |
| $E_{14}$ | 52387 | 5020387 | 5783021473 |
| $E_{15}$ | 13999 | 74203 | 9638659 |

Обратите внимание, что двадцатидвухразрядное число, указанное в качестве простого множителя числа Эвклида $E_{13}$ действительно является простым!

Однако сам Эвклид использовал для доказательства бесконечности множества простых не числа Эвклида, а примориалы. Пусть $q$ простое число. Произведение $q\# = \prod p$ всех простых $p \leq q$ не превосходящих $q$ называется **примориалом** числа $q$. Обозначение $q\#$ было предложено в 1987 году Дабнером[115] и в настоящее время стало общепринятым. Так как $q\#$ делится на все простые $\leq q$, то ни $q\# + 1$ ни $q\# - 1$ не делятся ни на одно из них и, значит, содержат новые простые делители. Довольно часто эти числа содержат очень большие простые множители, а в некоторых случаях сами являются большими простыми (`primorial primes`). В следующих задачах простой множитель $q$ числа $n$ называется большим, если $q > n/q$, и, кроме того, в случае, когда $n/q = p$ само является простым, $q > p^2$.

**2.3.** Сколько среди чисел вида $n\# + 1$, $1 \leq n \leq 50$, простых?

**Ответ.** Никаких других простых, кроме очевидных $2\# + 1 = 3$, $3\# + 1 = 7$, $5\# + 1 = 31$, $7\# + 1 = 211$ и $11\# + 1 = 2311$ не просматривается.

**2.4.** Вычислите разложение на множители чисел вида $n\# + 1$, $n \leq 50$. Какое наибольшее количество различных простых множителей при этом встречается? В каких случаях встречаются большие простые множители?

**2.5.** Сколько среди чисел вида $n\# - 1$, $1 \leq n \leq 50$, простых?

[115] H.Dubner, Factorial and primorial primes. — J. Recr. Math., 1987, vol.19, p.197–203.

**Ответ.** А вот здесь, кроме очевидных простых $3\# - 1 = 5$, $5\# - 1 = 29$, $11\# - 1 = 2309$ и $13\# - 1 = 30029$ есть еще одно, уже совсем не очевидное:

$$89\# - 1 = 23768741896345550770650537601358309.$$

**2.6.** Вычислите разложение на множители чисел вида $n\# - 1$, $1 \leq n \leq 50$. Какое наибольшее количество различных простых множителей при этом встречается? В каких случаях встречаются большие простые множители?

Вместо примориалов для доказательства теоремы Эвклида можно было бы использовать и факториалы. Снова $n!$ делится на все простые $\leq n$ и, значит, $n! + 1$ и $n! - 1$ содержат новые простые делители. Опять же, очень часто эти числа содержат громадные простые множители, а в некоторых случаях и сами являются простыми (**factorial primes**).

**2.7.** Сколько среди чисел вида $n! + 1$, $2 \leq n \leq 50$ простых? Квадратов простых?

**Ответ.** Кроме очевидных случаев $1! + 1 = 2$, $2! + 1 = 3$, $3! + 1 = 7$ простыми оказываются только

$$11! + 1 = 39916801,$$
$$27! + 1 = 10888869450418352160768000001,$$
$$37! + 1 = 13763753091226345046315979581580902400000001,$$
$$41! + 1 = 33452526613163807108170062053440751665152000000001.$$

Единственные квадраты простых встречаются в самом начаале: $4! + 1 = 25 = 5^2$, $5! + 1 = 121 = 11^2$ и $7! + 1 = 5041 = 71^2$.

**2.8.** Вычислите разложение на множители чисел вида $n! + 1$, $e \leq n \leq 50$. Какое наибольшее количество различных простых множителей при этом встречается? В каких случаях встречаются большие простые множители?

**2.9.** Сколько среди чисел вида $n! - 1$, $2 \leq n \leq 50$, простых?

**Ответ.** Кроме очевидных случаев $3! - 1 = 5$, $4! - 1 = 23$, $6! - 1 = 719$ и $7! - 1 = 5039$ простыми оказываются только

$$12! - 1 = 479001599,$$
$$30! - 1 = 265252859812191058636308479999999,$$
$$32! - 1 = 263130836933693530167218012159999999,$$
$$33! - 1 = 8683317618811886495518194401279999999,$$
$$38! - 1 = 52302261746660111176000722410007429119999999 9.$$

**2.10.** Вычислите разложение на множители чисел вида $n! + 1$, $2 \leq n \leq 50$. Какое наибольшее количество различных простых множителей при этом встречается? В каких случаях встречаются большие простые множители?

## 3. Теорема Банга-Жигмонди.[116]

[116] I can do without essentials but I must have my luxuries.©Oscar Wilde

В действительности, чтобы найти бесконечное количество простых чисел, не обязательно даже, чтобы члены последовательности были попарно взаимно просты, достаточно, чтобы (начиная с некоторого места) каждый следующий член имел *примитивный* простой делитель, не являющийся делителем ни одного из предыдущих членов этой последовательности.

Оказывается, каждая пара *взаимно простых* натуральных чисел $x > y$ порождает такую последовательность, $n$-м членом которой является $x^n - y^n$. Банг (1886 год, в частном случае $y = 1$) и Жигмонди[117] (1892 год, в общем случае) обнаружили следующий исключительно важный факт, очень часто используемый в теории чисел, комбинаторике и теории групп: $x^n - y^n$ *почти всегда* имеет **примитивный** простой делитель, который не делит никакую из разностей $x^m - y^m$ при $m < n$. В дальнейшем теорема Банга—Жигмонди многократно переоткрывалась и известна под разными названиями. В старых учебниках теории чисел она чаще всего называется теоремой Биркгофа—Вандивера[118].

Одним из забавных следствий этой теоремы является утверждение, что для любого $s$ существует простое число $p$ такое, что десятичное разложение $1/p$ имеет период $s$.

Очевидные исключения получаются здесь при $n = 1$ и $x = y + 1$ и при $n = 2$, $x + y = 2^k$ для некоторого $k$. Кроме того, имеется еще одно не совсем очевидное исключение.

**3.1.** Имеется еще *ровно одна* тройка $(x, y, n)$, для которой $x^n - y^n$ не имеет примитивных простых делителей. Найдите ее.

**Ответ.** Организовав полный перебор легко обнаружить, что при $x = 2$, $y = 1$, и $n = 6$ число $2^6 - 1 = 63 = 3^2 \cdot 7$ не имеет примитивных простых делителей: $2^2 - 1 = 3$ и $2^3 - 1 = 7$.

**3.2.** Укажите для каждого $n \le 150$ наименьший примитивный простой делитель $2^n - 1$. Что можно сказать об их величине?

**Решение.** Обозначим наименьший простой делитель $2^n - 1$ через $q_n$. Множество *всех* новых простых делителей $2^n - 1$ можно найти, например, так:

```
new[n_]:=Complement[First[Transpose[FactorInteger[2^n-1]]],
          Apply[Union,Table[
               First[Transpose[FactorInteger[2^i-1]]],
                                        {i,2,n-1}]]]
```

Теперь $q_n$ это просто первый элемент `new[n]`. Начало ответа легко посчитать в уме: $q_2 = 3$, $q_3 = 7$, $q_4 = 5$, $q_5 = 31$, $q_7 = 127$, $q_8 = 17$, $q_9 = 73$,

---

[117] K.Zsigmondy, Zur Theorie der Potenzreste. — Monatshefte Math. Phys., 1892, Bd.3, S.265–284.

[118] G.D.Birkhoff, H.S.Vandiver, On the integral divisors of $a^n - b^n$. — Ann. Math., 1904, vol.5, p.173–180.

$q_{10} = 11$. Вот еще несколько значений:

| | | | |
|---|---|---|---|
| $q_{11} = 23$ | $q_{12} = 13$ | $q_{13} = 8191$ | $q_{14} = 43$ |
| $q_{15} = 151$ | $q_{16} = 257$ | $q_{17} = 131071$ | $q_{18} = 19$ |
| $q_{19} = 524287$ | $q_{20} = 41$ | $q_{21} = 337$ | $q_{22} = 683$ |
| $q_{23} = 47$ | $q_{24} = 241$ | $q_{25} = 601$ | $q_{26} = 2731$ |
| $q_{27} = 262657$ | $q_{28} = 29$ | $q_{29} = 233$ | $q_{30} = 331$ |
| $q_{31} = 2147483647$ | $q_{32} = 65537$ | $q_{33} = 599479$ | $q_{34} = 43691$ |
| $q_{35} = 71$ | $q_{36} = 37$ | $q_{37} = 223$ | $q_{38} = 174763$ |
| $q_{39} = 79$ | $q_{40} = 61681$ | $q_{41} = 13367$ | $q_{42} = 5419$ |
| $q_{43} = 431$ | $q_{44} = 397$ | $q_{45} = 631$ | $q_{46} = 2796203$ |
| $q_{47} = 2351$ | $q_{48} = 97$ | $q_{49} = 4432676798593$ | $q_{50} = 251$ |

**3.3.** Укажите для каждого $n \le 120$ наименьший примитивный простой делитель $3^n - 1$. А теперь выберете те $n$, для которых $(3^n - 1)/2$ простое.

**3.4.** Укажите для каждого $n \le 110$ наименьший примитивный простой делитель $3^n - 2^n$. А теперь попробуйте найти его еще для 10 значений $n$. Как Вы думаете, с чем связан столь резкий рост времени, необходимого для факторизации?

**Ответ.** Около 5/6 этого времени идет на факторизацию одного числа, а именно,

$$3^{118} - 2^{118} = 5 \cdot 19471 \cdot 145142890373531870546641 \cdot 141303860911622737524613875 79$$

Множители с 24 и 29 цифрами достаточно близки для того, чтобы система могла их найти с помощью квадратичного решета, но недостаточно близки для того, чтобы она могла это сделать за секунду.

**3.5.** Убедитесь, что если $x$ и $y$ взаимно просты, то каждый примитивный простой делитель $p$ числа $x^n - y^n$ удовлетворяет сравнению $p \equiv 1 \pmod{n}$.

**3.6.** Что будет, если заменить последовательность $x^n - y^n$ на последовательность $x^n + y^n$? Всегда ли верно, что $x^n + y^n$ имеет примитивный простой делитель, не делящий никакую из сумм $x^m + y^m$ при $m \le n$, или тут тоже есть исключения?

Вот еще одна забавная иллюстрация теоремы Банга—Жигмонди.

**3.7.** Исследуйте факторизацию чисел

$$(p_1 \dots p_n + 1)^{2^m} - 1,$$

где $p_1, \dots, p_n$ суть первые $n$ нечетных простых и убедитесь, что это произведение имеет не менее $n + m$ различных простых делителей.

**4. Простые Мерсенна.**[119]

В этом и следующем пунктах мы обсудим два важнейших классических класса простых, возникающие во многих вопросах теории чисел, алгебры и комбинаторики.

Если $m$ — собственный делитель $n$, то $x^n-1$ делится на $x^m-1$. Поэтому если $M_n=2^n-1$ простое, то $n$ простое. Большинство ранних авторов были уверены, что верно и обратное, т.е. если $p$ простое, то $M_p$ тоже простое. Это заблуждение было развеяно в 1536 году Худальрикусом Региусом, который заметил, что $M_{11}-1=23\cdot 89$. В 1588 году Пьетро Кательди проверил, что $M_{17}$ и $M_{19}$ простые, при этом он заявил, что $M_{23}$, $M_{29}$, $M_{31}$ и $M_{37}$ тоже простые. В 1640 году Пьер Ферма проверил, что в действительности $M_{23}$ и $M_{37}$ составные. Позже Эйлер заметил, что $M_{29}$ составное, а в 1772 году он показал, что $M_{31}$ простое.

В связи с проблемой четных совершенных чисел Марин Мерсенн в 1644 году утверждал, что числа

$$M_p,\ \ p=2,\ \ 3,\ \ 5,\ \ 7,\ \ 13,\ \ 17,\ \ 19,\ \ 31,\ \ 67,\ \ 127,\ \ 257$$

просты, а все остальные числа $M_p$ для $p\le 257$ составные. Как позже выяснилось, этот список содержал ошибки. Числа вида $M_p=2^p-1$, где $p$ простое, называются **числами Мерсенна**. Почти все самые большие известные простые числа являются числами Мерсенна.

**4.1.** Найдите первые 15 простых чисел Мерсенна и исправьте все ошибки в его списке.

**Ответ.** Можно, например, так:

```
Select[Table[2^Prime[n]-1,{n,1,300}],PrimeQ]
```

Топорно, но для таких маленьких чисел это не имеет никакого значения. Напомним, что функция `Select[list,crit]` осуществляет выбор элементов из списка `list`, в данном случае из списка первых 300 чисел вида $2^p-1$, где $p$ простое, удовлетворяющих критерию `crit`, в данном случае критерию `PrimeQ`, осуществляющему проверку $2^p-1$ на простоту. При этом получится ровно 15 простых, а именно,

[119]The trouble with integers is that we have examined only the very small ones. Maybe all the exciting stuff happens at really big numbers, ones we can't even begin to think about in any very definite way. Our brains have evolved to get us out of the rain, find where the berries are, and keep us from getting killed. Our brains did not evolve to help us grasp really large numbers or to look at things in a hundred thousand dimensions.©Ronald L. Graham

$M_2 = 3$
$M_3 = 7$
$M_5 = 31$
$M_7 = 127$
$M_{13} = 8191$
$M_{17} = 131071$
$M_{19} = 524287$
$M_{31} = 2147483647$
$M_{61} = 2305843009213693951$
$M_{89} = 618970019642690137449562111$
$M_{107} = 162259276829213363391578010288127$
$M_{127} = 170141183460469231731687303715884105727$

и $M_{521}$, $M_{607}$, $M_{1279}$, которые слищком велики, чтобы воспроизводить их здесь.

Заметим, что на протяжении 75 лет $M_{127}$ оставалось самым большим известным простым числом. А именно, используя сформулированный чуть ниже критерий Люка—Лемера в 1876 году Люка доказал, что $M_{127}$ простое. Только в 1951 году удалось найти большие простые числа. Некоторые считают, впрочем, что первое безукоризненное доказательство простоты $M_{127}$ было дано только в 1894 году Фокембергом, но даже и в этом случае рекорд простоял 57 лет! Еще дольше, 84 года, продержался рекорд простого числа, *не* являющегося числом Мерсенна, установленный Ландри в 1867 году, а именно, $M_{59}/179951$.

Вот все остальные индексы $p$, для которых сегодня (весна 2006 года) известно, что число Мерсенна $M_p$ является простым:

2203, 2281, 3217, 4253, 4423, 9689, 9941, 11213, 19937, 21701,
23209, 44497, 86243, 110503, 132049, 216091, 756839, 859433,
1257787, 1398269, 2976221, 3021377, 6972593, 13466917,
20996011, 24036583, 25964951, 30402457

Последние из этих чисел $M_p$ настолько велики, что для десятичной записи каждого из них нужно *несколько* книг объемом 1000 страниц. Например, число $M_{30402257}$ содержит 9152052 десятичные цифры.

**4.2.** Породите первые 15 пар вида $(p, M_p)$, где $M_p$ простое Мерсенна.

**Ответ.** Можно, например, так

```
Select[Table[{Prime[n],2^Prime[n]-1},{n,1,300}],
                                        PrimeQ[#[[2]]]&]
```

Обратите внимание на использование критерия в формате анонимной функции! Дело в том, что в данном случае производится не проверка простоты элемента списка, а проверка простоты второй части этого элемента.

Не забывайте, что вызов анонимной функции должен заканчиваться амперсандом.

Проверять простоту числа $M_p$ **значительно** проще, чем простоту других чисел того же порядка. Это связано с тем, что для них имеется следующий критерий простоты, открытый в 1876 году Люком (alias Лукас, Lucas) и упрощенный в 1930 году Лемером. Чтобы сформулировать этот критерий, определим прежде всего **числа Люка** $L_n$. Положим $L_1 = 4$ и зададим следующие числа рекуррентно посредством $L_{n+1} = L_n^2 - 2$.

**4.3.** Напишите программу для вычисления чисел Люка.

**Ответ.** Поскольку рекуррентная программа очевидна, ограничимся перечислением нескольких первых $L_n$:

$$L_2 = 14, \quad L_3 = 194, \quad L_4 = 37634, \quad L_5 = 1416317954,$$
$$L_6 = 2005956546822746114,$$
$$L_7 = 4023861667741036022825635656102100994.$$

Числа Люка растут довольно быстро растут, уже у $L_{100}$ больше, чем $10^{27}$ цифр.

Так вот, **критерий Люка—Лемера** утверждает, что для того, чтобы выяснить, является ли число Мерсенна $M_p$ простым, необходимо выполнить всего одно деление, а именно, $M_p$ в том и только том случае простое, когда оно делит $L_{p-1}$.

**4.4.** Напишите тест простоты чисео Мерсенна, основанный на критерии Люка—Лемера и сравните скорость его работы с `PrimeQ`.

Обратимся теперь к разложению на простые множители тех чисел Мерсенна, которые не являются простыми. Заметим, что поиск простых делителей резко упрощается следующим **критерием Ферма—Эйлера**. Пусть $p$ и $q$ — нечетные простые. Тогда если $p|M_q$, то

$$p \equiv 1 \pmod{q}, \qquad p \equiv \pm 1 \pmod{8}.$$

**4.5.** Разложите на множители все остальные числа Мерсенна до $M_{127}$

**Ответ.** Поскольку все эти числа имеют небольшие делители, можно обойтись функцией `FactorInteger`.

$M_{11} = 2047 = 23 \cdot 89$,
$M_{23} = 8388607 = 47 \cdot 178481$,
$M_{29} = 536870911 = 233 \cdot 1103 \cdot 2089$,
$M_{37} = 137438953471 = 223 \cdot 616318177$,
$M_{41} = 2199023255551 = 13367 \cdot 164511353$,
$M_{43} = 8796093022207 = 431 \cdot 9719 \cdot 2099863$,
$M_{47} = 140737488355327 = 2351 \cdot 4513 \cdot 13264529$,
$M_{53} = 9007199254740991 = 6361 \cdot 69431 \cdot 20394401$,
$M_{59} = 576460752303423487 = 179951 \cdot 3203431780337$,
$M_{67} = 147573952589676412927 = 193707721 \cdot 761838257287$,
$M_{71} = 228479 \cdot 48544121 \cdot 212885833$,
$M_{73} = 439 \cdot 2298041 \cdot 9361973132609$,
$M_{79} = 2687 \cdot 202029703 \cdot 1113491139767$,
$M_{83} = 167 \cdot 57912614113275649087721$,
$M_{97} = 11447 \cdot 13842607235828485645766393$,
$M_{101} = 7432339208719 \cdot 341117531003194129$,
$M_{103} = 2550183799 \cdot 3976656429941438590393$,
$M_{109} = 745988807 \cdot 870035986098720987332873$,
$M_{113} = 3391 \cdot 23279 \cdot 65993 \cdot 1868569 \cdot 1066818132868207$,
$M_{131} = 263 \cdot 1035079443105516238671861923746823456 9$,
$M_{137} = 32032215596496435569 \cdot 5439042183600204290159$,
$M_{139} = 5625767248687 \cdot 1238761322052083357622784236 01$,
$M_{149} = 86656268566282183151 \cdot 8235109336690846723986161$,
$M_{151} = 18121 \cdot 55871 \cdot 165799 \cdot 2332951 \cdot 7289088383388253664437433$,
$M_{157} = 852133201 \cdot 60726444167 \cdot 1654058017289 \cdot 2134387368610417$,
$M_{163} = 150287 \cdot 704161 \cdot 110211473 \cdot 27669118297 \cdot 36230454570129675721$,
$M_{167} = 2349023 \cdot 79638304766856507377778616296087448490695649$,
$M_{173} = 730753 \cdot 1505447 \cdot 70084436712553223 \cdot 155285743288572277679887$,

**4.6.** А теперь напишите программу поиска делителей $M_p$, использующую критерий Ферма—Эйлера, которая работает быстрее, чем `FactorInteger`.

Бросается в глаза наличие у некоторых $M_p$ совсем маленьких простых делителей, скажем $23|M_{11}$, $47|M_{23}$ и $167|M_{83}$. Оказывается, это не случайность. А именно, **критерий Эйлера—Лагранжа** утверждает, что если $p \equiv 3 \pmod 4$, то $q = 2p + 1$ в том и только том случае является простым, когда $q|M_p$.

**4.7.** Найдите все $p < 1000$ такие, что $q = 2p + 1$ делит $M_p$.

## 5. Простые Ферма.[120]

Начнем со следующего незамысловатого наблюдения

**5.1.** Проверьте (или докажите!), что если $2^m + 1$, где $m \in \mathbb{N}$, простое, то $m = 2^n$, $n \in \mathbb{N}_0$.

Число вида $F_n = 2^{2^n} + 1$, где $n \in \mathbb{N}_0$, называется **числом Ферма**. Числа Ферма возникают в самых различных вопросах теории чисел, комбинаторики, алгебры и геометрии. Интересно заметить, что сам Ферма достаточно определенно утверждал, что **все** числа Ферма $F_n$ простые, но смог проверить лишь, что

$$F_0 = 3, \qquad F_1 = 5, \qquad F_2 = 17, \qquad F_3 = 257, \qquad F_4 = 65537$$

просты.

**5.2.** Подтвердите или опровергните утверждение Ферма.

**Ответ.** Приведенные выше пять чисел являются *единственными* известными сегодня простыми числами Ферма! В 1732 году Эйлер нашел разложение на множители следующего числа Ферма

$$F_5 = 4294967297 = 641 \cdot 6700417 = (2^7 5 + 1)(2^7 52347 + 1).$$

Число $F_6$ тоже легко раскладывается на множители:

$$F_6 = 18446744073709551617 = 274177 \cdot 67280421310721 = (2^8 1071 + 1)(2^8 262814145745 + 1).$$

В это большинстве классических книг по теории чисел утверждается, что это разложение было найдено в 1880 году Ландри и Ле Лассером. Однако в 1964 году К.Бирманн обнаружил, что что Томас Клаузен привел эту факторизацию в письме к Гауссу, датированном 1 января 1855 года, и что он знал, что оба множителя простые!

А вот разложить на множители число $F_7$ в докомпьютерную эпоху не было никакой возможности. В действительности, следующее разложение было найдено лишь в 1970 году[121]:

$$F_7 = 59649589127497217 \cdot 5704689200685129054721 = (2^9 116503103764643 + 1)(2^9 11141971095088142685 + 1).$$

Число $F_8$, было факторизовано лишь в 1980 году[122]:

$$F_8 = 1238926361552897 \cdot 93461639715357977769163558199606896584051237541638188580280321 = (2^{11} 604944512477 + 1)$$
$$(2^{11} 4563556626726463758259939365215180497268126833087802176771 5 + 1)$$

---

[120] Мы всегда готовы говорить правду. Но как мы ее узнаем? ©Леонид Шебаршин, *Заметки бывшего начальника разведки*

[121] M.A.Morrison, J.Brillhart, The factorisation of $F_7$. — Bull. Amer. Math. Soc., 1971, vol.77, p.264.

[122] R.P.Brent, J.M.Pollard, The factorisation of the eighth Fermat number. — Math. Comput., 1981, vol.36, p.627–630.

Приятно, что сегодня это разложение за секунды ищется на бытовом компьютере при помощи функции `FactorInteger`.

Что касается $F_9$, то в 1903 году Вестерн обнаружил, что оно делится на $2^{16}37+1 = 2424833$. Несмотря на это полная факторизация $F_9$ на множители была получена лишь в 1990 году[123] При этом оказалось, что два других делителя числа $F_9$ содержат 46 и 96 цифр, соответственно.

*Единственными* другими числами Ферма, которые сегодня *полностью* разложены на простые множители, являются $F_{10}$ и $F_{11}$, смотри[124]. Для всех остальных чисел Ферма известны лишь *какие-то* простые делители, но не полная факторизация. Вот начало факторизации $F_{10}$:

$$F_{10} = (2^{12}11131 + 1)(2^{14}395937 + 1)\dots$$

два пропущенных делителя содержат 40 и 252 цифр, соответственно. Вот начало факторизации $F_{11}$:

$$F_{11} = (2^{13}39 + 1)(2^{13}119 + 1)(2^{14}10253207784531279 + 1)$$
$$(2^{13}434673084282938711 + 1)\dots$$

и еще один делитель, содержащий 564 цифр.

Вычислительная сложность этой задачи растет *невероятно* быстро с ростом $n$. Числа Ферма $F_{12}$, $F_{13}$, $F_{14}$ и $F_{15}$ имеют 1234, 2467, 4933, 9865 разрядов, соответственно. Даже установление простоты чисел такого порядка на бытовых компьютерах может оказаться проблематичным, а искать разложение таких чисел на множители сегодня мы просто не умеем. Вот начало факторизации $F_{12}$:

$$F_{12} = (2^{14}7 + 1)(2^{16}397 + 1)(2^{16}973 + 1)(2^{14}11613415 + 1)$$
$$(2^{14}76668221077 + 1)\dots$$

Вот начало факторизации $F_{13}$:

$$F_{13} = (2^{16}41365885 + 1)(2^{17}20323554055421 + 1)(2^{19}6872386635861 + 1)$$
$$(2^{19}609485665932753836099 + 1)\dots .$$

Однако, оказывается, что узнать, является ли число Ферма $F_n$ простым, совсем просто. Например, сегодня мы не знаем никаких простых делителей чисел Ферма $F_{14}$ $F_{20}$, $F_{22}$, $F_{24}$ и многих других. В то же время известно, что эти числа не являются простыми. Это устанавливается при помощи следующего легко проверяемого теста, известного как **критерий Пепина**.

---

[123] A.K.Lenstra, H.W.Lenstra, M.S.Manasse, J.M.Pollard, The factorisation of the ninth Fermat number. — Math. Comput., 1993, vol.61, p.319–149.

[124] R.P.Brent, Factorisation of the tenth Fermat number. — Math. Comput., 1999, vol.68, p.429–451.

Для того, чтобы число Ферма $F_n$, $n > 1$, было простым, необходимо и достаточно, чтобы

$$3^{(F_n-1)/2} \equiv -1 \pmod{F_n}.$$

**5.3.** Проверьте, что числа $F_n$, $n = 10, \ldots, 15$, не являются простыми.

**Указание.** Непосредственно возвести 3 в степень такого порядка нет шансов, поэтому используйте функцию `PowerMod`. Посмотрите, какой остаток она возвращает и аккуратно сформулируйте условие!

Обратимся теперь к задаче поиска простых делителей чисел Ферма. Не следует думать, что Эйлер настолько любил считать, чтобы делить вручную 10-значное число на все простые подряд, пока *случайно* не наткнулся на делитель 641. В действительности, ему пришлось для этого выполнить всего одно или два деления, а, скорее всего, ни одного.

Реконструируем рассуждения Эйлера, чтобы читатель мог на этом *игрушечном* примере представить, при помощи каких примерно соображений ищутся простые делители у чисел, содержащих многие сотни или тысячи десятичных знаков, *если известна их структура*. Дело в том, что для того, чтобы разложить число Ферма $F_n$ на множители, достаточно проверять не все простые $p \le \sqrt{F_n}$, а лишь простые вида $p = 2^{n+1}m + 1$, где $m \in \mathbb{N}$. Это вытекает из следующего легко проверяемого соображения, известного как сравнение Эйлера: любой делитель числа $F_n$, $n \ge 3$, имеет вид $2^{n+2}m + 1$, для некоторого $m \in \mathbb{N}$.

В силу сравнения Эйлера делителями $F_5$ могут быть только простые числа вида $p = 128m + 1$. Первые два таких числа, это $p = 257$ и $p = 641$, которые получаются при $m = 2$ и $m = 5$, соответственно. Однако очевидно, что $641 = 2^4+5^4$ делит $a = 2^{32}+2^{28}5^4$. С другой стороны, применяя формулу для разности квадратов, мы видим, что $641 = 2^7 5+1$ делит $b = 2^{28}5^4-1$. Таким образом, 641 делит и разность этих чисел $F_5 = a-b$. Для того, чтобы проверить, будет ли 6700417 простым, достаточно, *в худшем случае*, произвести еще не более 4 делений, а именно, проверить, что оно не делится на простые числа вида $p = 128m + 1$, $5 \le m \le 20$, каковых, очевидно (см. таблицу простых) ровно 4, а именно, 641, 769, 1153, 1409. Однако, зная Эйлера, можно предположить, что он, скорее всего, и здесь обошелся вообще без явных вычислений. Экстраполируя этот пример, мы понимаем, что сказано в эпиграфе к этому параграфу: один изобретательный математик может с успехом заменить *сотни* вычислителей.

**5.4.** А теперь напишите программу, раскладывающую числа $F_6$, $F_7$ и $F_8$ на множители быстрее, чем это делает внутренняя команда `FactorInteger`.

Приведем список известных небольших простых делителей нескольких следующих чисел Ферма:

| | |
|---|---|
| $F_{15}$ | $2^{21}579+1, \quad 2^{17}17753925353+1,$ |
| | $2^{17}1287603889690528658928101555+1$ |
| $F_{16}$ | $2^{19}1575+1, \quad 2^{20}180227048850079840107+1$ |
| $F_{17}$ | $2^{19}59251857+1$ |
| $F_{18}$ | $2^{20}13+1$ |
| $F_{19}$ | $2^{21}33629+1, \quad 2^{21}308385+1$ |
| $F_{21}$ | $2^{23}534689+1$ |
| $F_{23}$ | $2^{25}5+1$ |
| $F_{25}$ | $2^{29}48413,29+1, \quad 2^{27}1522849979+1, \quad 2^{27}16168301139+1$ |
| $F_{26}$ | $2^{29}143165+1$ |
| $F_{27}$ | $2^{30}141015+1, \quad 2^{29}430816215+1$ |
| $F_{28}$ | $2^{36}25709319373+1$ |
| $F_{29}$ | $2^{31}1120049+1$ |
| $F_{30}$ | $2^{32}149041+1, \quad 2^{33}127589+1$ |

Заметим, что числа Ферма дают еще один подход к доказательству теоремы Эвклида бесконечности числа простых. В самом деле, из следующей задачи (взятой непосредственно из переписки Гольдбаха и Эйлера[125]) вытекает, что числа Ферма попарно взаимно просты.

**5.5.** Проверьте (или докажите!), что

$$F_0F_1F_2\dots F_n = F_{n+1}-2.$$

Таким образом, если $F_m$ делит $F_n$, при некотором $n>m$, то $F_m$ делит 2, что невозможно.

Рассматриваются различные вариации на тему чисел Ферма. Вот три наиболее известные из них.

• Числа вида $b^{2^n}+1$ называется **обобщенными числами Ферма**[126,127], обычные числа Ферма получаются здесь при $b=2$.

• Число $C_n = n\cdot 2^n+1$ называется **числом Каллена**.

• Число $W_n = n\cdot 2^n-1$ называется **числом Вудалла**.

---

[125] Доказательство, основанное на той же идее, но содержащее несколько чрезвычайно удачных ухудшений, воспроизводится в книге "Задачи и теоремы из анализа", поэтому некоторые авторы ошибочно приписывают его Пойа и Сеге.

[126] H.Dubner, W.Keller, Factors of generalized Fermat numbers. — Math. Comput., 1995, vol.64, p.397–405.

[127] A.Björn, H.Riesel, Factors of generalized Fermat numbers. — Math. Comput., 1998, vol.67, p.441–446.

**5.6.** Вычислите первые 140 чисел Каллена $C_n = n \cdot 2^n + 1$ и убедитесь, что все они, кроме $C_1 = 3$ составные. Достаточно ли этого, чтобы сформулировать гипотезу, что все числа $C_n$, $n > 1$, составные?

**Ответ.** Нет, число $C_{141}$ простое[128]. Кроме того, известны следующие простые Каллена[129,130]:

$$C_{4713},\ C_{5795},\ C_{6611},\ C_{18496},\ C_{32292},\ C_{32469},$$
$$C_{59656},\ C_{90825},\ C_{262419},\ C_{361275},\ C_{481899}.$$

В отличие от простых Каллена, среди простых Вудалла с небольшими индексами достаточно много простых.

**5.7.** Найдите первые 15 чисел Вудалла $W_n = n \cdot 2^n - 1$.

**Ответ.** Вот они: $W_2 = 7$, $W_3 = 23$, $W_6 = 383$,

$W_{30} = 32212254719$,

$W_{75} = 283341988972178712821759 9$,

$W_{81} = 1958459827775699263024005 11$,

$W_{115} = 47769131098520414182480566228824883 19$,

$W_{123} = 13079603478523572189373461473158590627 83$,

и, кроме того, $W_{249}$, $W_{362}$, $W_{384}$, $W_{462}$, $W_{512}$, $W_{751}$, $W_{822}$. Следующие простые Вудалла непомерно велики:

$$W_{5312},\ W_{7755},\ W_{9531},\ W_{12379},\ W_{15822},\ W_{18885},$$
$$W_{22971},\ W_{23005},\ W_{98726},\ W_{143018},\ W_{151023},\ W_{667071}.$$

## 6. Распределение простых.[131]

Обозначим через $\pi(n)$ количество натуральных простых не превосходящих $n \in \mathbb{N}$. Ясно, что если $n = p \in \mathbb{P}$ простое, то $\pi(p) = \pi(p-1) + 1$. Напомним, что в соответствии с общим приницпом образования имен на языке `Mathematica` функция $\pi$ называется `PrimePi`.

**6.1.** Составьте таблицу значений функций $\pi(n)$ и $n/\pi(n)$ для $n = 10^m$, $1 \le m \le 14$, и сравните вычисленные значения с $\ln(n)$.

---

[128] R.M.Robinson, A report on primes of the form $k \cdot 2^n + 1$ and on factors of Fermat numbers. — Proc. Amer. Math. Soc., 1958, vol.9, p.673–681.

[129] W.Keller, Factors of Fermat numbers and large primes of the form $k \cdot 2^n + 1$. — Math. Comput., 1983, vol.41, p.661–673.

[130] W.Keller, New Cullen primes. — Math. Comput., 1995, vol.64, p.1733–1741.

[131] Nature is a good approximation of Mathematics ©Zvi Artstein

**Ответ.** Вот как выглядят эти значения:

| $n$ | $\pi(n)$ | $n/\pi(n)$ | $\ln(n)$ |
|---|---|---|---|
| 10 | 4 | 2.5 | 2.30259 |
| 100 | 25 | 4. | 4.60517 |
| 1000 | 168 | 5.95238 | 6.90776 |
| 10000 | 1229 | 8.1367 | 9.21034 |
| 1 00000 | 9592 | 10.4254 | 11.5129 |
| 10 00000 | 78498 | 12.7392 | 13.8155 |
| 100 00000 | 6 64579 | 15.0471 | 16.1181 |
| 1000 00000 | 57 61455 | 17.3567 | 18.4207 |
| 10000 00000 | 508 47534 | 19.6666 | 20.7233 |
| 1 00000 00000 | 4550 52511 | 21.9755 | 23.0259 |
| 10 00000 00000 | 41180 54813 | 24.2833 | 25.3284 |
| 100 00000 00000 | 3 76079 12018 | 26.5901 | 27.6310 |
| 1000 00000 00000 | 34 60655 36839 | 28.8963 | 29.9336 |
| 10000 00000 00000 | 320 49417 50802 | 31.2018 | 32.2362 |

Эта таблица показывает, что маленьких простых чисел довольно много: скажем, среди первых 10000 чисел 1229 простых — больше, чем 12%. В действительности, среди первого миллиарда натуральных чисел все еще больше 5% простых!!!

Глядя — в пятнадцатилетнем возрасте! — на такую, или чуть более короткую, таблицу, Гаусс заметил, что, при переходе от каждой степени 10 к следующей, отношение $n/\pi(n)$ увеличивается примерно на $\ln(10) \cong 2.30259$. Это натолкнуло его на замечательное предположение, что при $n$ стремящемся к $\infty$ функция $n \mapsto \pi(n)$ растет примерно как $n \mapsto n/\ln(n)$. Утверждение об асимптотической эквивалентности $\pi(n)$ и $n/\ln(n)$ называется **асимптотическим законом** распределения простых чисел или, иногда, просто **теоремой о простых числах**.

Первое выдающееся продвижение в направлении доказательства асимптотического закона получил в 1849 году П.Л.Чебышев. Однако, полностью асимпотический закон распределения простых был доказан лишь в 1896 году независимо Адамаром и де ла Валле-Пуссеном с использованием теории функций комплексной переменной. Запомнить дату 1896 чрезвычайно легко, поскольку Адамар прожил 98 лет, а де ла Валле-Пуссен — 96. Рибенбойм выражает уверенность, что это произошло именно как следствие того, что они доказали столь замечательную теорему. Элементарное доказательство асимптотического закона нашли Сельберг[132] и Эрдеш в 1949 году.

При изучении подгрупп в симметрической группе Жозеф Бертран использовал следующее утверждение, известное как **постулат Бертрана**: при $n > 7$ между $n/2$ и $n-2$ всегда содержится хотя бы одно простое

[132] A.Selberg, An elementary proof of the prime number theorem. — Ann. Math., 1951, vol.85, p.203–362.

число. Сам Бертран проверил это утверждение для всех $n < 1500000$, но его доказательством в общем случае он не владел. Первое доказательство постулата Бертрана придумал в 1852 году П. Л. Чебышев.

**6.2.** Проверьте постулат Бертрана для всех $n \leq 2000000$.

Шинцель предложил следующее усиление постулата Бертрана, известное как **гипотеза Шинцеля**: для каждого $n \geq 117$ между $n$ и $n + \sqrt{n}$ найдется хотя бы одно простое число.

**6.3.** Проверьте гипотезу Шинцеля для всех $117 \leq n \leq 100000$. Как Вы думаете, с чем связано ограничение $n \geq 117$?

Верно ли, что для любых $x, y \geq 2$ выполняется неравенство $\pi(x + y) \leq \pi(x) + \pi(y)$?

**6.4.** Проверьте выполнение этого неравенства для всех $x, y \leq 1000$.

**7. Теорема Дирихле.**[133]

В простейшем варианте знаменитая **теорема Дирихле** о простых в арифметических прогрессиях, с которой собственно и начинается современная аналитическая теория чисел, утверждает, что для любых *взаимно простых* $a$ и $d$ в арифметической прогрессии $a+nd$, $n \in \mathbb{N}$, бесконечно много простых. В частности, существует бесконечно много простых с *любыми* наперед заданными $m$ последними цифрами, если только последняя цифра не равна 0, 2, 4, 5, 6, 8.

Однако, как мы сейчас увидим, в действительности теорема Дирихле утверждает *гораздо* больше, чем просто бесконечность простых в арифметической прогрессии.

**7.1.** Найдите количество простых $\leq 10^6$, последняя цифра которых равна 1, 3, 7, 9.

**Решение.** Вычисляя

```
Map[Length[Select[Range[#,10^6,10],PrimeQ]]&,{1,3,7,9}]
```

мы видим, что количество простых с последней цифрой 1, 3, 7, 9 практически не зависит от этой цифры: Вот как распределяются по последней цифре 78498 простых чисел меньших одного миллиона:

$$19617, \qquad 19665, \qquad 19621, \qquad 19593.$$

Эта закономерность становится еще очевиднее, если продолжить вычисления. Вот как распределяются по последней цифре 664579 простых чисел меньших десяти миллионов:

$$166104, \qquad 166230, \qquad 166211, \qquad 166032,$$

[133] Какое чудо, если есть
Тот, кто затеплил в нашу честь
Ночное множество созвездий !
А если всё само собой
Устроилось, тогда, друг мой,
Еще чудесней! ©Александр Кушнер

и вот, наконец, 5761455 простых чисел меньших ста миллионов:

$$1440299, \qquad 1440474, \qquad 1440495, \qquad 1440186$$

Мы видим, что каждый раз количество простых с данной последней цифрой почти в точности равно четверти общего количества простых. Не бойтесь повторить последнее вычисление - на персональном компьютере класса `Intel Core 7` с оперативной памятью 8ГБ процесс займет около минуты.

Взглянем теперь на последние *две* цифры.

**7.2.** Породите множество чисел, которые могут быть остатками простого числа по модулю 100.

**Решение.** Как всегда, `Mathematica` дает почти бесконечное разнообразие способов решения этой задачи. Вот первые приходящие в голову:

```
Map[FromDigits[#]&,Flatten[Outer[List,Range[0,9],{1,3,7,9}],1]]
Select[Range[100],MemberQ[{1,3,7,9},Last[IntegerDigits[#]]]&]
Select[Range[100],MemberQ[{1,3,7,9},Mod[#,10]]&]
Select[Range[100],GCD[#,10]==1&]
Apply[Union,Map[Range[#,100,10]&,{1,3,7,9}]]
```

Конечно, при получении списка из 40 чисел вопрос выбора алгоритма является чисто схоластическим. Заметим, впрочем, что из приведенных выше определений последнее заведомо лучше предыдущих: оно лучше первого потому, что гораздо легче обобщается на любое количество цифр и лучше трех других потому, что не требует выбора по критерию из огромного списка. Уже при порожении списка шестизначных чисел оно эффективнее раз в 50.

**7.3.** Найдите распределение простых $\leq 10^6$ по последним двум цифрам.

**Ответ.** Вот это распределение, где строки занумерованы цифрами 1, 3, 7, 9, а столбцы отвечают десяткам:

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1964 | 1958 | 1937 | 1964 | 1955 | 1970 | 1960 | 1986 | 1942 | 1981 |
| 1969 | 1965 | 1976 | 1967 | 1959 | 1977 | 1962 | 1956 | 1969 | 1965 |
| 1932 | 1970 | 1976 | 1973 | 1956 | 1961 | 1943 | 1960 | 1984 | 1966 |
| 1957 | 1973 | 1926 | 1970 | 1960 | 1967 | 1955 | 1960 | 1958 | 1967 |

Снова в каждый класс попадает примерно одинаковое количество простых, а именно около 1/40 от общего количества.

Теперь уже очевидно, что не только в каждой арифметической прогрессии содержится бесконечное количество простых, но и то, что простые распределены равномерно между всеми прогрессиями основание которых взаимно просто с фиксированной разностью $d$. Именно это, конечно, и утверждает теорема Дирихле!

**7.4.** Проверьте утверждение теоремы Дирихле на других примерах.

**Ответ.** Вот, скажем, как распределяются 78496 простых меньших одного миллиона по двум классам по модулям 3 и 4:

- по модулю 3: 39231 из них дают остаток 1 и 39266 остаток 2;
- по модулю 4: 39175 из них дают остаток 1 и 39322 остаток 3.

Убедительно?

> All the wonders of our universe can in effect be captured by simple rules, yet there can be no way to know all the consequences of these rules, except in effect just to watch and see how they unfold.
>
> Stephen Wolfram, *A New Kind of Science*

## § 6. Модулярная арифметика

> Люди, думающие, что духовные, эзотерические, высшие движения в мире возникают ни с того, ни с сего, должны уяснить, что нет ничего более далекого от истины, чем это предположение. Любому подлинному знанию всегда предшествуют чрезвычайно сложные планирование и подготовка.
>
> Идрис Шах, *Знание как знать*

> The only thing that he did as Deputy Mayor was to reduce the Shirriffs to their proper functions and numbers.
>
> J.R.R Tolkien, *The Lord of the Rings*

В данном параграфе мы обсуждаем арифметические структуры, связанные с делением целых чисел с остатком. С математической точки зрения речь здесь идет о вычислениях в кольце классов вычетов $\mathbb{Z}/m\mathbb{Z}$. Это один из самых древних разделов математики, восходящий к *И Цзин*, египетским писцам и секте пифагорейцев. Некоторые из излагаемых ниже алгоритмов известны *по крайней мере* около 2500 лет и являются самыми старыми алгоритмами, полностью сохранившими свою актуальность. В последние десятилетия модулярная арифметика широчайшим образом используется в безошибочных вычислениях.

### 1. Делимость целых чисел.[134,135]

Говорят, что целое число $x \in \mathbb{Z}$ **делит** $y \in \mathbb{Z}$ или, что то же самое, что $y$ **делится** на $x$, если существует такое $z \in \mathbb{Z}$, что $y = xz$. Это обозначается одним из двух следующих образов:

$$x|y \text{ — } x \text{ делит } y, \qquad \text{либо} \qquad y\dot{:}x \text{ — } y \text{ делится на } x.$$

При этом $x$ называется **делителем** $y$, а $y$ — **кратным** $x$.

---

[134] Breast size multiplied by IQ always equals 69. — Cambridge Mathematical Quotes

[135] Now here's a start I made the other day: Dante wrote a poem, about a place called H—. H-dash, because I don't want any trouble with the censors. ©Henry Miller, *Black Spring*

Перечислим основные свойства делимости.

• **Рефлексивность**: $x|x$.

• **Транзитивность**: если $x|y$ и $y|z$, то $x|z$.

• Если $x|y, z$, то $x|(y+z)$.

• Если $x|y$, то $x|yz$ для любого $z$.

Эти свойства допускают следующее совместное обобщение.

• Если $x|y_1, \ldots, y_s$, то для любых $z_1, \ldots, z_s$ имеем $x|y_1z_1 + \ldots + y_sz_s$ т.е., иначе говоря, $x$ делит любую линейную комбинацию элементов $y_1, \ldots, y_s$.

Выяснять, является ли число $x$ делителем числа $y$ лучше всего при помощи определенной в следующем параграфе функции `Mod`. А именно, $x$ в том и только том случае делит $y$, когда `Mod[y,x]` равно 0. Еще один способ состоит в том, чтобы убедиться, что `Floor[y/x]` совпадает с $y/x$. Это вычисление чуть менее эффективно, чем вычисление с помощью функции `Mod` — к слову, совсем не потому, что функция `Mod` целочисленная, тем более, что аргументы `Mod` не обязаны быть целыми числами. Однако, в данном случае разница в производительности становится по настоящему заметной только для чисел с несколькими миллионами цифр. Наконец, наименее эффективный способ состоит в том, чтобы явно проверить, содержится ли $x$ в списке делителей $y$ посредством `MemberQ[Divisors[y],x]`. Так как этот способ требует полной факторизации $y$, и поиска в списке, то он применим только к числам, содержащим не более несколько десятков разрядов.

**Общим делителем** $x$ и $y$ называется такое число $d \in \mathbb{Z}$, что $d|x$, $d|y$. **Наибольшим общим делителем** этих чисел называется такой их общий делитель $d$, который делится на любой другой их общий делитель, иными словами, если для любого целого числа $z$ из того, что $z|x$ и $z|y$ следует, что $z|d$. Наибольший общий делитель элементов $x$ и $y$ обычно обозначается НОД$(x, y)$ или $\gcd(x, y)$, от английского `greatest common divisor`. Если дополнительно предполагать, что $d \geq 0$, то он определен однозначно и обладает следующими свойствами.

• **Поглощение**: $\gcd(x, y) = x \iff x|y$. В частности, $\gcd(x, x) = x$ и $\gcd(x, 0) = x$.

• **Коммутативность**: $\gcd(x, y) = \gcd(y, x)$.

• **Ассоциативность**: $\gcd(\gcd(x, y), z) = \gcd(x, \gcd(y, z))$.

• Умножение **дистрибутивно** относительно операции взятия наибольшего общего делителя: для любых $x, y, z$ выполнено равенство $\gcd(zx, zy) = z\gcd(x, y)$.

Наибольший общий делитель допускает **линейное представление**: если $d = \gcd(x, y)$, то найдутся такие $a$, $b$, что $d = ax + by$. Обратно, наибольший общий делитель чисел $x$ и $y$ может быть определен как такой их общий делитель, который допускает линейное представление.

Целые числа $x$ и $y$ называются **взаимно простыми**, если их наибольший общий делитель равен 1. Взаимно простые числа **комаксимальны**,

иными словами, для них найдутся такие $a$, $b$, что $ax+by=1$. Следуя Кнуту мы обозначаем взаимную простоту чисел $x$ и $y$ посредством $x \perp y$.

Двойственным образом к понятию наибольшего общего делителя, иными словами, заменой всюду `делит` на `делится`) вводится понятие наименьшего общего кратного. А именно, **общим кратным** чисел $x$ и $y$ называется такое число $m \in \mathbb{Z}$, что $m \vdots x$, $m \vdots y$. **Наименьшим общим кратным** этих чисел называется такое их общее кратное $m$, которое делит любое другое их общее кратное, иными словами, из того, что $z \vdots x$ и $z \vdots y$ следует $z \vdots m$. Наименьшее общее кратное элементов $x$ и $y$ обычно обозначается $\mathrm{НОК}(x,y)$ или $\mathrm{lcm}(x,y)$ (`least common multiple`).

Если дополнительно предполагать, что $d \geq m$, то оно определено однозначно и удовлетворяет аналогам тождеств, только что перечисленных для наибольшего общего делителя. Кроме того, для натуральных $x$, $y$ оно связано с наибольшим общим делителем соотношением $\gcd(x,y)\,\mathrm{lcm}(x,y) = xy$.

Перечислим некоторые команды языка `Mathematica`, связанные с только что введенными понятиями.

| | |
|---|---|
| `Divisors[n]` | список делителей $n$ |
| `GCD[m,n]` | наибольший общий делитель $m$ и $n$ |
| `ExtendedGCD[m,n]` | линейное представление GCD |
| `LCM[m,n]` | наименьшее общее кратное $m$ и $n$ |

Функции `GCD` и `LCM` имеют атрибуты `Flat` и `Orderless` и, поэтому, могут вызываться с любым количеством аргументов. Функция `ExtendedGCD[x,y]` возвращает ответ в формате `{d,{a,b}}`, где $d$ — наибольший общий делитель $x$ и $y$, а $a$ и $b$ — коэффициенты его линейного представления, $d = ax + by$.

**1.1.** Верно ли, что операции взятия наибольшего общего делителя и наименьшего общего кратного дистрибутивны друг относительно друга? Иными словами, всегда ли выполняются равенства

$$\gcd(\mathrm{lcm}(x,y),z) = \mathrm{lcm}(\gcd(x,z)\gcd(y,z),$$

$$\mathrm{lcm}(\gcd(x,y),z) = \gcd(\mathrm{lcm}(x,z)\,\mathrm{lcm}(y,z)?$$

**1.2.** Убедитесь, что если $m$ нечетно, то $2^m-1$ и $2^n+1$ взаимно просты.

**1.3.** Найдите наибольший общий делитель всех чисел, получающихся из данного числа всевозможными перестановками его цифр.

**1.4.** Для каждого $n$ найдите наименьшее $m$ такое, что $n$ делит $m^m$.

**1.5.** Сколько существует натуральных чисел $\leq 10000$, для которых разность $2^x - x^2$ не делится на 7?

**1.6.** Многие числа вида $2^n+1$ делятся на $2^2-1=3$. Может ли $2^n+1$ делиться на $2^m-1$ при $m \geq 3$?

**1.7.** Пусть $n$ произвольное натуральное число. Существует ли делящееся на $n$ число, в десятичную запись которого входят только нули и единицы?

## 2. Деление с остатком.[136]

Основное известное с глубокой древности свойство целых чисел состоит в том, что если $m, n \in \mathbb{Z}$, причем $n > 0$, то существуют *единственные* $q, r \in \mathbb{Z}$ такие, что $m = qn + r$, где $0 \leq r < n$. Число $q$ называется **неполным частным** (`quotient`) при делении $m$ на $n$, а число $r$ — **остатком** (`remainder`). Операция, вычисляющая $q$ и $r$ по $m$ и $n$ называется **делением с остатком**. Внутренние функции `Quotient` и `Mod` как раз и ищут по числам $m$ и $n$ их неполное частное и остаток. Заметим, что неполное частное двух целых чисел это в точности *целая часть* их частного, так что `Quotient[m,n]` всегда дает тот же результат, что `Floor[m/n]`.

| | |
|---|---|
| `Quotient[m,n]` | неполное частное при делении $m$ на $n$ |
| `Mod[m,n]` | остаток от деления $m$ на $n$ |

**Предостережение.** По этому поводу стоит подчеркнуть, что ПОДАВЛЯЮЩЕЕ БОЛЬШИНСТВО русскоязычных КНИГ ПО ПРОГРАММИРОВАНИЮ абсолютно НЕВОЗМОЖНО ИСПОЛЬЗОВАТЬ ровно потому, что переводчики не улавливают разницы между словами `ratio` — частное и `quotient` — неполное частное. Полная утрата смысла при переводе с одного языка на другой явление весьма обычное. Например, большинство словарей тракуют слова `accuracy` и `precision`, `speed` и `velocity` как синонимы. Между тем, эти слова не имеют между собой ничего общего: `accuracy` имеет размерность измеряемой величины, а `precision` — величина безразмерная; `velocity` обозначает вектор (скорость), а `speed` — скаляр (модуль скорости). Переводчики компьютерной литературы всегда характеризовались счастливым сочетанием полного незнания английского языка, полного незнания русского языка и полного непонимания смысла происходящего[137]. Именно отсюда возникают всякие анекдотические `остаточные классы` вместо правильных `классов вычетов` и тому подобная галиматья.

**2.1.** Дайте рекурсивные определения функций `Quotient` и `Mod`.

**Решение.** Например, так

```
q[0,m_]:=0; r[0,m_]=0;
q[n_,m_]:=q[n-1,m]+If[r[n-1,m]+1==m,1,0];
q[n_,m_]:=(r[n-1,m]+1)*If[r[n-1,m]+1<m,1,0];
```

**2.2.** Убедитесь, что произведение пяти последовательных целых чисел делится на 120.

**2.3.** Убедитесь, что $m!n!$ делит $(m+n)!$.

**2.4.** Какой остаток дает число $2^{2019} - 1$ при делении на $2^{20} - 1$?

---

[136] There is division betwixt the Dukes, and a worse matter than that. ©William Shakespeare, *King Lear*

[137] Впрочем, в последние годы уточнение `компьютерной литературы` стало излишним.

**2.5.** Какой остаток дает число 20192019 . . . 2019 (100 раз) при делении на 133?

**2.6.** Убедитесь, что остаток от деления простого числа на 30 равен 1 или простому числу. Какое свойство числа 30 при этом используется?

По умолчанию остаток при делении $m$ на $n$, возвращаемый функцией `Mod`, лежит между 0 и $n-1$. Однако, во многих теоретико-числовых и комбинаторных алгоритмах полезно использовать **деление с отступом** $d$, когда $m$ представляется в виде $m = qn + r$, где $d \le r < d + n$.

| | |
|---|---|
| `Quotient[m,n,d]` | неполное частное при делении $m$ на $n$ с отступом $d$ |
| `Mod[m,n,d]` | остаток от деления $m$ на $n$ с отступом $d$ |

Таким образом, по определению `Mod[m,n,d]=m-Quotient[m,n,d]*n`. В комбинаторных алгоритмах особенно часто используется отступ 1, а в теоретико-числовых — отступ $-n/2$ ("симметричная система вычетов").

**2.7.** Дайте рекурсивные определения `Quotient[m,n,d]` и `Mod[m,n,d]`.

**3. Модулярная арифметика.**[138]

Рассмотрим некоторое натуральное число $m$. Говорят, что $x, y$ **сравнимы по модулю** $m$, и пишут $x \equiv y \pmod m$, если их разность делится на $m$. Это означает в точности, что $x$ и $y$ дают одинаковые остатки при делении на $m$.

Например, два числа сравнимы по модулю 2 в том и только том случае, когда они оба одновременно четны или оба одновременно нечетны. Два числа сравнимы по модулю 3 в том и только том случае, когда они оба одновременно делятся на 3, оба одновременно дают остаток 1 или оба одновременно дают остаток 2 при делении на 3.

Отношение сравнимости по модулю $m$ представляет собой отношение эквивалентности на $\mathbb{Z}$. Иными словами, оно обладает следующими свойствами.

- **Рефлексивность**: $x \equiv x \pmod m$.
- **Симметричность**: $x \equiv y \pmod m \iff y \equiv x \pmod m$.
- **Транзитивность**:

$$x \equiv y \pmod m \quad \text{и} \quad y \equiv z \pmod m \qquad \Longrightarrow \qquad x \equiv z \pmod m.$$

Классы этой эквивалентности имеют вид $x = x \mod m = x + m\mathbb{Z}$, они состоят из всех чисел, дающих при делении на $m$ тот же остаток, что $x$. Эти классы называются **классами вычетов** по модулю $m$.

---

[138] A mathematician called Ben

Could only count modulo ten

He said 'When I go

Past my last little toe

I have to start over again.'

Так как остаток при делении любого целого числа на $m > 0$ может принимать лишь значения $0, 1, 2, \ldots, m-1$, то имеется ровно $m$ классов вычетов по модулю $m$, а именно, классы $0, 1, 2, \ldots, m-1$. Множество классов вычетов по модулю $m$ обозначается обычно $\mathbb{Z}/m\mathbb{Z}$ и называется **кольцом классов вычетов** по модулю $m$.

Любое множество представителей этих классов называется **полной системой вычетов** по модулю $m$. Очевидным примером полной системы вычетов по модулю $m$ является набор $0, 1, 2, \ldots, m-1$, но в некоторых вопросах удобнее брать другие системы вычетов, например, для упрощения вычислений в качестве системы вычетов по модулю $m = 2l + 1$ обычно удобнее брать $-l, \ldots, -1, 0, 1, \ldots, l$.

В действительности, отношения сравнения по фиксированному модулю $m$ является не просто отношением эквивалентности на $\mathbb{Z}$, а **конгруэнцией**. Иными словамит, оно согласовано с основными арифметическими операциями.

- Если $x \equiv y \pmod m$ и $z \equiv w \pmod m$, то $x + z \equiv y + w \pmod m$.
- Если $x \equiv y \pmod m$ и $z \equiv w \pmod m$, то $xz \equiv yw \pmod m$.

Эти свойства утверждают, что формулы сложения и умножения по модулю $m$

$$\overline{x} + \overline{y} = \overline{x + y}, \qquad \overline{x} \cdot \overline{y} = \overline{xy},$$

служат **корректными** (не зависящими от выбора представителей) определениями операций на множестве классов вычетов $\mathbb{Z}/m\mathbb{Z}$. Легко проверить, что эти операции задают на $\mathbb{Z}/m\mathbb{Z}$ структуру коммутативного кольца с 1.

Класс $\overline{x}$ элемента $x$ по модулю $m$ в том и только том случае обратим в $\mathbb{Z}/m\mathbb{Z}$, когда $x$ взаимно прост с $m$. Кольцо классов вычетов $\mathbb{Z}/m\mathbb{Z}$ по модулю $m$ в том и только том случае является полем, когда $m = p$ — простое число. Построенное нами поле $\mathbb{Z}/p\mathbb{Z}$ из $p$ элементов обозначется обычно $\mathbb{F}_p$ и называется полем из $p$ элементов или простым полем характеристики $p$. Иногда оно обозначается $\mathbb{F}_(p)$ и называется полем Галуа из $p$ элементов (при этом `GF` является сокращением от `Galois Field`: *In Galois Fields full of flowers primitive elements dance for hours*.

Приведем для примера таблицы операций по модулям 2, 3 и 4. Иными словами, мы строим кольца $\mathbb{Z}/2\mathbb{Z} = \mathbb{F}_2 = \{0, 1\}$, $\mathbb{Z}/4\mathbb{Z} = \mathbb{F}_3 = \{0, 1, 2\}$ и $\mathbb{Z}/4\mathbb{Z} = \{0, 1, 2, 3\}$ из двух, трех и четырех элементов. При этом для краткости мы опускаем черту над классом и пишем просто $x$ вместо $\overline{x}$.

| $+$ | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 0 |

| $\times$ | 0 | 1 |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |

| $+$ | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 0 | 1 | 2 |
| 1 | 1 | 2 | 0 |
| 2 | 2 | 0 | 1 |

| $\times$ | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 2 |
| 2 | 0 | 2 | 1 |

| + | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 0 | 1 | 2 | 3 |
| 1 | 1 | 2 | 3 | 0 |
| 2 | 2 | 3 | 0 | 1 |
| 3 | 3 | 0 | 1 | 2 |

| × | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 2 | 3 |
| 2 | 0 | 2 | 0 | 2 |
| 3 | 0 | 3 | 2 | 1 |

**3.1.** Реализуйте операции сложения и умножения по модулю $m$.

**3.2.** Постройте таблицы сложения и умножения по модулям 5, 6 и 7.

**4. Алгоритм Эвклида.**[139,140]

Из основной теоремы арифметики вытекает, что наибольший общий делитель двух чисел моментально находится, если известно их разложение на простые множители. Однако, практическое разложение на простые множители является совершенно нетривиальной задачей. Тем более замечательно, что с глубокой древности известны быстрые алгоритмы нахождения наибольшего общего делителя и его линейного представления, не требующие разложения на простые множители! Эти алгоритмы основаны на наблюдении, что $\gcd(x, y) = \gcd(x - y, y) = \gcd(y, x - y)$.

Традиционно эти алгоритмы известны под собирательным именем **алгоритма Эвклида**. Версия этого алгоритма была известна в Греции примерно за два века до Эвклида, а его — более эффективные! — варианты с незапамятных времен использовались в древнем Египте, Китае и Индии. Алгоритм Эвклида и его модификации и сегодня остаются одним из основных инструментов модулярной арифметики. Кроме того, он теснейшим образом связан со многими классическими вопросами математики, в частности, с непрерывными дробями.

Описанный в *Элементах* Эвклида алгоритм по сути состоит в следующем[141]. Пусть $x, y \in \mathbb{Z}$, заменяя $x$ и $y$ на их абсолютные величины и пользуясь тем, что $\gcd(x, y) = \gcd(y, x)$, можно с самого начала считать, что $0 \leq y \leq x$. Если $y = 0$, то $\gcd(x, y) = x$. Если же $y \neq 0$, то заменяя $(x, y)$ на $(y, x - y)$, мы получим пару чисел с тем же наибольшим общим делителем. При этом большее из чисел $y$, $x - y$ *меньше*, чем $x$. Повторяя эту процедуру, мы будем получать все меньшие и меньшие пары натуральных чисел. Ясно, что этот процесс должен оборваться на конечном шаге, а оборваться он может только на паре вида $(d, 0)$. Так как наибольший общий делитель пары при этом не изменился, то $\gcd(x, y) = \gcd(d, 0) = d$.

---

[139] The Euclidean algorithm is the granddaddy of all algoriths, because it is the oldest nontrivial algorithm that has survived to the present day.©Donald Knuth

[140] Trespassers will be shot, survivors will be shot again!

[141] Разумеется, Эвклид не рассматривал ни отрицательных чисел, ни нуля, да и единица была числом лишь с большой натяжкой, так что его формулировки назойливо репетитивны и тяжеловесны, но *по существу* в предложениях 1 и 2 Книги VII говорится именно это.

**4.1.** Напишите программу, реализующую первоначальный алгоритм Эвклида.

**Решение.** Да чего там, нужно просто перечислить все использованные свойства наибольшего общего делителя, а дальше система сама решит, что делать:

```
gcd[x_,y_]:=gcd[Abs[x],Abs[y]] /; x<0||y<0
gcd[x_,0]:=x
gcd[x_,y_]:=gcd[y,x] /; x<y
gcd[x_,y_]:=gcd[y,x-y] /; x>=y
```

Напомним, что `/;` = `Condition` вызываемое в формате `lhs:=rhs /; test` задает **присваивание с условием**. При этом `lhs` полагается равным `rhs` только если `test` дает значение `True`.

Сегодня вместо вычитания в алгоритме Эвклида обычно используется деление с остатком. А именно, если $x = qy + r$, то $\gcd(x, y) = \gcd(y, r)$, поэтому на шагом алгоритма Эвклида может быть замена пары $(x, y)$ на пару $(y, r)$. Именно в таком варианте этот алгоритм обычно излагается в учебниках алгебры.

**4.2.** Напишите программу, реализующую версию алгоритма Эвклида, использующую деление с остатком.

Уже в самом примитивном виде алгоритм Эвклида является *несравненно* более эффективным способом нахождения наибольшего делителя двух чисел, чем разложение этих чисел на множители.

**4.3.** Сравните время работы реализованного Вами алгоритма Эвклида на парах случайных целых чисел с несколькими десятками разрядов и время работы встроенной функции `FactorInteger`.

Эффективность работы алгоритма Эвклида в массовом случае основана на том, что с очень большой вероятностью наибольший общий делитель двух *случайных* чисел очень мал.

**4.4.** Убедитесь, что с вероятностью $> 99\%$ наибольший общий делитель двух случайных 100-разрядных чисел меньше 1000.

В действительности, знаменитая теорема Дирихле 1849 года утверждает, что с вероятностью $6/\pi^2 \approx 0.607927$ два случайных целых числа взаимно просты.

**4.5.** Проверьте утверждение этой теоремы для случайных 100-разрядных чисел.

**4.6.** Убедитесь, что для любого натурального $n$ существуют натуральные числа $x$ и $y$ такие, что в алгоритме Эвклида требуется ровно $n$ делений.

**4.7.** Убедитесь, что наименьшие натуральные числа, для которых в алгоритме Эвклида требуется ровно $n$ делений, это числа Фибоначчи $x = F_{n+2}$, $y = F_{n+1}$.

Однако, алгоритм Эвклида далеко не оптимален и его легко улучшить. Во-первых, алгоритм сходится несколько быстрее, если на каждом шаге

брать в нем не неотрицательный, а наименьший по модулю остаток — вот где понадобится деление с отступом!

**4.8.** Напишите программу, реализующую алгоритм Эвклида с выбором на каждом шаге наименьшего по модулю остатка.

**Указание.** Как и в предыдущей задаче используйте функцию `Mod`, но теперь не от двух, а от трех аргументов.

Еще более существенная экономия получается, если применять деление с остатком только к нечетным числам, а для четных чисел использовать деление пополам. А именно,

- если оба числа $x, y$ четны, то $\gcd(x, y) = 2\gcd(x/2, y/2)$;
- если четно ровно одно из них, скажем $y$, то $\gcd(x, y) = \gcd(x, y/2)$.

Таким образом, вычисление наибольшего общего делителя двух целых чисел сводится к делению пополам и вычислению наибольшего общего делителя двух *нечетных* чисел. В классическом китайском труде *Математика в девяти книгах* содержится алгоритм вычисления $\gcd(x, y)$, основанный на этой идее. Разумеется, фактически и там не производилось никакого деления с остатком, а меньшее из чисел $x$ или $y$ просто *вычиталось* из большего. В отличие от классического алгоритма Эвклида в данном случае вычитание может быть даже лучше, чем деление с остатком, так как разность двух нечетных чисел четна и, значит, алгоритм сходится очень быстро. Этот алгоритм называется **бинарным алгоритмом**.

**4.9.** Напишите программу, реализующую классический китайский бинарный алгоритм. То же для варианта бинарного алгоритма, использующего деление с остатком вместо вычитания.

В учебниках `высшей алгебры` обычно говорится, что проделав **обратный ход** в алгоритме Эвклида можно найти линейное представление $d = ax + bx$ наибольшего общего делителя $d = \gcd(x, y)$ чисел $x$ и $y$. Однако, классически известно, что совсем небольшая модификация алгоритма Эвклида позволяет искать наибольший общий делитель $d$ *одновременно* с его линейным представлением. Эта модификация, известная как **расширенный алгоритм Эвклида** = `extended Euclid algorithm`[142], была развита индийскими математиками V–VI века в связи с китайской теоремой об остатках и в явном виде описана Бхаскара I. *Единственное* изменение, которое при этом необходимо внести в обычный алгоритм Эвклида, состоит в том, что вместо целых чисел рассматриваются строки длины 3 с целыми компонентами, последняя из которых отвечает исходному числу, а первые две — коэффициентам его представления как линейной комбинации $x$ и $y$. Таким образом, алгоритм начинается со строк $(1, 0, x)$ и $(0, 1, y)$. Шаг алгоритма состоит в том, что строки $(u, v, x)$ и $(w, z, y)$ заменяются на

- строки $(w, z, y)$ и $(u - w, v - z, x - y)$ в варианте, использующем вычитание;

[142] Откуда `ExtendedGCD`. Иногда, в том числе и в русском переводе книги Кнута, *ошибочно* называется `обобщенным` алгоритмом Эвклида.

• строки $(w, z, y)$ и $(u - qw, v - qz, r)$, где $x = qy + r$, в варианте, использующем деление с остатком.

Так как при этом третья координата ведет себя так же, как остатки в обычном алгоритме Эвклида, то через конечное число шагов мы неминуемо придем к строкам вида $(a, b, d)$ и $(u, v, 0)$, в этот момент алгоритм останавливается и мы можем заключить, что $d = \gcd(x, y)$ и $d = ax + by$.

**4.10.** Напишите программы, реализующие расширенный алгоритм Эвклида в обоих вариантах.

Отметим две замечательные особенности этого алгоритма. Во-первых, он является самопроверяющимся, так как наибольший делитель, это в точности общий делитель, допускающий линейное представление. Во-вторых, он дает полезный субпродукт (= `by-product`). А именно, тот факт, что вторая строка, получающаяся на момент остановки алгоритма, равна $(u, v, 0)$, означает, что $ux + vy = 0$.

**4.11.** Что можно сказать об $u$ и $v$, получающихся на момент остановки расширенного алгоритма Эвклида?

**Указание.** Проведите эксперимент!

Все описанные выше алгоритмы моментально обобщаются на случай любого количества чисел. Конечно, можно воспользоваться ассоциативностью операции gcd и искать наибольший общий делитель нескольких чисел по индукции, например, для трех чисел так $\gcd(x, y, z) = \gcd(\gcd(x, y), z)$.

**4.12.** Реализуйте рекурсивный алгоритм для вычисления наибольшего общего делителя нескольких чисел и его линейного представления.

Для небольших примеров рекурсивный алгоритм будет работать, но легко предложить гораздо более эффективные алгоритмы!

**4.13.** Реализуйте несколько различных вариантов алгоритма Эвклида для нескольких чисел и сравните скорость их работы с рекурсивным алгоритмом.

**4.14.** Реализуйте расширенный алгоритма Эвклида для нескольких чисел.

**Указание.** Для $s$ чисел этот алгоритм оперирует с $s$ целочисленными строками длины $s + 1$, но имеются различные варианты организации шага алгоритма.

К счастью, для всех практических целей эти потуги ускорить работу алгоритма Эвклида излишни, так как мы можем с успехом использовать внутренние функции `GCD` и `ExtendedGCD`. Имплементация этих функций включает не только эффективную комбинацию алгоритма Эвклида и бинарного алгоритма, но и быстрые алгоритмы точной арифметики, развитые в последние 15–20 лет.

## 5. Китайская теорема об остатках.[143]

[143] Я не слыхал рассказов Оссиана,

Не пробовал старинного вина

©Осип Мандельштам

Сейчас мы установим фундаментальный результат, который позволяет

• сводить изучение сравнений к примарному случаю,

• сводить вычисления по большому модулю к вычислениям по нескольким маленьким модулям.

Этот результат является основным инструментом при проведении быстрых вычислений с очень большими числами.

Для случая произвольного количества *взаимно простых* модулей этот результат был известен Сунь Цзу в первом веке нашей эры и использовался для логистических и астрономических вычислений. Для случая двух модулей излагаемый ниже алгоритм в явном виде содержится в трактате Ариабхаты 499 года. В 1247 году Чин Чжу-Шао обобщил теорему на случай произвольного количества не обязательно взаимно простых модулей.

Начнем с ключевого случая двух взаимно простых модулей, который позволяет провести индукцию. Пусть $m \perp n$ — два взаимно простых модуля. Тогда **китайская теорема об остатках** утверждает, что для любых $a$, $b$ существует такое $x$, что

$$\begin{cases} x \equiv a \pmod{m} \\ x \equiv b \pmod{n} \end{cases}$$

причем это $x$ единственно по модулю $mn$. Иными словами, если дополнительно предполагать, что $0 \leq x < mn$, то $x$ единственно.

Начнем с простого но важного замечания, что

$$x \equiv y \pmod{m} \quad \text{и} \quad x \equiv y \pmod{n} \qquad \Longrightarrow \qquad x \equiv y \pmod{mn}.$$

В самом деле, $x - y$ делится как на $m$ так и на $n$, а, значит, делится на $\operatorname{lcm}(m, n)$. Но $m$ и $n$ взаимно просты, так что $\operatorname{lcm}(m, n) = mn$. Тем самым, единственность $x$ очевидна и существование следует теперь из принципа Дирихле! Это значит, что если мы просто переберем *все* числа от 0, до $mn - 1$, то мы обязательно наткнемся на решение этой системы.

**5.1.** Реализуйте полный перебор, находящий $x$ в китайской теореме для случая двух модулей.

Впрочем, описанный метод решения трудно назвать *эффективным* алгоритмом. Хотелось бы явно предъявить *какое-то* решение этой системы. Так как $m \perp n$, то найдутся такие $c$ и $d$, что $cm + dn = 1$. Их можно найти, например, при помощи расширенного алгоритма Эвклида. Теперь в качестве $x$ можно взять $x = dna + cmb$. В самом деле,

$$x \equiv dna + cma \equiv a \pmod{m}, \qquad x \equiv dnb + cmb \equiv b \pmod{n}.$$

Однако, указанное решение может быть $\geq mn$. Чтобы получить решение, удовлетворяющее неравенствам $0 \leq x < mn$, нужно еще взять остаток $dna + cmb$ по модулю $mn$.

**5.2.** Реализуйте алгоритм, вычисляющий $x$ в китайской теореме для случая двух модулей.

В действительности, такой алгоритм реализован в пакете

`NumberTheory‘NumberTheoryFunctions‘`

под именем `ChineseRemainder`. Для случая двух модулей эта функция вызывается в формате `ChineseRemainder[{a,b},{m,n}]`.

С математической точки зрения эта теорема утверждает, что $\mathbb{Z}/mn\mathbb{Z} \cong \mathbb{Z}/m\mathbb{Z} \oplus \mathbb{Z}/n\mathbb{Z}$, иными словами, вычисления по модулю $mn$ *полностью* сводятся к вычислениям отдельно по модулю $m$ и по модулю $n$.

Китайская теорема об остатках непосредственно обобщается на случай любого количества модулей. А именно, пусть $m = m_1 \dots m_s$, где $m_i$ попарно взаимно просты, а $x_1, \dots, x_s$ — произвольные целые числа. Тогда существует такое $x$, что

$$\begin{cases} x \equiv x_1 \pmod{m_1} \\ \dots \\ x \equiv x_s \pmod{m_s} \end{cases}$$

причем это $x$ единственно по модулю $m$. Иными словами, утверждается, что если $m_i$ попарно взаимно просты, то имеется изоморфизм колец

$$\mathbb{Z}/m_1 \dots m_s\mathbb{Z} \cong \mathbb{Z}/m_1\mathbb{Z} \oplus \dots \oplus \mathbb{Z}/m_s\mathbb{Z},$$

так что вычисления по модулю $m = m_1 \dots m_s$ полностью сводятся к вычислениям отдельно по модулю каждого из $m_i$.

Проще всего доказать китайскую теорему по индукции. Она же, естественно, дает и алгоритм для нахождения $x$. А именно, предположим, что мы уже умеем решать аналогичную систему для $s-1$ взаимно простого модуля. Пусть $y$ удовлетворяет первым $s-1$ сравнению по модулям $m_1, \dots, m_{s-1}$. Тогда $x$ можно искать как решение системы

$$\begin{cases} x \equiv y \pmod{m_1 \dots m_{s-1}} \\ x \equiv x_s \pmod{m_s} \end{cases}$$

**5.3.** Реализуйте рекурсивный алгоритм, находящий $x$ в китайской теореме для любого количества модулей.

Впрочем, можно не использовать индукции, а сразу указать решение. Для этого положим $n_i = m_1 \dots \widehat{m}_i \dots m_s$, и заметим, что числа $n_i$ взаимно просты *в совокупности*. Тем самым, существуют $a_i$ такие, что $a_1n_1 + \dots + a_sn_s = 1$, их можно найти, например, при помощи обобщенного алгоритма Эвклида. Остается положить

$$x = a_1n_1x_1 + \dots + a_sn_sx_s,$$

и, если мы хотим получить каноническое решение, поделить это $x$ с остатком на $m_1 \dots m_s$

**5.4.** Реализуйте алгоритм, непосредственно вычисляющий $x$ в китайской теореме для любого количества модулей.

После подгрузки упомятнутого выше пакета $x$ можно искать вызывая функцию `ChineseRemainder` в формате

`ChineseRemainder[{x1,...,xs},{m1,...,ms}]`.

**5.5.** Пусть $x_s \equiv i \pmod{p_i}$ для первых $s$ простых $p_i$, причем $0 \leq x < p_1 \dots p_s$. Верно ли, что при любом $s \geq 2$ это $x_s$ является простым?

**Ответ.** Первые шесть из них действительно простые:

$$x_2 = 5,\ x_3 = 23,\ x_4 = 53,\ x_5 = 1523,\ x_6 = 29243,\ x_7 = 299513,$$

но вот $x_8 = 4383593 = 23 \cdot 190591$ простым не является. Следуюшее $x_9 = 188677703$ снова простое, но вот $x_{10} = 5765999453 = 41 \cdot 131 \cdot 809 \cdot 1327$ имееи четыре простых делителя. Дальше простые встречаются, но довольно редко, $x_{18}$, $x_{63}$, $x_{105}$ и т.д.

**5.6.** Пусть $x_s \equiv (-1)^{i-1} \pmod{p_i}$ для первых $s$ простых $p_i$, причем $0 \leq x < p_1 \dots p_s$. Верно ли, что при любом $s \geq 2$ это $x_s$ является простым?

**Ответ.** Первые пять из них действительно простые:

$$x_2 = 5,\ x_3 = 11,\ x_4 = 41,\ x_5 = 881,\ x_6 = 14741,$$

но $x_7 = 74801 = 131 \cdot 571$ простым не является.

Для практически встречающихся задач (несколько сотен относительно небольших модулей) описанный выше классический алгоритм вполне эффективен. Однако, он требует нахождения линейного представления 1 и вычисления остатка по модулю $m = m_1 \dots m_s$. Для *действительно* больших чисел обычно используется более эффективный **алгоритм Гарнера**[144], который не требует деления на $m$.

> The reader will be content to wait for a full explanation of these matters till the next year, — when a series of things will be laid open which he little expects.
>
> Laurence Sterne, *Tristram Shandy*

[144] H.Garner, The residue number system. — IRE Trans. Electronic Computers, 1959, vol.8, p.140–147.

# Глава 8. КОМБИНАТОРИКА И ДИСКРЕТНАЯ МАТЕМАТИКА

Истинное учение начинается с сохранения знания, овладения знанием и понимания. Оно не начинается с любви, усилия или действия, ибо истинные любовь, усилие и действие становятся возможными лишь благодаря истинному знанию.

Идрис Шах. *Путь суфия*

По дороге в царство Чу Конфуций вышел из леса и увидел Горбуна, который ловил цикад так ловко, будто подбирал их с земли.

— Неужто ты так искусен? Или у тебя есть Путь? — спросил Конфуций.

— У меня есть Путь, — ответил Горбун. — В пятую-шестую луну, когда наступает время охоты на цикад, я кладу на кончик своей палки шарики. Если я смогу положить друг на друга два шарика, я не упущу много цикад. Если мне удастся положить три шарика, я упущу одну из десяти, а если я смогу удержать пять шариков, то поймаю всех без труда. Я стою, словно старый пень, руки держу, словно сухие ветки. И в целом огромном мире, среди всей тьмы вещей, меня занимают только крылатые цикады. Я не смотрю по сторонам и не променяю крылышки цикады на все богатства мира. Могу ли я не добиться желаемого?

Конфуций повернулся к ученикам и сказал: *Помыслы собраны воедино, дух безмятежно спокоен*. Не об этом ли Горбуне сказано такое?

Чжуан-цзы, Гл. XIX. *Постигший жизнь*

Интеллект обычно определяется как способность к пониманию ... Он состоит в возможности использовать *осознанные* знания при столкновении с *новыми* ситуациями и *предвидеть* возникновение проблем благодаря абстрактному осмыслению взаимосвязей, выраженных в символах. Подобно воображению и интуиции, интеллект работает путем комбинирования фактов, хранящихся в памяти, но делает он это не с помощью причудливой игры в потемках бессознательного, а с помощью логического анализа при полном свете сознания. Его основными инструментами являются: логика, память, способность к концентрации внимания на одной проблеме вместе с ее логическими следствиями, способность к абстракции, пренебрежение ко всему, что не относится к делу.

Ганс Селье. *От мечты к открытию*

Быть магом, — продолжал дон Хуан, — не означает заниматься колдовством, воздействовать на людей или насылать на них демонов. Это означает достижение того уровня осознания, который делает доступным непостижимое.

Карлос Кастаньеда, *Активная сторона бесконечности*

Корень учения один, но ветви его отстоят далеко друг от друга. Чтобы не погубить себя и вернуть себе потерянное, нужно вернуться к общему корню.

Ле-цзы, Гл. VIII. *Рассказы о совпадениях*

Читателю полезно изучить как можно больше программ, приведенных в данной книге, поскольку чрезвычайно важно приобрести опыт чтения программ, написанных другими. Но, к сожалению, подобной формой обучения пренебрегали во многих компьютерных курсах, что привело к крайне неэффективному использованию компьютерной техники.

Дональд Кнут

## § 1. Комбинаторика

Исчислите все общество сынов Израилевых по родам их, по семействам их, по числу имен, всех мужеского пола поголовно.

*Числа*

— Вы, может, знаете — был такой поэт Мандельштам. Так вот, он писал в одном стихотворении: “Бессонница, Гомер, тугие паруса — я список кораблей прочел до середины ... ” Это, значит, из “Илиады”, про древнегреческий флот в Средиземном море. Мандельштам только до середины дошел, а Вадим Степанович этот список читал до самого конца. Вы можете себе это представить?

Виктор Пелевин, *Греческий вариант*

Широкая распространенность на математических олимпиадах задач по комбинаторике вызывает недовольство участников, да и просто хороших людей.

Федор Петров[145]

В настоящей параграфе мы обсуждаем, в основном в чисто калькулятивных аспектах, основные классы комбинаторных коэффициентов. Впрочем, в некоторых случаях знание комбинаторного смысла абсолютно необходимо для понимания и построения алгоритмов.

**1. Факториалы.**[146]

Напомним, что **факториал** $n!$ неотрицательного целого числа $n \in \mathbb{N}_0$ определяется как $n! = 1\cdot 2\cdot 3\cdot \ldots \cdot n$. Для $n = 0$ произведение пусто, поэтому $0! = 1$. **Гамма-функция Эйлера** $\Gamma(z)$ является обобщением факториала на произвольное комплексное значение аргумента. В натуральных точках $n! = \Gamma(1 + n)$.

| | | |
|---|---|---|
| `Factorial[n]` | $n!$ | факториал |
| `Gamma[x]` | $\Gamma(x)$ | гамма-функция Эйлера |
| `Factorial2[n]` | $n!!$ | двойной факториал |
| `Subfactorial[n]` | | субфакториал |

[145] Ф.В.Петров, Комбинаторика как раскрытие скобок. — В кн.: Задачи Санкт-Петербургской олимпиады школьников по математике 2004, Невский диалект — БХМ-Петербург, Спб, 2004, с.105–115.

[146] There is nothing new under the Sun, but there are a lot of old things we do not know. ©Ambrose Bierce

**1.1.** Составьте таблицу $n!$ при $0 \le n \le 20$.

**Ответ.** Приведем начало этой таблицы

$$0! = 1,\ 1! = 1,\ 2! = 2,\ 3! = 6,\ 4! = 24,\ 5! = 120,\ 6! = 720,\ 7! = 5040,$$
$$8! = 40320,\ 9! = 362880,\ 10! = 3628800.$$

Как отмечает Кнут, *психологически* $10! \approx 3.5 \cdot 10^6$ — это наибольшее количество случаев, которые еще можно рассматривать полным перебором. При большем количестве случаев стоит задуматься о более умных алгоритмах.

**1.2.** Напишите десять программ для вычисления факториалов, не использующих встроенную функцию `Factorial` и сравните скорость их работы.

**Решение.** Вот несколько наиболее простых программ:

```
faa[0]=1; faa[n_]:=n*faa[n-1]
fbb[n_]:=Block[{i,x=1},For[i=1,i<=n,i++,x=x*i];Return[x]]
fcc[n_]:=Apply[Times,Range[n]]
fdd[n_]:=Product[i,{i,1,n}]
fee[n_]:=Fold[Times,1,Range[n]]
```

**1.3.** Напишите программу для вычисления последней ненулевой цифры $n!$.

Для решения двух следующих задач полезно знать *главный член* в **формуле Стирлинга**, дающий грубое приближение $n!$:

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n.$$

Впрочем, грубым оно является только с точки зрения анализа, для целей дискретизации при больших $n$ его точность достаточна.

**1.4.** Что больше, $(n-1)!$ или $n^{n/2}$?

**1.5.** Задайте функцию такую, что $f(n) = m$, если $n = m!$ для некоторого $m > 0$ в противном случае.

**Ответ.** Проблема, разумеется, состоит не в том, чтобы задать такую функцию, а в том, чтобы задать ее таким образом, что $f(200000!)$ вычисляется за 1 секунду. Вот решение из книги Вольфрама, основанное на формуле Стирлинга:

```
InverseFactorial[1]:=1
InverseFactorial[n_Integer /; n>0]:=
          With[{m=Round[Log[n]/ProductLog[Log[n]/E]]-1},
                                        m /; m!===n]]
```

Напомним, что встречающаяся здесь функция `ProductLog[x]` представляет собой главное решение уравнения $ye^y = x$, удовлетворяющее дифференциальному уравнению $\dfrac{dy}{dx} = \dfrac{y}{x(1+y)}$. Эта функция очень часто возникает

при решении уравнений, в которые входят экспонента и/или логарифм. Обратите внимание на программистские уловки: паттерн с условием, использование полупрозрачной конструкции локализации переменных `With` и проверку тождества в условии. Если $m! \neq n$, эта функция остается неэвалюированной. При желании можно доопределить ее произвольным образом.

Впрочем, чтобы получить более точное приближение, дающее несколько правильных цифр уже при совсем небольших $n$, можно взять следующий член по $n$ в формуле Стирлинга:

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n\left(1+\frac{1}{12n}+\ldots\right).$$

**1.6.** Задайте функцию, выражающую правую часть формулы Стирлинга, и сравните ее значение с $n!$ для небольших $n$. Начиная с какого $n$ эта формула дает шесть правильных знаков?

Как показывает следующая задача, весьма полезно помнить, что $n! = \Gamma(1+n)$. Как мы видели в Модуле 1, в связи с характером используемых `Mathematica` алгоритмов, результат упрощения сумм и произведений часто возвращается в виде комбинации значений специальных функций (в первую очередь таких, как гипергеометрические, гамма и дзета) даже в тех случаях, когда эти значения допускают элементарное выражение.

**1.7.** Вычислите $1+\sum_{i=1}^{n} i\, i!$

**Ответ.** Это $(n+1)!$, но система, естественно, возвращает $\Gamma(n+2)$.

В качестве коэффициентов различных рядов Тэйлора, при изучении перестановок, булевых функций, да и вообще в дискретной математике и алгебре чрезвычайно часто возникает также **двойной факториал**

$$n!! = n(n-2)(n-4)...$$

Таким образом, для четного $n$ двойной факториал $n!!$ является произведением всех *четных* натуральных чисел, не превосходящих $n$, а для четного $n$ — произведением всех *нечетных* натуральных чисел, не превосходящих $n$:

$$(2n)!! = 2^n n!, \qquad (2n-1)!! = (2n)!/(2n)!!$$

**1.8.** Составьте таблицу первых $n!!$ при $0 \le n \le 21$.

**Ответ.** Вот ответ, отдельно для четных $n$:

$$0!! = 1,\ 2!! = 2,\ 4!! = 8,\ 6!! = 48,\ 8!! = 384,\ 10!! = 3840,\ 12!! = 46080,$$
$$14!! = 645120,\ 16!! = 10321920,\ 18!! = 185794560,\ 20!! = 3715891200$$

и для нечетных:

$$1!!=1,\ 3!!=3,\ 5!!=15,\ 7!!=105,\ 9!!=945,\ 11!!=10395,\ 13!!=135135,$$
$$15!!=2027025,\ 17!!=34459425,\ 19!!=654729075,\ 21!!=13749310575$$

Ясно, что для использовать выражение через $(2n)!$ для фактического вычисления $(2n-1)!!$ совершенно нецелесообразно.

**1.9.** Напишите десять программ для вычисления двойных факториалов, не использующих встроенные функции `Factorial` и `Factorial2`.

Двойные факториалы теснейшим образом связаны со значениями гамма-функции в *полуцелых* точках.

**1.10.** Вычислите несколько первых значений $\Gamma(n+1/2)$, $n\in\mathbb{N}_0$, и угадайте, чему равен ответ в общем случае.

**Ответ.** Взгляд на

$$\Gamma\left(\frac12\right)=\sqrt{\pi},\ \Gamma\left(\frac32\right)=\frac{\sqrt{\pi}}{2},\ \Gamma\left(\frac52\right)=\frac{3\sqrt{\pi}}{4},\ \Gamma\left(\frac72\right)=\frac{15\sqrt{\pi}}{8},$$
$$\Gamma\left(\frac92\right)=\frac{105\sqrt{\pi}}{16},\ \Gamma\left(\frac{11}{2}\right)=\frac{945\sqrt{\pi}}{32},\ \Gamma\left(\frac{13}{2}\right)=\frac{10395\sqrt{\pi}}{64},$$

сразу убеждает нас в том, что $\Gamma\left(n+\dfrac12\right)=\dfrac{(2n-1)!!\sqrt{\pi}}{2^n}$

Основная роль факториала состоит в том, что $n!$ есть количество перестановок степени $n$. Точно так же $(2n)!!=2^n n!$ есть количество *означенных* перестановок степени $n$. В связи с задачей Монмора о числе беспорядков степени $n$ нам встретится еще одна функция факториального типа, **субфакториал**

$$D_n=n!\sum_{i=0}^{n}\frac{(-1)^i}{i!}$$

Эта функция реализована в ядре `Mathematica` под именем `Subfactorial[n]`. Асимптотически она равна $n!/e$.

**1.11.** Напишите программу для вычисления субфакториалов и составьте таблицу первых двадцати субфакториалов.

**Ответ.** Вот начало ответа

$$D_1=0,\ D_2=1,\ D_3=2,\ D_4=9,\ D_5=44,\ D_6=265,\ D_7=1854,$$
$$D_8=14833,\ D_9=133496,\ D_{10}=1334961$$

## 2. Убывающий и возрастающий факториалы.[147]

[147] There is something in this more than natural. ©William Shakespeare, *Hamlet*

Наряду с обычными факториалами во многих комбинаторных и алгебраических задачах, задачах аппроксимации, а также при разложении различных специальных функций в ряды, возникают возрастающие и убывающие факториалы. В действительности на них естественно смотреть как на *многочлены* Здесь же убывающие и возрастающие факториалы будут интересовать главным образом как натуральные числа.

| | | |
|---|---|---|
| `Pochhammer[x,n]` | $[x]^n$ | возрастающий факториал |
| `(-1)^n*Pochhammer[-x,n]` | $[x]_n$ | убывающий факториал |

Пусть $m \in \mathbb{N}_0$. Выражение

$$[x]_m = x(x-1)\dots(x-m+1)$$

называется **убывающим факториалом** длины $m$. Для $m = 0$ произведение пусто, поэтому $[x]_0 = 1$ для всех $x$. Для $n \in \mathbb{N}_0$ имеем $[n]_m = n!/(n-m)!$

Выражение

$$[x]^m = x(x+1)\dots(x+m-1)$$

называется **возрастающим факториалом** длины $m$. Для $m = 0$ произведение пусто, поэтому $[n]^0 = 1$ для всех $x$. Для любого $x$ выполняется равенство $[x]^m = \Gamma(x+m)/\Gamma(x)$. В частности, для натурального $n$ имеем $[n]^m = (n+m-1)!/(n-1)!$

Убывающий и возрастающий факториалы находятся в **двойственности** и выражаются друг через друга формулами

$$[x]_m = [x-m+1]^m = (-1)^m[-x]^m, \quad [x]^m = [x+m-1]_m = (-1)^m[-x]_m.$$

Убывающие факториалы начал впервые систематически изучать Вандермонд, который и ввел для них обозначение $[x]^n$ (sic!) Кнут называет убывающий и возрастающий факториалы **факториальными степенями** и пользуется для них обозначением Капелли $[x]_m = x^{\underline{m}}$ и $[x]^n = x^{\overline{m}}$. Для разнообразия в `Mathematica` возрастающий факториал называется **символом Поххаммера** и обозначается $(x)_n$.

**2.1.** Напишите десять программ для вычисления убывающих и возрастающих факториалов, не использующих встроенные функции `Pochhammer`, `Factorial` и `Gamma`.

**2.2.** Вычислите $[n]_m$ и $[n]^m$ при $0 \le m, n \le 20$.

**Ответ.** Вот начало таблицы убывающих факториалов:

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 2 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 3 | 6 | 6 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 4 | 12 | 24 | 24 | 0 | 0 | 0 | 0 | 0 |
| 1 | 5 | 20 | 60 | 120 | 120 | 0 | 0 | 0 | 0 |
| 1 | 6 | 30 | 120 | 360 | 720 | 720 | 0 | 0 | 0 |
| 1 | 7 | 42 | 210 | 840 | 2520 | 5040 | 5040 | 0 | 0 |
| 1 | 8 | 56 | 336 | 1680 | 6720 | 20160 | 40320 | 40320 | 0 |
| 1 | 9 | 72 | 504 | 3024 | 15120 | 60480 | 181440 | 362880 | 362880 |

Вот начало таблицы возрастающих факториалов:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 2 | 6 | 24 | 120 | 720 | 5040 | 40320 |
| 1 | 2 | 6 | 24 | 120 | 720 | 5040 | 40320 | 362880 |
| 1 | 3 | 12 | 60 | 360 | 2520 | 20160 | 181440 | 1814400 |
| 1 | 4 | 20 | 120 | 840 | 6720 | 60480 | 604800 | 6652800 |
| 1 | 5 | 30 | 210 | 1680 | 15120 | 151200 | 1663200 | 19958400 |
| 1 | 6 | 42 | 336 | 3024 | 30240 | 332640 | 3991680 | 51891840 |
| 1 | 7 | 56 | 504 | 5040 | 55440 | 665280 | 8648640 | 121080960 |
| 1 | 8 | 72 | 720 | 7920 | 95040 | 1235520 | 17297280 | 259459200 |

Обратите внимание, что на главной диагонали первой таблицы и в первой строке второй таблицы (нумерация строк и столбцов начинается с 0) расположены обычные факториалы.

**3. Биномиальные коэффициенты.**[148]

Множество всех $m$-элементных подмножеств множества $X$ называется **$m$-й внешней степенью множества** $X$ и обозначается через $\bigwedge^m(X)$ (в комбинаторике часто используется также обозначение $X^{(m)}$):

$$\bigwedge^m(X) = \{Y \subseteq X \mid |Y| = m\}.$$

Традиционно количество $m$-элементных подмножеств $n$-элементного множества называется **числом сочетаний из $n$ по $m$** ($n$ `choose` $m$) и обозначается $\binom{n}{m}$ или $C_n^m$. Таким образом, по определению,

$$C_n^m = \binom{n}{m} = |\bigwedge^m(\underline{n})|.$$

Числа $\binom{n}{m}$ называются также **биномиальными коэффициентами**. Так как $Y \mapsto X \setminus Y$ устанавливает биекцию между $m$-элементыми подмножествами и $(n-m)$-элементными подмножествами, то биномиальные коэффициенты симметричны по второму аргументу:

$$\binom{n}{m} = \binom{n}{n-m}.$$

В системе биномиальные коэффициенты можно вычислять при помощи внутренних функций `Binomial` или `Multinomial`.

| | | |
|---|---|---|
| `Binomial[n,m]` | $\binom{n}{m} = \binom{n}{n-m}$ | биномиальный коэффициент |
| `Multinomial[m,n-m]` | $\binom{n}{m, n-m}$ | ibid. |

[148] И перекличка ворона и арфы
Мне чудится в зловещей тишине.
©Осип Мандельштам

**3.1.** Бриджевая рука представляет собой выбор 13 карт из колоды в 52 карты. Сколько существует различных бриджевых рук?

**3.2.** Известно, что

$$\binom{15}{5} = \binom{14}{6} = 3003.$$

Имеет ли уравнение

$$\binom{x+1}{y} = \binom{x}{y+1}$$

другие решения?

Основным фактом про биномиальные коэффициенты является следующее **треугольное рекуррентное соотношение**. Для любых $m$ и $n$ имеет место равенство

$$\binom{n}{m} = \binom{n-1}{m-1} + \binom{n-1}{m}.$$

Теперь, чтобы полностью восстановить все биномиальные коэффициенты, достаточно воспользоваться **граничными условиями**

$$\binom{n}{0} = \binom{n}{n} = 1.$$

Граничные условия очевидны: в $n$-элементном множестве $X$ существует единственное подмножество из 0 элементов — а именно, пустое множество $\emptyset$, и единственное подмножество из $n$ элементов — само множество $X$. Комбинаторное истолкование рекуррентного соотношения дано в следующем параграфе.

Традиционно эти соотношения кодируют в виде следующего треугольника, называемого **треугольником Паскаля**:

$$\begin{array}{ccccccccccc}
 & & & & & 1 & & & & & \\
 & & & & 1 & & 1 & & & & \\
 & & & 1 & & 2 & & 1 & & & \\
 & & 1 & & 3 & & 3 & & 1 & & \\
 & 1 & & 4 & & 6 & & 4 & & 1 & \\
1 & & 5 & & 10 & & 10 & & 5 & & 1 \\
1 & & 6 & & 15 & & 20 & & 15 & & 6 & & 1 \\
1 & & 7 & & 21 & & 35 & & 35 & & 21 & & 7 & & 1 \\
1 & & 8 & & 28 & & 56 & & 70 & & 56 & & 28 & & 8 & & 1
\end{array}$$

Каждое число здесь равно сумме двух стоящих в предыдущей строке левее и правее его, отсутствующие числа равны 0. На самом деле треугольник Паскаля был известен в Китае не позже IX века, а в Европе не позже начала XVI века — в Италии треугольник Паскаля и сегодня называется **треугольником Тартальи**. Однако, *Трактат об арифметическом треугольнике* Паскаля был самым полным сочинением на эту тему, оказавшим

влияние на Ньютона и Лейбница. Вот еще одно эффектное наблюдение, касающееся треугольника Паскаля: биномиальный коэффициент равен числу маршрутов[149] от вершины треугольника к точке, в которой расположен данный коэффициент.

**3.3.** Напишите рекуррентную программу, вычисляющую биномиальные коэффициенты с помощью треугольного рекуррентного соотношения.

**Ответ.** Будем вычислять биномиальные коэффициенты так, как строится треугольник Паскаля:

```
bina[n_,0]:=1; bina[n_,n_]:=1;
bina[n_,m_]:=If[m>n,0,bina[n-1,m-1]+bina[n-1,m]]
```

Впрочем, начиная с XIII века китайские математики изображали биномиальные коэффициенты в виде матрицы $P$ с общим элементом

$$P_{ij} = \binom{i}{j}, \qquad 0 \le i, j \le n.$$

Сегодня эта матрица обычно называется **матрицей Паскаля**.

**3.4.** Вычислите биномиальные коэффициенты $\binom{n}{m}$ при $0 \le m, n \le 20$ и составьте матрицу Паскаля порядка 11 (степени 10).

**Ответ.** Вычисление `Table[Binomial[i,j],{i,0,10},{j,0,10}]` дает следующую **таблицу**[150]:

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 3 | 3 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 4 | 6 | 4 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 5 | 10 | 10 | 5 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 6 | 15 | 20 | 15 | 6 | 1 | 0 | 0 | 0 | 0 |
| 1 | 7 | 21 | 35 | 35 | 21 | 7 | 1 | 0 | 0 | 0 |
| 1 | 8 | 28 | 56 | 70 | 56 | 28 | 8 | 1 | 0 | 0 |
| 1 | 9 | 36 | 84 | 126 | 126 | 84 | 36 | 9 | 1 | 0 |
| 1 | 10 | 45 | 120 | 210 | 252 | 210 | 120 | 45 | 10 | 1 |

Часто используется такде **симметрическая матрица Паскаля** ($PP^t$) с общим элементом

$$(PP^t)_{ij} = \binom{i+j}{j}, \qquad 0, \le i, j \le n.$$

**3.5.** Составьте симметрическую матрицу Паскаля порядка 11 (степени 10).

---

[149] Напомним, что **маршрут** — в отличие от **пути** — всегда идет в *положительных* направлениях, в данном случае вниз-влево и вниз-вправо.

[150] Запись матрицы без скобок.

**Ответ.** Вот эта матрица (записанная как таблица), в правом нижнем углу стоит $\binom{20}{10}$:

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 1 | 3 | 6 | 10 | 15 | 21 | 28 | 36 | 45 | 55 | 66 |
| 1 | 4 | 10 | 20 | 35 | 56 | 84 | 120 | 165 | 220 | 286 |
| 1 | 5 | 15 | 35 | 70 | 126 | 210 | 330 | 495 | 715 | 1001 |
| 1 | 6 | 21 | 56 | 126 | 252 | 462 | 792 | 1287 | 2002 | 3003 |
| 1 | 7 | 28 | 84 | 210 | 462 | 924 | 1716 | 3003 | 5005 | 8008 |
| 1 | 8 | 36 | 120 | 330 | 792 | 1716 | 3432 | 6435 | 11440 | 19448 |
| 1 | 9 | 45 | 165 | 495 | 1287 | 3003 | 6435 | 12870 | 24310 | 43758 |
| 1 | 10 | 55 | 220 | 715 | 2002 | 5005 | 11440 | 24310 | 48620 | 92378 |
| 1 | 11 | 66 | 286 | 1001 | 3003 | 8008 | 19448 | 43758 | 92378 | 184756 |

В щкольной математике биномиальные коэффициенты часто определяются следующей явной формулой

$$\binom{n}{m} = \frac{[n]_m}{m!} = \frac{n!}{m!(n-m)!},$$

справедливой для любых $m, n \in \mathbb{N}_0$. Обратите внимание, что эта формула, в частности, еще раз доказывает уже известный нам из Модуля 1 факт, что произведение $m$ последовательных целых чисел делится на $m!$.

**3.6.** Докажите эту формулу по индукции, используя треугольное рекуррентное соотношение.

Разумеется, на самом деле никакой индукции здесь не надо, так как эта формула сразу следует из того, что количество перестановок $n$-элементного множества равно $n!$. В самом деле, пусть $Y \subseteq X$, причем $|X| = n$, $|Y| = m$. Каждое $m$-элементное подмножество $Z \subseteq X$ является образом $Y$ под действием некоторой перестановки множества $X$, причем те перестановки, которые оставляют $Y$ на месте это произведения $m!$ перестановок множества $Y$ и $(n-m)!$ перестановок множества $X \setminus Y$. Это и значит, что всего $m$-элементных подмножеств в $X$ ровно $n!/m!(n-m)!$ штук ("теорема о связи орбит и стабилизаторов").

Формула с убывающим факториалом *лучше*, так как она позволяет распространить определение биномиального коэффициента на случай, когда $n \in \mathbb{Z}$. А именно, теперь мы можем определить

$$\binom{-n}{m} = \frac{[-n]_m}{m!} = \frac{(-1)^m[n]^m}{m!}$$

Эта формула определяет число $m$-элементных подмножеств в МНОЖЕСТВЕ ОТРИЦАТЕЛЬНОЙ МОЩНОСТИ. Мы распространили биномиальные коэффициенты на отрицательные значения аргументов, потому что получать

формулы для $\mathbb{Z}$ обычно значительно проще, чем для $\mathbb{N}$. Однако в **знакопеременных формулах** не все слагаемые имеют комбинаторный смысл, *написать* их обычно просто, а реально *вычислить* при помощи них что-то очень долго и дорого. *Основным* содержанием комбинаторики является борьба за **комбинаторные формулы**, в которых все слагаемые положительны.

**4. Разбиение на случаи.**[151]

Большинство алгебраистов тщательно различает `доказательство` и `проверку`. Какой-то факт может быть `проверен`, но не `доказан`. С этой точки зрения вычислительные доказательства тождеств для биномиальных коэффициентов, использующие выражение $\binom{n}{m}$ через факториалы, являются `проверками`, а те концептуальные доказательства, которые мы обсудим сейчас, — собственно `доказательствами`, или, как сказали бы остальные математики, `априорными доказательствами`, опирающимся только на определения. Это различие приобретает драматический характер в разделах алгебры, использующих нетривиальные классификационные теоремы.

В этом курсе мы почти не обсуждаем *концептуальные* доказательства рассматриваемых фактов. Однако, в данном случае мы *вынуждены* сделать исключение: при доказательстве тождеств для биномиальных коэффициентов и чисел Стирлинга возникают некоторые КЛЮЧЕВЫЕ СООБРАЖЕНИЯ, КОТОРЫЕ в дальнейшем ДЕСЯТКИ РАЗ ИСПОЛЬЗУЮТСЯ ПРИ ПОСТРОЕНИИ РЕКУРРЕНТНЫХ АЛГОРИТМОВ. Единственный способ написать правильную программу, порождающую разбиения множества, перестановки, сюръективные отображения, или что-нибудь в таком духе, состоит в том, чтобы понимать, что при этом происходит с *математической* точки зрения.

Первое из этих ключевых соображений называется **разбиение на случаи**. А именно, если какие-то объекты разбиты на несколько *непересекающихся* типов, то, чтобы посчитать общее количество объектов, нужно просто сложить количество объектов каждого из этих типов. В теории перечисления это соображение известно как **правило суммы**. В следующих задачах речь идет об **альтернативе**, иными словами, выборе из *двух* взаимоисключающих возможностей, в соответствии с тем, принадлежит некоторый фиксированный элемент данному подмножеству, или нет.

**4.1.** Докажите соотношение

$$\binom{n}{m} = \binom{n-1}{m-1} + \binom{n-1}{m}.$$

**Решение.** Пусть $|X| = n$. Зафиксируем какую-то точку $x \in X$. Для любого подмножества $Y \subseteq X$ имеет место следующая альтернатива: либо $x \in Y$,

[151] Colvard's Logical Premises: All probabilities are 50%. Either a thing will happen or it won't.

либо $x \notin Y$. Теперь тождество вытекает из правила суммы и следующих двух наблюдений:

• Подмножество $Y$ такое, что $x \notin Y$, однозначно определяется своим пересечением $Y \setminus \{x\} = Y \cap (X \setminus \{x\})$, которое является $(m-1)$-элементным подмножеством $(n-1)$-элементного множества $X \setminus \{x\}$.

• Подмножество $Y$ такое, что $x \in Y$, является $m$-элементным подмножеством $(n-1)$-элементного множества $X \setminus \{x\}$.

**4.2.** Вычислите сумму

$$\sum_{m=0}^{n} \binom{n}{m} = \binom{n}{0} + \binom{n}{1} + \binom{n}{2} + \ldots + \binom{n}{n}.$$

**Решение.** Эта сумма равна количеству всех подмножеств $n$-элементного множества $X$, т.е. порядку $2^X$. Хорошо известно, что $|2^X| = 2^{|X|} = 2^n$. Для полноты заметим, что это получается при помощи той же идеи, которая использована в решении первой задачи. Доказываем предложение индукцией по $n = |X|$. В качестве базы индукции можно взять случай $X = \emptyset$, когда $2^\emptyset = \{\emptyset\}$ состоит из одного элемента, что соответствует формуле $2^0 = 1$. Предположим теперь, что $X$ непусто и фиксируем точку $x \in X$. Рассмотрим, как подмножества множества $X$ расположены относительно $x$. Для подмножества $Y \subseteq X$ имеет место следующая альтернатива: либо $x \in Y$, либо $x \notin Y$.

• При $x \notin Y$ подмножество $Y$ содержится уже в $(n-1)$-элементном множестве $X \setminus \{x\}$ и по индукционному предположению имеется ровно $2^{n-1}$ таких множеств.

• При $x \notin Y$ подмножество $Y$ имеет вид $Y = Z \cup \{x\}$, для единственного $Z \subseteq X \setminus \{x\}$, так что таких множеств снова ровно $2^{n-1}$ штук.

Это и значит, что всего в $X$ ровно $2^{n-1} + 2^{n-1} = 2^n$ подмножеств, как и утверждалось.

**4.3.** Вычислите знакопеременную сумму

$$\sum_{m=0}^{n} (-1)^m \binom{n}{m} = \binom{n}{0} - \binom{n}{1} + \binom{n}{2} - \binom{n}{3} + \ldots + (-1)^n \binom{n}{n}.$$

**Решение.** Эта сумма равна 0, иными словами, подмножеств четного порядка в $X$, $|X| = n$, сколько подмножеств нечетного порядка. Если $n$ само **нечетно**, то такое соответствие задается дополнением: если $Y \subseteq X$ имеет нечетный порядок, то порядок $\overline{Y} = X \setminus Y$ четен. В общем случае снова воспользуемся той же самой идеей, что выше. А именно, фиксируем точку $x \in X$ и рассмотрим отображение $\mathrm{inv}_x : X \longrightarrow X$, определенное посредством $\mathrm{inv}_x(Y) = Y \triangle \{x\}$. Иными словами, мы *добавляем* элемент $x$ к подмножеству $Y$, если он там не содержится, и *выбрасываем* $x$ из $Y$,

если он там содержится. Ясно, что $\mathrm{inv}_x$ обратимо[152] и переводит четные подмножества в нечетные.

**4.4.** Вычислите сумму

$$\binom{2n}{0} + \binom{2n}{2} + \binom{2n}{4} + \ldots + \binom{2n}{n-1}$$

при нечетном $n$ и сумму

$$\binom{2n}{1} + \binom{2n}{3} + \binom{2n}{5} + \ldots + \binom{2n}{n-1}$$

при четном $n$.

**Указание.** Скомбинируйте результаты двух предыдущих задач и воспользуйтесь симметрией или доверьтесь внутреннему чутью `Mathematica`.

В следующей задаче используется еще одно ключевое соображение, **подсчет двумя способами**.

**4.5.** Докажите, что

$$\sum_{m=0}^{n} m \binom{n}{m} = n2^{n-1}.$$

**Решение.** Как левая, так и правая части представляют этого равенства представляют собой посчитанный двумя способами порядок множества

$$\{(x, Y) \in X \times 2^X \mid x \in Y\},$$

где $|X| = n$. А именно, как мы знаем из предыдущих задач, каждый элемент $x \in X$ содержится ровно в *половине* подмножеств $Y \subseteq X$. Всего в $X$ имеется $n$ элементов и $2^n$ подмножеств, что как раз и дает правую часть. С другой стороны, количество $m$-элементных подмножеств в $X$ равно $\binom{n}{m}$ и *каждое* из них содержит $m$ элементов — а это и есть левая часть.

**4.6.** Докажите, что

$$\sum_{m=0}^{n} (-1)^m m \binom{n}{m} = 0.$$

Вот — по принципу *музыка навеяла* — еще одна иллюстрация того, что `Mathematica` считает суммы *как умеет*: используя суммирование Бернулли—Эйлера, интегральные представления, алгоритмы Адамчика и Госпера—Цайльбергера, `pattern matching`, и все такое. В возникающем при этом ответе *как правило* фигурируют значения специальных функций, в первую очередь гипергеометрических, но не только! Чтобы получить ответ в привычной форме, нужно с ним поэкспериментировать.

**4.7.** Вычислите сумму $\displaystyle\sum_{m=1}^{n} \frac{(-1)^m}{m} \binom{n}{m}$.

[152] В действительности $\mathrm{inv}_x$ является **инволюцией**, т.е. $(\mathrm{inv}_x)^2 = \mathrm{id}_X$.

**Ответ.** В терминах `Mathematica` сумма записывается как

```
Sum[(-1)^(i-1)/i*Binomial[n,i],{i,1,n}]
```

и равна $n$-му **гармоническому числу** $H_n = \sum_{m=1}^{n} \frac{1}{m}$, представляемому в системе как `HarmonicNumber[n]`. С использованием `FullSimplify` можно убедиться, что она также равна

```
EulerGamma+PolyGamma[0,1+n]
```

где `EulerGamma` - **константа Эйлера**, а `PolyGamma[0,x]` - функция **дигамма** $\psi(x)$, встречавшиеся нам в Модуле 1. *При некотором желании* этот ответ может быть получен и элементарными методами, но мы этого делать не будем[153]

**4.8.** Докажите, что

$$\sum_{m=0}^{n} m^2 \binom{n}{m} = n(n+1)2^{n-2}.$$

**Указание.** Несколько проще вычислить сумму

$$\sum_{m=0}^{n} m(m-1) \binom{n}{m}$$

к которой можно применить тот же прием, что в предыдущей задаче. Иными словами, мы предлагаем воспользоваться равенством

$$m^2 = 2\binom{m}{2} + \binom{m}{1} = \binom{m}{2} + \binom{m+1}{2}.$$

Очевидно, что эта идея обобщается на суммы вида $\sum_{m=0}^{n} m^l \binom{n}{m}$ но, так как выражение $x^l$ через биномиальные коэффициенты использует числа Стирлинга или числа Эйлера, то мы вернемся к этой теме позже.

Вот еще одна вариация на основную тему настоящего параграфа, **тождество внесения-вынесения**.

**4.9.** Докажите тождество

$$n \binom{n-1}{m-1} = m \binom{n}{m}.$$

**Решение.** Это тождество получается, если подсчиать двумя способами порядок множества

$$\{(x, Y) \in X \times \bigwedge^m(X) \mid x \in Y\},$$

[153] см., например, Дж.Риордан. Комбинаторные тождества.— М: Наука, 1982, 255 С.

где $|X| = n$. В самом деле, количество $m$-элементных подмножеств множестве $X$ равно $\binom{n}{m}$ и каждое из них содержит $m$ элементов — это правая часть. С другой стороны, как мы видели в первой задаче, для каждого из $n$ элементов $x \in X$ имеется ровно $\binom{n-1}{m-1}$ содержащих его $m$-элементных подмножеств.

Тождество внесения-вынесения часто переписывают также в виде

$$\binom{n}{m} = \frac{n}{m}\binom{n-1}{m-1} = \frac{n}{n-m}\binom{n-1}{m}.$$

**5. Некоторые важнейшие тождества.**[154]

Известны *многие тысячи* тождеств, связанных с биномиальными коэффициентами[155]. У нас нет ни возможности, ни желания обсуждать здесь все эти тождества. Вбросим, тем не менее, пригоршню *простейших* тождеств, которыми мы будем постоянно пользоваться.

Одним из самых полезных тождеств для биномиальных коэффициентов является **соотношение Вандермонда**, называемое еще **формулой свертки Вандермонда**. Это еще одна иллюстрация разбиения на случаи, только случаев здесь не два, а $l+1$.

**5.1.** Докажите тождество

$$\binom{m+n}{l} = \sum_{i=0}^{l}\binom{m}{i}\binom{n}{l-i}.$$

**Решение.** Представим множество порядка $m+n$ как дизъюнктное объединение $X \sqcup Y$ двух множеств: $m$-элементного множества $X$ и $n$-элементного множества $Y$. По определению левая часть равна количеству $l$-элементных подмножеств $Z \subseteq X \sqcup Y$. С другой стороны, если $Z$, $|Z| = l$, то пересечение $Z \cap X$ может априори иметь любой порядок $0 \leq i \leq m$. В этом случае порядок $Z \cap Y$ равен $l - i$. Обратно, если $U \subseteq X$ и $V \subseteq Y$ два подмножества такие, что $|U| + |V| = l$, то так как $U \cap V = \emptyset$, порядок $Z = U \cup V$ равен $|Z| = |U| + |V| = l$. Имеется $\binom{m}{i}$ способов выбрать $i$-элементное подмножество $U \subseteq X$ и для каждого из них способов $\binom{n}{l-i}$ выбрать $(l-i)$-элементное подмножество $V \subseteq Y$. Осталось просуммировать все эти *взаимоисключающие* возможности.

---

[154] Il est absurde de diviser les écrivains en bons et en mauvais. D'un côté, il y a mes amis, et de l'autre, le reste (фр.) – Нелепо делить писателей на хороших и плохих. С одной стороны, есть мои друзья, с другой — все остальные. ©Philippe Soupault. *Manifeste Dada*

[155] Несколько десятков таких тождеств и много дальнейших ссылок можно найти в книге Р.Грехем, Д.Кнут, О.Паташник. Конкретная математика. Основание информатики — Мю: Мир, 1998. 703 С.

Обычное истолкование этого тождества таково. Выбрать $l$ человек из группы, в которой $m$ женщин и $n$ мужчин[156] можно $\binom{m+n}{l}$ способами. С другой стороны, можно вначале выбрать $i$ женщин $\binom{m}{i}$ способами, а потом добрать $l-i$ мужчин $\binom{n}{l-i}$. Таким образом, всего имеется $\binom{m}{i}\binom{n}{l-i}$ способов выбрать $l$ человек, из которых $i$ женщины. По *загадочным* причинам некоторые авторы называют опубликованное в 1772 году соотношение Вандермонда **тождеством Коши** — *when Mozart was my age, he was dead for six years*. Впрочем, Кнут отмечает, что Чжу Ши-Чжие опубликовал этот результат еще в 1303 году, но об этом ничего не было известно в Европе до 1950-х годов.

В частном случае $l=m=n$ тождество Вандермонда принимает следующую форму:

$$\binom{2n}{n} = \sum \binom{n}{i}^2.$$

**5.2.** Докажите следующее обобщение тождества Вандермонда:

$$\binom{n_1+n_2+\ldots+n_s}{m} = \sum_{i_1+i_2+\ldots+i_s=m} \binom{n_1}{i_1}\binom{n_2}{i_2}\ldots\binom{n_s}{i_s}.$$

**5.3.** Докажите тождество

$$\sum_{i=0}^{l}(-1)^{m+l}\binom{n}{m}\binom{m}{l} = \delta_{ml}.$$

Следующее тождество является широким обобщением тождества внесения-вынесения.

**5.4.** Докажите **электоральное тождество**

$$\binom{n}{m}\binom{m}{l} = \binom{n}{l}\binom{n-l}{m-l} = \binom{n}{l}\binom{n-l}{n-m}.$$

**Решение.** Нужно лишь посчитать двумя способами порядок множества

$$\{(Y,Z)\in \textstyle\bigwedge^m(X)\times\bigwedge^l(X) \mid Y\supseteq Z\},$$

где $|X|=n$. В самом деле, имеется $\binom{n}{m}$ способов выбрать $m$-элементное подмножество $Y\subseteq X$ и для *каждого* из них $\binom{m}{l}$ способов выбрать $l$-элементное подмножество $Z\subseteq Y$, что и дает левую часть. С другой стороны, имеется $\binom{n}{l}$ способов выбрать $l$-элементное подмножество $Z\subseteq X$ и для

[156] Это рассуждение опирается на следующее экстраматематическое предположение: среди любых трех обыкновенных людей по крайней мере двое одного пола. Подробнее по поводу этой версии принципа Дирихле см. § 3 Главы 8.

*каждого* из них $\binom{n-l}{n-m}$ способов выбрать $(m-l)$-элементное подмножество $U$ в $n-l$-элементном множестве $X\setminus Z$, дополняющее $Z$ до $l$-элементного множества.

Название этого тождества связано со следующей интерпретацией в терминах двуступенчатых выборов. Съезд партии должен избрать центральный комитет в составе $m$ голов и политбюро в составе $l$ членов.

• В партии вымогателей вначале съезд избирает центральный комитет, а потом центральный комитет *из своего состава* избирает политбюро.

• В партии воров вначале съезд избирает политбюро, а потом *дополнительно* он же избирает $m-l$ голов центрального комитета, не входящих в политбюро.

Доказанное только что тождество утверждает, что результат выборов не зависит от используемой процедуры.

Соотношение, определяющее треугольник Паскаля — это так называемое рекуррентное соотношение *треугольного типа*. Итерируя его, легко получить **диагональные рекуррентные соотношения**. Чтобы понять, почему они так называются, найдите входящие в них коэффициенты в треугольнике Паскаля.

**5.5.** Докажите, что

$$\binom{n+1}{m}=\binom{n}{m}+\binom{n-1}{m-1}+\binom{n-2}{m-2}+\ldots+\binom{n-m}{0}.$$

**Решение.** Это тождество тоже допускает простое комбинаторное истолкование. А именно, пусть $X=\{0,1,2,\ldots,n\}$. Коэффициент $\binom{n+1}{m}$ в левой части есть количество $m$-элементных подмножеств в $X$. С другой стороны, $\binom{n-i}{m-i}$ выражает количество $m$-элементных подмножеств $Y\subseteq X$ таких, что $i$ наименьший индекс, не принадлежащий $Y$.

**5.6.** Докажите, что

$$\binom{n+1}{m+1}=\binom{n}{m}+\binom{n-1}{m}+\binom{n-2}{m}+\ldots+\binom{m}{m}.$$

Найдите комбинаторное истолкование этого тождества.

**5.7.** Напишите рекуррентную программу, вычисляющую биномиальные коэффициенты с помощью диагональных рекуррентных соотношений. Какие граничные условия понадобятся Вам при этом?

**5.8.** Докажите **тождество шестиугольника**:

$$\binom{n-1}{m-1}\binom{n}{m+1}\binom{n+1}{m}=\binom{n-1}{m}\binom{n}{m-1}\binom{n+1}{m+1}.$$

Чтобы понять, почему оно так называется, найдите эти коэффициенты в треугольнике Паскаля.

## § 2. Списки и последовательности

Аз есмь Алфа и Омега, начало и конец, первый и последний.
Откровение Святого Иоанна Богослова, Глава 22–13

List, list, oh, list!
William Shakespeare, *Hamlet*

В этом параграфе мы *начинаем* обсуждать технику работы со списками. Как концептуально, так и инструментально понятие списка является основой[157] языка `Mathematica`. Тому, кто не овладел йогой работы со списками и всем богатством встроенных функций для структурных манипуляций над ними, применения функций к спискам и их частям и т.д., `Mathematica` представляется либо просто большим калькулятором наподобие `MathCad`, либо *еще одним* языком программирования. В действительности подлинная мощь этого языка раскрывается *только* в тот момент, когда Вы начинаете систематически использовать списки вместо процедур или рекурсии.

**1. Списки и последовательности.**[158]

С математической точки зрения список и последовательность это примерно одно и то же, но между ними существует чрезвычайно важное синтаксическое различие. Дело в том, что многие функции имеют формат `f[{x,y,z}]`, т.е. работают со *списками* аргументов, а другие — формат `f[x,y,z]`, т.е. работают с *последовательностями* аргументов. С точки зрения языка это различие в высшей степени существенно, так как у функции `f[{x,y,z}]` **один** аргумент, а у функции `f[x,y,z]` — **три**! С другой стороны, задавать последовательность в форме $x, y, z$ без заголовка *синтаксически* неправильно. Поэтому `Sequence` используется как **пустой заголовок**, который самоликвидируется при фактической подстановке последовательности в качестве последовательности аргументов любую другую функцию.

| | | |
|---|---|---|
| `{x,y,z}` | `List[x,y,z]` | список с компонентами $x, y, z$ |
| `x,y,z` | `Sequence[x,y,z]` | последовательность $x, y, z$ |

В Computer Science принято говорить об **элементах** списка, множестве элементов списка etc., там где в действительности идет речь о его **компонентах**, множестве компонент, etc. Так часто будем поступать и мы, хотя с математической точки зрения это совершенно неверно. Дело в том, что с математической точки зрения список `{x,y,z}` является **тупелем**, в

[157] В стандартных пакетах функция `List` используется *значительно* чаще, чем *любая* другая функция, а именно, 24293 раза, почти в два раза чаще, чем основные команды функционального программирования: `Pattern` — 14928 раза, `Blank` — 14746 раза, `Set` — 13501 раза, и в 2–5 раз чаще, чем основные арифметические операции `Times` — 11083 раза, `Power` — 6536 раза, `Plus` — 5544 раза.

[158] Все состоит из атомов, но кофе — из очень хороших атомов. ©Демокрит.

данном случае упорядоченной тройкой $(x,y,z)$, и ни в одной теоретико-множественной интерпретации тупеля $(x,y,z)$ компоненты $x$, $y$ и $z$ не являются его элементами.

**1.1.** Задайте слияние списков в терминах функции `Sequence`.

**Решение.** Нужно убрать скобки вокруг каждого из списков, приписать их друг к другу и снова поставить скобки:

```
myjoin[x_,y_]:=List[Apply[Sequence,x],Apply[Sequence,y]]
```

Список может быть **вложенным** = `nested`, иными словам, его компоненты, компоненты компонент и т.д. могут сами быть списками. Проверка принадлежности элемента списку — или какой-то из его частей, частей его частей и т.д. — производится при помощи одной из следующих команд.

| | |
|---|---|
| `MemberQ[x,y]` | $y$ встречается в списке $x$ |
| `MemberQ[x,y,d]` | $y$ входит в список $x$ на уровне $\le d$ |
| `MemberQ[x,y,{d}]` | $y$ входит в список $x$ на уровне $d$ |
| `FreeQ[x,y]` | $y$ не входит в список $x$ |

Понятие уровня детально обсуждается Модуле 2, и мы вернемся к нему при обсуждении вложенных списков. Пока отметим лишь, что, например, $x$ *не встречается* в списке `{{x}}`, но $x$ *входит* в этот список на уровне 2.

Функцию `MemberQ` не следует путать с функцией `Element`, которая вызывается в формате `Element[x,domain]` и проверяет принадлежность элемента $x$ *домену* `domain`.

**1.2.** Принадлежит ли число `1.` списку `{1,2}`?

Все обычные арифметические операции, числовые функции, многие комбинаторные и теоретико-числовые функции, большинство обычных тестов и некоторые другие внутренние функции имеют атрибут `Listable`. Это значит, что их применение *автоматически* распределяется по спискам одинаковой длины. Точный список таких функций можно узнать посредством

```
Select[Names["*"],MemberQ[Attributes[#],Listable]&]
```

Остальные функции можно *принудительно* распределить по спискам посредством `Thread` или чего-нибудь в таком духе.

**1.3.** Скажите, не используя компьютер, чему равно

`{a,b}+{c,d}`, `{a,b}*{c,d}`, `{a,b}/{c,d}`, `{a,b}^{c,d}`?

А теперь проверьте.

**1.4.** Скажите, не используя компьютер, чему равно

`Prime[{2,3,5}]`, `PrimePi[{2,3,5}]`, `PrimeQ[{2,3,5}]`?

А теперь проверьте.

**1.5.** Скажите, не используя компьютер, чему равно

`Binomial[{5,7},{2,3}]`, `StirlingS2[{5,7},{2,3}]`?

**Ответ.** Ну, $\left\{ {7 \atop 3} \right\}$ в уме, это вряд ли, 301 все-таки.

Длина списка — не считая заголовка `List`! — и количество уровней вложенности — считая заголовок `List`! — находятся при помощи функций `Length` и `Depth`.

| | |
|---|---|
| `Length[z]` | длина списка $z$, *не считая* заголовка |
| `Depth[z]` | глубина списка $z$, *считая* заголовок |

В `Mathematica` все обычные операции с элементами списками выполняются в произвольном месте списка, поэтому ВСЕ КОШЕРНЫЕ ПРОГРАММИСТСКИЕ ДЕЛИКАТЕСЫ, типа различий

- по типу элементов: односортные, многосортные;
- по выделению памяти: фиксированная/переменная длина;
- по принципу связи: `массивы`, `связанные списки`, `дважды связанные списки`, etc.;
- по набору операций: `стеки`, `очереди`, `деки`, etc.;

и тому подобное, НЕ ИМЕЮТ здесь НИКАКОГО ЗНАЧЕНИЯ.

**2. Генерация списков.**

Основной командой генерации векторов, матриц и любых других списков в языке `Mathematica` является команда `Table`. Вызванная с одним итератором команда `Table[f[i],{i,m,n}]` порождает список значений функции $f$ в точках $i = m, m+1, \ldots, n$. Вызванная с двумя итераторами команда

```
Table[f[i,j],{i,k,l},{j,m,n}]
```

порождает **вложенный список** значений функции двух аргументов $f$ в парах $(i, j)$, где $i = k, k+1, \ldots, l$, $j = m, m+1, \ldots, n$. Этот список будет организован как матрица, причем итератор $i$ считается **внешним**, а $j$ — **внутренним**, иными словами, $i$ нумерует строки, а $j$ — позиции внутри строк. Два принципиальных момента здесь таковы:

- ○ МАТРИЦА ТРАКТУЕТСЯ КАК СТРОКА, СОСТАВЛЕННАЯ ИЗ СТРОК,
- ○ ВНУТРЕННИЕ ИТЕРАТОРЫ ПИШУТСЯ ПОСЛЕДНИМИ.

Команда `Array` является просто сокращением команды `Table` со сжатой формой итераторов. Обычно мы вообще не пользуемся командой `Array`, так как полная форма итераторов в `Table` представляется нам более удобной и наглядной.

| | |
|---|---|
| `Apply[List,f[x1,...,xn]]` | тупель $\{x_1, \ldots, x_n\}$ |
| `Table[f[i],{i,1,n}]` | таблица значений функции $f$ |
| `Table[f[i,j],{i,1,m},{j,1,n}]` | ibid., для функции двух аргументов |
| `Array[f,{m}]` | массив значений функции $f$ |

Для порождения чрезвычайно часто встречающихся в программировании и вычислительной математике таблиц *равноотстоящих* = `equally`

spaced чисел служит специальная команда Range, а для порождения таблиц, состоящих из символов ASCII-кода – команда CharacterRange, которая, однако, оперирует с символами не как с математическими, а как с *типографскими* сущностями, а именно, **стрингами** длины 1. Чтобы превратить типографские символы в математические, нужно применить к ним функцию ToExpression. При этом следует иметь в виду, что *далеко* не всякий *стринг* конвертируется в правильно составленное *выражение*!

| | |
|---|---|
| `Range[n]` | тупель $\{1, \ldots, n\}$ |
| `Range[m,n]` | тупель $\{m, \ldots, n\}$ |
| `Range[m,n,d]` | тупель $\{m, m+d, \ldots, m+\lfloor (n-m)/d \rfloor\}$ |
| `CharacterRange["x","y"]` | фрагмент $\{x, \ldots, y\}$ таблицы ASCII |

Обратите внимание, что аргументы функции Range совсем не обязаны быть *целыми* числами — a parte: они вообще не обязаны быть *числами*!

**2.1.** Породите список $\{1000, 999, \ldots, 2, 1\}$.

**Решение.** Первое, что приходит в голову, это `Reverse[Range[1000]]`, но в серьезных вычислениях гораздо лучше `Range[1000,1,-1]`.

**2.2.** Породите список $\{0, \pi/2, \pi, 3\pi/2, \ldots, 10\pi\}$

**Решение.** Конечно, можно так, `Pi/2*Range[0,20]`, но почему не сразу `Range[0,10*Pi,Pi/2]`?

**2.3.** Определите убывающий и возрастающий факториал при помощи команды Range.

**2.4.** Выведите всю видимую[159] часть ASCII-кода.

**Решение.** Полезно знать, что видимая часть ASCII-кода состоит из 94 символов, от 33-го !, до 126-го ~, в десятичной нумерации[160]. Ее можно вывести посредством `CharacterRange["!","~"]`.

Тот же прием работает и для второй половины кодовой таблицы.

**2.5.** Породите русский алфавит. Заодно узнайте, сколько в нем букв.

**Решение.** Да чего уж, `CharacterRange["а","я"]`, 32 буквы. Буква ё выглядит плохо — зато звучит хорошо.

Перечисленные выше способы порождения списков далеко не исчерпывают всех предоставляемых системой возможностей. Вот еще несколько важнейших приемов генерации списков:

• Команды функционального программирования, связанные с композициями и итерациями функций ComposeList, FoldList, NestList, NestWhileList, FixedPointList.

---

[159] По-английски — printable.

[160] Программисты обычно нумеруют символы ASCII-кода не от 0 до 127, а в восьмеричной системе от 000 до 177. В этом случае видимые символы имеют номера от 041-го до 176-го. См., например, Д.Кнут, Все про TEX. — АО RDTEX, Протвино, 1993. Приложение C: Символьные коды.

• Команда `ReplaceList`, порождающая список результатов всех возможных применений правила к какому-то выражению.

• Восстановление списков по их производящим функциям или другим алгебраическим объектам `CoefficientList`, `CoefficientArrays`, `FactorList`, и т.д.

• Формирование разреженного списка `SparseArray`. Обычный список может быть превращен в список правил формирования разреженного списка посредством `ArrayRules`, а разреженный в обычный посредством `Normal`.

Кроме того, конечно, можно строить списки из других списков, в частности, при помощи следующих команд, которые изучаются частично в этой главе.

• Манипуляции с частями списка: извлечения, вычеркивания, вставки, замены, выборки `Part`, `Extract`, `Delete`, `Select`, `Cases`, `DeleteCases`, `ReplacePart`, `Insert`, и многие другие.

• Структурные манипуляции: сортировки, перестановки, выравнивания, разбиения, управление вложенностью: `Sort`, `Reverse`, `RotateLeft`, `RotateRight`, `Flatten`, `Partition`, `Split`, `Transpose` и многие другие.

• Алгебраические и теоретико-множественные операции над несколькими списками: `Join`, `Union`, `Intersection`, `Complement` и другие.

• Команды функционального программирования, `Map`, `MapAll`, `MapAt`, `Thread`, `MapThread`, `Outer` и многие другие.

**3. Биномиальные коэффициенты.**[161]

Выделение первого и последнего элементов списка встречаются в программировании настолько часто, что заслуживают специальных названий.

| | | |
|---|---|---|
| `x[[1]]` | `First[x]` | первая компонента $x$ |
| `x[[Length[x]]]` | `Last[x]` | последняя компонента $x$ |

Основной командой, выделяющей части списка, является команда `Part`, которая может вызываться как в функциональной, так и в операторной форме.

| | | |
|---|---|---|
| `x[[i]]` | `Part[x,i]` | $i$-я компонента $x$ |
| `x[[i,j]]` | `Part[x,i,j]` | компонента $x$ в позиции $(i,j)$ |
| `x[[{i,j}]]` | `Part[x,{i,j}]` | компоненты $x$ в позициях $i$ и $j$ |
| `x[[All,j]]` | `Part[x,All,j]` | $j$-е компоненты всех компонент $x$ |

В некоторых ситуациях используется также команда `Extract`, которую можно рассматривать как вариант `Part`. Более того, для *линейных* списков

[161]— Нужно хотя бы часть чести отстоять...

— Дура! Не часть чести, а честь части.

`Extract[x,i]` и `Part[x,i]` тождественны.

| | |
|---|---|
| `Extract[x,i]` | $i$-я компонента $x$ |
| `Extract[x,{i,j}]` | компонента $x$ в позиции $(i,j)$ |
| `Extract[x,{{i},{j}}]` | компоненты $x$ в позициях $i$ и $j$ |

Главное видимое[162] отличие этих команд – чисто синтаксическое: для определения *вложенных* списков, лежащих на глубоких уровнях, в команду `Part` вводится *последовательность* индексов, а в `Extract` — их *список*. В то же время `Part` трактует *список* индексов как предложение вывести *список* частей верхнего уровня. С другой стороны, `Extract` ожидает в этом месте увидеть индексы частей не на глубине один, а на глубине два!

Породим для наших экспериментов какой-нибудь список глубины 2, скажем, матрицу. Вычисление

```
xxx=Partition[ToExpression[CharacterRange["a","i"]],3]
```

возвращает

```
{{a,b,c},{d,e,f},{g,h,i}}
```

Понятно, что `Extract[xxx,2]`, как и `Part[xxx,2]` вернет `{d,e,f}`. Гораздо интересней, что происходит на уровне 2!

**3.1.** Скажите, не используя компьютер, что даст вычисление каждого из следующих восьми выражений:

```
Part[xxx,2,3],          Extract[xxx,2,3],
Part[xxx,{2,3}],        Extract[xxx,{2,3}],
Part[xxx,{2},{3}],      Extract[xxx,{2},{3}],
Part[xxx,{{2},{3}}],    Extract[xxx,{{2},{3}}],
```

— и даст ли что-нибудь вообще! А теперь проверьте.

Важно помнить, что некоторые функции работы со списками подчиняются тем же соглашениям индексации компонент, что `Part`, в то время как другие — тем же, что `Extract`.

**3.2.** Определите функцию, возвращающую $m$-й с конца элемент списка $x$.

**Решение.** По замыслу `Part[x,-m]` или `Extract[x,{-m}]`, но можно, конечно, и *как-нибудт иначе*, хотя бы `Last[Nest[Most,x,m]]`.

**3.3.** Определите функцию, возвращающую элементы списка $x$ от $l$-го до $m$-го.

**3.4.** Определите функции, возвращающие следующий и предыдущий элементы списка.

**3.5.** Определите функцию, возвращающую элементы списка с нечетными номерами.

**Решение.** Проще всего использовать `Part` и `Range`:

[162] Есть и другие более тонкие отличия, связанные с тем, как `Part` и `Extract` производят эвалюацию.

```
oddpart[x_]:=Part[x,Range[1,Length[x],2]]
```

Впрочем, с таким же успехом можно использовать и `Extract`. Конечно, при этом нужно не забыть каким-то образом ввести дополнительный уровень вложенности. Существует 69 способов сложить песню племен, и все они по-своему хороши. Вот несколько самых простых:

```
oddpart[x_]:=Extract[x,Partition[Range[1,Length[x],2],1]]
oddpart[x_]:=Extract[x,Split[Range[1,Length[x],2]]]
oddpart[x_]:=Extract[x,Map[List,Range[1,Length[x],2]]
oddpart[x_]:=Extract[x,Table[{i},{i,1,Length[x],2}]]
oddpart[x_]:=Extract[x,Position[Table[(-1)^i,
                                  {i,1,Length[x]}],-1]
oddpart[x_]:=Extract[x,NestList[(#+2)&,{1},
                                  Floor[Length[x]/2]-1]
```

Тем не менее, в природных условиях мы всегда используем конструкцию с `Part` — она синтаксически проще и *поэтому* естественнее.

**3.6.** Определите функцию, возвращающую элементы списка с четными номерами.

**3.7.** Породите списки букв латинского алфавита с четными и с нечетными номерами.

**3.8.** Породите список букв латинского алфавита через 7 начиная с `a`, считая, что после `z` снова идет `a`.

**Решение.** Проще всего использовать изученную в Модуле 1 функцию деления с отступом:

```
Part[ToExpression[CharacterRange["a","z"]],
                                  Mod[Range[1,182,7],26,1]]
```

— кстати, почему 182? Вот получающийся список:

```
{a,h,o,v,c,j,q,x,e,l,s,z,g,n,u,b,i,p,w,d,k,r,y,f,m,t}
```

**3.9.** Выделите из списка его элементы в позициях $1, 4, 9, 16, \ldots$

**Решение.** Можно так

```
squarepart[x_]:=Part[x,Table[i^2,{i,1,Floor[Sqrt[Length[x]]]}]]
```

**3.10.** Выберите случайный элемент списка.

**Решение.** Например, при помощи внутреннего генератора случайных чисел

```
randomelem[x_]:=Part[x,Random[Integer,{1,Length[x]}]]
```

**4. Выборки и вычеркивания.**

Один из основных способов построения списков, который мы уже щедро использовали в Главе 7, таков: возьмем большой список и выберем или вычеркнем из него какие-то элементы. **Выборка** и **вычеркивание** могут быть

- либо чисто *позиционными*,
- либо зависеть от свойств самих элементов.

Во втором случае условие на элемент может выражаться

- либо как *критерий*, которому этот элемент должен удовлетворять,
- либо как соответствие этого элемента некоторому *паттерну*.

Вот несколько простейших команд вычеркивания, в которых роль играют не сами элементы, а только их позиция в списке. Две следующие команды настолько часто используются в рекурсивных программах, что заслуживают специальных названий.

| | |
|---|---|
| `Rest[x]` | выбросить первую компоненту $x$ |
| `Most[x]` | выбросить последнюю компоненту $x$ |

Команды `Take` и `Drop` чаще всего используются как обобщения `Rest` и `Most`, когда вычеркивание производится только с начала или с конца списка. Но на самом деле при помощи них можно выбирать/вычеркивать части списка, индексы которых образуют арифметические прогрессии.

| | |
|---|---|
| `Take[x,r]` | взять первые $r$ компонент $x$ |
| `Take[x,-r]` | взять последние $r$ компонент $x$ |
| `Take[x,{r,s}]` | взять компоненты $x$ с $r$-й до $s$-й |
| `Take[x,{r,s,d}]` | взять компоненты $x$ с $r$-й до $s$-й с шагом $d$ |
| `Drop[x,r]` | выбросить первые $r$ компонент $x$ |
| `Drop[x,-r]` | выбросить последние $r$ компонент $x$ |
| `Drop[x,{r,s}]` | выбросить компоненты $x$ с $r$-й до $s$-й |
| `Drop[x,{r,s,d}]` | выбросить компоненты $x$ с $r$-й до $s$-й с шагом $d$ |

**4.1.** Другим манером определите функцию, выбирающую элементы списка с нечетными номерами или вычеркивающую элементы четными номерами.

**Решение.** Ну, конечно, так

```
oddpart[x_]:=Take[x,{1,Length[x],2}]
oddpart[x_]:=Drop[x,{2,Length[x],2}]
```

Видимо, эти решения даже проще, чем решение с `Part`.

Несколько большую свободу дает команда вычеркивания `Delete`, которая использует те же соглашения, что и команда `Extract`.

| | |
|---|---|
| `Delete[x,i]` | вычеркнуть $i$-ю компоненту списка $x$ |
| `Delete[x,{i,j}]` | вычеркнуть компоненту $(i,j)$ списка $x$ |
| `Delete[x,{{i},{j}}]` | вычеркнуть $i$-ю и $j$-ю компоненты списка $x$ |

**4.2.** А теперь еще раз определите функцию, вычеркивающую элементы списка с четными номерами.

**Решение.** Да чего уж там, мы все это уже видели:

```
oddpart[x_]:=Delete[x,Partition[Range[2,Length[x],2],1]]
```

```
oddpart[x_]:=Delete[x,Split[Range[2,Length[x],2]]]
oddpart[x_]:=Delete[x,Map[List,Range[2,Length[x],2]]
oddpart[x_]:=Delete[x,Table[{i},{i,2,Length[x],2}]]
```

и остальные 65 способов, которые мы использовали с `Extract`, но с заменой нечетных индексов на четные.

**4.3.** Определите функцию, вычеркивающую элементы списка с нечетными номерами.

**4.4.** Вычеркните из списка его элементы в позициях $1, 4, 9, 16, \ldots$

Мы не будем здесь детально обсуждать команду `Select`, осуществляющую **выбор по критерию**. Конструкции с этой командой десятки раз использовались в Модуле 1, поэтому ограничимся несколькими примерами.

| | |
|---|---|
| `Select[x,criterion]` | выбрать элементы $x$ по критерию |
| `Select[x,criterion,n]` | выбрать первые $n$ элементов $x$ по критерию |

**4.5.** Выберите из вложенного списка те элементы, которые являются упорядоченными списками.

**Решение.** Например, так: `Select[x,OrderedQ]`.

Отметим лишь следующий важнейший синтаксический момент. Используемый в команде `Select` критерий не обязательно должен быть определен заранее. Он может определяться непосредственно в теле команды в формате анонимной функции `blabla[#]&`.

**4.6.** Задайте функцию, выбирающую те части вложенного списка, в которые не входит $z$.

**Решение.** Проще всего так: `Select[x,FreeQ[#,z]&]`.

Еще один важнейший способ построения подсписков, это выбор и вычеркивание по **соответствию паттерну**.

| | |
|---|---|
| `Cases[x,pattern]` | выбрать элементы, отвечающие паттерну |
| `DeleteCases[x,pattern]` | убрать элементы, отвечающие паттерну |
| `Count[x,pattern]` | число элементов, отвечающих паттерну |
| `Position[x,pattern]` | позиции элементов, отвечающих паттерну |

**4.7.** Задайте функцию, которая выбирает из вложенного списка те части, которые сами являются списками.

**Решение.** Ну, конечно, `Cases[x,_List]`.

Так как количество общих паттернов — `Integer`, `Real`, `Complex`, `List`, ... , — крайне невелико, то, чтобы быть столь же гибким как `Select`, команды, проверяющие соответствие паттерну, должны допускать модификацию паттерна по критерию. Для этого используются две основные конструкции:

- конструкция **паттерн-тест** = `PatternTest` в формате `_pattern?test`.

• паттерн с **условием** = `Condition` в формате `_pattern /; test`,

В первом случае `test` либо должен уже иметь собственное имя, либо быть задан в формате анонимной функции `blabla[#]&`. При этом следует иметь в виду, что приоритет оператора `? = PatternTest` *настолько* велик, что задание критерия в формате анонимной функции необходимо делимитировать круглыми скобками. Иными словами, паттерн, определяющий рациональные числа такие, что $x^2 < 2$, должен выглядеть так `_Rational?(#^2<2&)`.

При ограничении паттерна условием все фигурирующие в условии переменные должны быть явным образом поименованы в паттерне. Иными словами, тот же паттерн, что и выше, задается теперь так `x_Rational /; x^2<2`. Эта конструкция чуть менее эффективна с вычислительной точки зрения, но заметно проще для начинающего.

**4.8.** Задайте функцию, которая выбирает из списка все положительные элементы.

**Решение.** Если с помощью `Cases`, то, например, так

```
positiv[x_]:=Cases[x,_?Positive]
```

**4.9.** Задайте функцию, которая выбирает из списка все элементы меньшие $m$.

**Решение.** Если с помощью `Cases`, то, например, так

```
lowercone[x_,m_]:=Cases[x,y_ /; y<m]
```

**5. Подсписки.**

**5.1.** Напишите программу, порождающую множество всех начальных/конечных отрезков списка.

**Решение.** Это можно сделать миллионом способов. Вот решение в духе функционального программирования:

```
frontend[x_]:=ReplaceList[x,{y__,v___}->{y}]
backend[x_]:=ReplaceList[x,{u___,y__}->{y}]
```

**5.2.** Напишите программу, всевозможными способами разбивающую список на начальный и конечный отрезки.

**Решение.** Например, так

```
partitio[x_]:=ReplaceList[x,{u__,v__}->{{u},{v}}]
```

**Фрагмент** списка состоит из идущих подряд элементов списка. В то же время **подсписок** состоит из элементов исходного списка, расположенных в том же порядке, что и в исходном списке, но, вообще говоря, не обязательно идущих подряд. Иными словами, фрагмент получается из списка вычеркиванием начального и конечного отрезков (может быть пустых). Подсписок получается *произвольными* вычеркиваниями. Например, $\{b, c\}$ является фрагментом списка $\{a, b, c, d\}$, а $\{a, d\}$ является его подсписком, но не фрагментом.

**5.3.** Напишите программу, выводящую список всех фрагментов списка $x$.

**5.4.** Напишите программу, порождающую список всех фрагментов длины $n$ списка $x$.

**Решение.** Можно так:

```
frag[x_,n_]:=Table[Take[x,{i,i+n-1}],{i,1,Length[x]-n+1}]
```

Но в действительности намного эффективнее воспользоваться внутренней функцией `Partition`, которая как раз и служит для этой цели:

```
Partition[x,n,1].
```

**5.5.** Напишите программу, выясняющую, является ли список $y$ фрагментом списка $x$.

**Решение.** Наивный подход состоит в том, чтобы породить список всех фрагментов списка $x$ такой же длины и проверить, содержится ли там $y$:

```
fragmQ[x_,y_]:=MemberQ[Partition[x,Length[y],1],y]
```

**5.6.** Напишите программу, порождающую по списку $x$ список всех списков, получающихся из него вычеркиванием всевозможных фрагментов.

**Решение.** Проще всего так

```
bucato[x_]:=ReplaceList[list,{u___,y__,v___}->{u,v}]
```

Следующие три заметно более сложные задачи, связанные с формированием подсписков, предлагались на экзамене на отличную оценку.

**5.7.** Напишите программу, выясняющую, является ли список $y$ подсписком списка $x$.

Как известно, **палиндромом** называется список, который одинаково читается как слева направо, так и справа налево. Подпалиндромом данного списка называется такой его подсписок, который является палиндромом.

**5.8.** Напишите программу, находящую в данном списке подпалиндром максимальной длины.

**5.9.** Напишите программу, порождающую все последовательности длины $2n$, состоящие из $n$ штук $+1$ и $n$ штук $-1$ такие, что количество $+1$ в каждом начальном отрезке не меньше, чем количество $-1$.

## 6. Основные структурные манипуляции.[163]

Здесь мы учимся использовать простейшие структурные манипуляции над списками, *не связанные с изменением уровней вложенности*, а именно, вставки, замены, слияния, вращения и раздутия.

Вставка элемента в начало или конец списка встречается в рекурсивных программах настолько часто, что заслуживает специального имени.

| | |
|---|---|
| `Append[x,y]` | вставить $y$ в конец списка $x$ |
| `Prepend[x,y]` | вставить $y$ в начало списка $x$ |
| `AppendTo[x,y]` | вставить $y$ в конец списка $x$ |
| `PrependTo[x,y]` | вставить $y$ в начало списка $x$ |

[163] Митек многое умеет. Митек может просчитать теодолитный ход, перевести подпитку котла с регулятора на байпас, раскурочить отбойным молотком мостовую — все это он сделать может. ©Владимир Шинкарев, *Митьки*

Функции `Append` и `Prepend`, осуществляющие вставку элемента в конец или в начало списка, многократно встречались нам в Модуле 1 и их использование не представляет никаких трудностей. Функция `AppendTo[x,y]` является просто сокращенной формой присваивания `x=Append[x,y]`, она требует, чтобы на момент присваивания $x$ уже был списком.

Основной командой **вставки** в `Mathematica` является `Insert`, а основной командой **замены** — `ReplacePart`.

| | |
|---|---|
| `ReplacePart[x,y,r]` | заменить $r$-ю компоненту списка $x$ на $y$ |
| `Insert[x,y,r]` | вставить $y$ на $r$-ю позицию в список $x$ |

Вопреки ожиданиям, команды `ReplacePart` и `Insert` используют те же соглашения относительно адресации частей, что и команды `Extract` и `Delete` — отличающиеся от соглашений, используемых командой `Part`! Вот несколько типичных задач, которые решаются с помощью функций `ReplacePart` и `Insert`.

**6.1.** Определите функцию, которая заменяет элементы списка на нечетных местах на $z$.

**6.2.** Определите функцию, которая заменяет элементы списка на четных местах на $z$.

**6.3.** Определите функцию, которая заменяет первые $m$ элементов списка на $z$.

**6.4.** Определите функцию, которая заменяет первые $m$ элементов списка на $z$.

**6.5.** Определите функцию, которая врисовывает $z$ в начало и конец списка.

**Решение.** Скажем, так `Insert[x,z,{{1},{-1}}]`.

**6.6.** Определите функцию, которая врисовывает $z$ в середину списка четной длины и $z, z$ в середину списка нечетной длины.

Обратите внимание, что адресация, используемая командой `Insert`, всегда относится к *исходному* списку, а не к тем спискам, которые получаются из него после того, как мы уже врисовали какие-то элементы.

**6.7.** Определите функцию, которая сопоставляет списку длины $n$ список длины $2n-1$, в котором между каждыми двумя членами исходного списка вписано $z$.

**Решение.** Проще всего так:

```
funny[x_]:=Insert[x,z,Map[List,Range[2,Length[x]]]]
```

**6.8.** Определите функцию, которая сопоставляет списку длины $n$ список длины $2n$, в котором в самом начале и между каждыми двумя членами исходного списка вписано $z$.

**6.9.** Определите функцию, которая сопоставляет списку длины $n$ список длины $2n$, в котором между каждыми двумя членами исходного списка и в самом конце вписано $z$.

Основной командой **слияния** списков в системе является `Join`. Вычисление `Join[x,y]` возвращает **конкатенацию** списков $x$ и $y$. Иными словами, в терминах следующего параграфа вычисляется `Flatten[List[x,y],1]` — никаких сортировок и исключений в получающемся списке не производится.

| | |
|---|---|
| `Join[x,y]` | слить списки $x$ и $y$ |
| `Union[x,y]` | объединить списки $x$ и $y$ |

**6.10.** Составьте список, в котором вначале идут строчные буквы латинского алфавита от `a` до `z`, а потом прописные буквы от `A` до `Z`.

**Решение.** Естественно, так

```
Join[CharacterRange["a","z"],CharacterRange["A","Z"]]
```

Кстати, почему в данном случае мы не стали применять `ToExpression`?

Следующую задачу можно, конечно, решить при помощи `Insert` или `Replace`, но проще использовать `Join`.

**6.11.** Определите функцию, которая разделяет список длины $n$ на начальный отрезок длины $m$ и конечный отрезок длины $n-m$ и переставляет их между собой.

**6.12.** Определите функцию, которая сопоставляет списку длины $n$ список длины $2n$, состоящий из элементов исходного списка, написанных сначала в прямом, а потом в обратном порядке.

**6.13.** Определите функцию, которая сопоставляет списку длины $n$ список длины $2n-1$, в котором между каждыми двумя членами исходного списка вписана их сумма.

**Решение.** Отбросим из списка последний элемент, образуем требуемый список для этого более короткого списка, а потом вольем недостающий кусочек. Для разнообразия оформим это, не применяя `Last` и `Most` к исходному списку:

```
stern[{x_}]:={x}
stern[{x__,y_}]:=stern[{x,y}]
                    =Join[stern[{x}],{Last[stern[{x}]]+y,y}]
```

**6.14.** Определите функцию, которая сопоставляет спискам $x=(x_1,\ldots,x_n)$ и $y=(y_1,\ldots,y_n)$ длины $n$ список длины $2n$

$$x\sharp y=(x_1,y_1,\ldots,x_n,y_n),$$

называемый их **перемешиванием**.

**6.15.** Задайте функцию, которая разделяет список длины $2n$ на начало и конец, а потом перемешивает их.

Основная команда **вращения** это `RotateRight` — или, что то же самое! — `RotateLeft`.

| | |
|---|---|
| `RotateLeft[x]` | циклически переставить элементы списка $x$ влево |
| `RotateLeft[x,m]` | циклически переставить элементы списка $x$ влево |
| `RotateRight[x]` | циклически переставить элементы списка $x$ вправо |
| `RotateRight[x,m]` | циклически переставить элементы списка $x$ вправо |

`RotateRight[x,m]` задает вращение списка $x$ на $m$ позиций вправо, а `RotateLeft[x,m]` — на $m$ позиций влево. Таким образом, `RotateRight[x]` это просто `RotateRight[x,1]`, а `RotateRight[x,m]` это то же самое, что `Nest[RotateRight,x,m]`. Различие между командами `RotateRight` и `RotateLeft` чисто психологическое.

**6.16.** Несколькими способами выразите `RotateRight` через `RotateLeft`.

**Решение.** Проще всего, конечно, так `RotateLeft[x,-1]`, но можно так `RotateLeft[x,Length[x]-1]]`, или так

```
Reverse[RotateLeft[Reverse[x]]].
```

**6.17.** Несколькими способами выразите `RotateRight[x,m]` через `RotateLeft`.

**6.18.** Список $x$ длины $n$ мыслится как соединение начального подсписка длины $m$ и конечного подсписка длины $n-m$. Задайте вращение, переставляющее начало и конец.

**Решение.** Проще всего `RotateLeft[x,m]` или `RotateRight[x,n-m]`. Кроме приведенных выше, у этой задачи есть много других решений. Например, можно использовать тот факт, что вращение есть произведение двух отражений:

```
Reverse[Join[Reverse[Take[x,m]],Reverse[Take[x,m-n]]]]
```

Основные команды **раздутия** — или, как говорят в линейной алгебре, **окаймления** — это `PadLeft` и `PadRight`

| | |
|---|---|
| `PadLeft[x,n,{y,z}]` | раздуть список $x$ влево посредством $y, z$ |
| `PadRight[x,n,{y,z}]` | раздуть список $x$ вправо посредством $y, z$ |

Вычисление `PadRight[x,n,{y,z,...}]` формирует список длины $n$, получающийся из списка $x$ дорисовыванием циклически повторяющихся элементов $y, z, \ldots$. Важнейший нюанс состоит в том, что элементы $y, z, \ldots$ начинают расходоваться не с позиции `Length[x]+1`, а с позиции 1, иными словами, для списка $x$ длины $m$ первые $m$ элементов $y, z, \ldots$ остаются невидимыми. Поэтому эта команда удобнее всего в одном из следующих случаев.

• Раздутие посредством одного элемента, в формате `PadRight[x,n,y]`. По умолчанию $y = 0$, так что `PadRight[x,n]` дорисовывает к списку $x$ длины $m$ справа $n-m$ нулей.

• Раздутие пустого списка `PadRight[{},n,{y,z,...}]`, иными словами, с самого начала формируемого списка циклически повторяются элементы списка $(y, z, \ldots)$

**6.19.** Составьте список $(1, -1, 1, -1, \ldots)$ длины $2n$.

**Решение.** Мы это уже делали многими разными способами, используя `Table`, `Join` и `Flatten`, но теперь проще всего так

```
PadRight[{},2*n,{1,-1}].
```

**6.20.** Составьте список $(a, b, c, a, b, c, \dots)$ длины $n$.

**6.21.** Задайте число 1.73 ... 3, где количество троек равно $n$.

**Решение.** Скажем, так

```
N[FromDigits[{PadRight[{1,7},n+2,3],1}]]
```

**6.22.** Сформируйте список $(a, b, x, y, z, x, y, z, \dots)$, где группа $(x, y, z)$ повторена $n$ раз.

**Решение.** Как мы уже предупреждали, вычисление

```
PadRight[{a,b},3*n+2,{x,y,z}]
```

не даст правильного результата. Например, вычисление

```
PadRight[{a,b},11,{x,y,z}]
```

дает `{a,b,z,x,y,z,x,y,z,x,y}`, а вовсе не `{a,b,x,y,z,x,y,z,x,y,z}`, как хотелось бы. Нужный результат можно, конечно, получить посредством

```
PadRight[{a,b},3*n+2,RotateRight[{x,y,z},2]]
```

Однако, чтобы не вычислять в каждом случае нужный сдвиг, гораздо удобнее обращаться к `PadRight` в одном из следующих форматов:

```
Join[{a,b},PadRight[{},3*n,{x,y,z}]]
```

```
Flatten[PadRight[{a,b},n+2,{{x,y,z}}],1]
```

Снова различие между `PadRight` и `PadLeft` не мериторическое, а *чисто* психологическое. Ясно, что раздутие влево `PadLeft[x,n,{y,z,...}]` это *в точности* раздутие вправо, но считая с конца и в отрицательном направлении, `PadRight[x,-n,{y,z,...}]`.

## 7. Вложенные списки.[164]

Список, элементы которого сами являются списками, называется **вложенным списком** = `nested list`. В языке `Mathematica` вложенные списки являются основным инструментом для моделирования математических структур. Матрицы, тензоры, деревья, отображения, отношения, графы, и многое другое — все это моделируется вложенными списками.

В частности, мы уже много раз пользовались тем фактом, что матрица в языке `Mathematica` записывается как вложенный список, составленный из *строк* этой матрицы. Скажем, `{{a,b,c},{d,e,f},{g,h,i}}` изображает матрицу

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}.$$

В этом параграфе мы изучим основные приемы работы со списками, связанные с *изменением уровней вложенности*, а именно,

- команды **выравнивания**, которые убирают уровни вложенности;
- команды **разбиения**, которые добавляют уровни вложенности;

[164] What bird has done yesterday, man may do next year, be it fly, be it moult, be it hatch, be it nest. ©James Joyce, *Finnegan's wake*

• команды **транспонирования**, которые переставляют уровни вложенности.

В частности, в применении к матрицам, выравнивание превращает матрицы в строки, разбиение превращает строки в матрицы, а транспонирование восстанавливает симметрию между строками и столбцами.

Основные команды выравнивания в системе это `Flatten` и `FlattenAt`.

| | |
|---|---|
| `Flatten[x]` | выровнять список $x$ |
| `Flatten[x,m]` | выровнять список $x$ до уровня $m$ |
| `FlattenAt[x,i]` | выровнять только $i$-ю позицию |

Команда `Flatten[x]` производит *полное* выравнивание списка, убирая все внутренние скобки. Иными словами, эта команда эффективно превращает *любой* вложенный список в линейный. Обычно, при этом исчезает большая часть нужной структуры, поэтому гораздо чаще применяются команды частичного выравнивания `Flatten[x,m]`, убирающие только $m$ *верхних* уровней вложенности. Такая необходимость связана с тем, что как команды формирования списков, так и многие команды применения функций к спискам, в частности, `Outer`, создают *лишние* уровни вложенности. С другой стороны, мощнейший метод работы со списками состоит в том, чтобы искусственно создать лишние уровни вложенности. Особенно часто используется команда `Flatten[list,1]`, убирающая *один* лишний вложенный уровень в `list`, а именно, *верхний*.

Гораздо избирательнее действует команда `FlattenAt`, которая убирает только один уровень вложенности и только у $i$-й части списка.

**7.1.** Задайте функцию, которая врисовывает после $i$-го элемента списка $x$ элементы $u, v, w$

**Решение.** Если с использованием `FlattenAt`, то так

```
FlattenAt[Insert[x,{u,v,w},i+1],i+1]
```

**7.2.** Выразите функцию `Flatten` в терминах `FlattenAt`.

**7.3.** Выразите функцию `FlattenAt` в терминах `Flatten`.

**Указание.** Используйте `MapAt`.

Основные команды разбиения это `Partition` и `Split`. Функция `Partition` может использоваться в связи с задачами линейной алгебры (генерация матриц и списков матриц). Здесь мы об этом лишь вскользь упоминаем, но не будем обсуждать весьма деликатные детали, связанные с ее параметрами.

| | |
|---|---|
| `Partition[x,m]` | разбить список $x$ на списки длины $m$ |
| `Partition[x,m,d]` | разбить список $x$ на списки длины $m$ со сдвигом $d$ |
| `Split[x]` | разбить список $x$ на списки одинаковых элементов |
| `Split[x,test]` | разбить список $x$ по критерию |

**7.4.** Задайте функцию, порождающую по списку список пар его соседних элементов.

**Решение.** Ну конечно, просто `Partition[x,2,1]`.

**7.5.** То же, считая, что список циклический, ными словами, после последнего элемента снова идет первый.

**Решение.** На самом деле, так: `Partition[x,2,1,{1,1}]`. Впрочем, это как раз одна из тех деталей, которые мы договорились не обсуждать. Поэтому воспользуйтесь `Append`.

**7.6.** Породите список

```
{{{{a,b},{c,d}},{{e,f},{g,h}}},{{{i,j},{k,l}},{{m,n},{o,p}}}}
```

**Решение.** Например, так

```
Nest[Partition[#,2]&,ToExpression[CharacterRange["a","p"]],3]
```

**7.7.** А теперь уберите в списке из предыдущей задачи *нижний* уровень вложенности.

**Решение.** Мы знаем, что команда `Flatten[x,1]` убирает *верхний* уровень вложенности, иными словами, превращает этот список в список

```
{{{a,b},{c,d}},{{e,f},{g,h}},{{i,j},{k,l}},{{m,n},{o,p}}}
```

Напрашивающаяся идея состоит в том, чтобы *переставить* верхний и нижний уровень, убрать верхний уровень и снова переставить верхний и нижний уровень. Реализовать эту идею можно, но это совсем не так просто, как кажется — *To tell between* `Murphy's` *and* `Guinness` *is sure not a task for beguinness*. Даже при правильной перестановке уровней элементы внутри самых глубоких списков окажутся переставленными по сравнению с тем, что получается при применении выравнивания к нижнему уровню!

Правильное решение состоит в том, чтобы *селективно* применить `Flatten` при помощи `Map`, на самом нижнем уровне, примерно так

```
Map[Flatten,x,{-3}]
```

Откуда здесь берется $-3$? Ну, уровень $-1$ — это атомы, которые не имеют дальнейшей структуры, и применять к ним `Flatten` нельзя. Уровень $-2$ — это линейные списки, в которых нет внутренних скобок, и применять к ним `Flatten` бессмысленно. Применение же этой команды к списку из предыдущей задачи возвращает

```
{{{a,b,c,d},{e,f,g,h}},{{i,j,k,l},{m,n,o,p}}},
```

как и задумывалось.

К спискам, содержащим большое количество повторений, часто применяется метод сжатия, называемый `RunLengthEncode`, когда список заменяется на список пар, состоящих из элемента и количества ее повторений в серии.

**7.8.** Реализуйте функцию `RunLengthEncode`.

**Решение.** Проще всего, конечно, с помощтю функции `Split`:

```
RunLengthEncode[x_]:=Map[{First[#],Length[#]}&,Split[x]]
```

Чаще всего встречающийся элемент списка называется **модой**.

**7.9.** Задайте функцию, сопоставляющую списку список его мод.

Однако, в действительности, команда `Split` служит не для того, чтобы разбивать список на одинаковые элементы, а для того, чтобы разбивать его на серии *эквивалентных* элементов. При этом эквивалентность может либо вызываться по имени — по умолчанию это `SameQ` — либо задаваться непосредственно в теле функции `Split` в формате анонимной функции от двух переменных `blabla[#1,#2]&`.

**7.10.** Разбейте список натуральных чисел на последовательные фрагменты, состоящие из чисел одинаковой четности.

**Решение.** Проще всего так

```
Split[x,Mod[#1-#2,2]==0&]
```

Перестановка уровней списка осуществляется командой `Transpose`, которую не следует путать с командой `Reverse`, переставляющей начало и конец *линейного* списка.

| | |
|---|---|
| `Reverse[x]` | инвертировать список $x$ |
| `Transpose[x]` | транспонировать список $x$ |
| `Transpose[x,perm]` | переставить уровни вложенности списка $x$ |

По умолчанию команда `Transpose` переставляет *два верхних уровня* вложенности списков. Таким образом, например, команда `Transpose`, примененная к *списку* матриц будет переставлять строки этих матриц между собой, а вовсе не транспонировать сами эти матрицы. В Модуле 1 мы реализовали команду, которая транспонирует каждую из матриц списка, при помощи `Map`.

**7.11.** Задайте функцию, которая транспонирует каждую из матриц списка непосредственно при помощи `Transpose`.

**7.12.** Задайте функцию, которая переставляет верхний уровень вложенности списка с нижним, оставляя на месте все остальные.

**7.13.** Задайте функцию, которая транспонирует два нижних уровня вложенности списка.

Чрезвычайно эффективная стратегия работы со списками — *основная* стратегия, которой придерживаются опытные пользователи — состоит в следующем:

- создать дополнительные уровни вложенности,
- выполнить операции над образовавшимися короткими списками,
- снова убрать лишние уровни вложенности.

**7.14.** Определите команду, переставляющую четные и нечетные позиции списка.

**Решение.** Конечно, это можно исполнить совсем тупо, скажем, так

```
interchange[x_]:=Table[If[OddQ[i],x[[i+1]],x[[i-1]]],
                                                {i,1,Length[x]}]
```

Однако, понимающий человек задаст ту же функцию так:

```
interchange[x_]:=Flatten[Map[Reverse,Partition[x,2]],1]
```

Кстати, задание первой функции содержит серьезную ошибку, в чем легко убедиться попытавшись вычислить

```
interchange[ToExpression[CharacterRange["a","z"]]]
```

**8. Деревья.**[165,166,167]

Важнейшей нелинейной структурой при работе с данными, выражениями и алгоритмами являются деревья. **Укорененное дерево** = `rooted tree` представляет собой следующую рекуррентно определяемую математическую структуру:

• *конечное* множество **вершин** $X$,

• в случае $X \neq \emptyset$, *выделенную точку* $x \in X$, называемую **корнем** дерева $X$.

• разбиение всех остальных вершин на $m$ *попарно* непересекающихся множеств $X_1, \ldots, X_m$, каждое из которых в свою очередь снабжено структурой укорененного дерева.

Корни $x_1, \ldots, x_m$ укорененных деревьев $X_1, \ldots, X_m$ называются **детьми** корня $x$, по отношению к ним $x$ является **родителем**. По отношению друг к другу дети одного и того же родителя называются **сиблингами**. Эта терминология не является общепринятой, часто сиблинги называются также братьями или сестрами, etc., как указано в следующей таблице:

| | | | |
|---|---|---|---|
| parent | mother | father | родитель |
| child | daughter | son | ребенок |
| sibling | sister | brother | сиблинг |
| ancestor | | | предок |
| descendant | | | потомок |

В дальнейшем для краткости укорененные деревья называются просто **деревьями**. Рассмотрим следующие способы задания деревьев посредством вложенных списков:

• **Список с отступами**: за каждой вершиной указывается список ее поддеревьев.

• Для каждой вершины указывается ее родитель.

• Для каждой вершины указывается список ее предков.

---

[165] И произрастил Господь Бог из земли всякое дерево, приятное на вид и хорошее для пищи, и дерево жизни посреди рая, и дерево познания добра и зла. *Бытие*

[166] Устроил себе сады и рощи, и насадил в них всякие плодовитые дерева. *Екклесиаст*

[167] Как известно, деревья появились на третий день сотворения мира и на протяжении веков очень широко применялись. ©Дональд Кнут *Искусство программирования*

• Для каждой вершины указывается список ее детей.

• Для каждой вершины указывается список ее потомков.

• Для каждой вершины указываются ее старший ребенок и непосредственно следующий сиблинг.

• Для каждой вершины указываются ее младший ребенок и непосредственно предшествующий сиблинг.

В следующих задачах предполагается решение для каждого из описанных выше представлений деревьев.

**8.1.** Задайте функцию, указывающую корень дерева.

**8.2.** Задайте функцию, указывающую для каждой вершины ее родителя.

**8.3.** Задайте функцию, указывающую для каждой вершины всех ее предков.

**8.4.** Задайте функцию, указывающую для каждой вершины ее старшего ребенка.

**8.5.** Задайте функцию, указывающую для каждой вершины ее младшего ребенка.

**8.6.** Задайте функцию, указывающую для каждой вершины список ее детей.

**8.7.** Задайте функцию, указывающую для каждой вершины непосредственно предшествующего ей сиблинга.

**8.8.** Задайте функцию, указывающую для каждой вершины непосредственно следующего за ней сиблинга.

**8.9.** Задайте функцию, указывающую для каждой вершины список ее младших сиблингов.

**8.10.** Задайте функцию, указывающую для каждой вершины список ее старших сиблингов.

**8.11.** Задайте функцию, указывающую для каждой вершины список всех ее сиблингов.

**8.12.** Задайте функцию, указывающую для каждой вершины список всех ее потомков.

**8.13.** Задайте функцию, возвращающую $m$-ю генерацию (= уровень) дерева.

**8.14.** Задайте функцию, возвращающую множество листов дерева.

Теперь, вооружившись функциями, определенными в этих задачах, вы можете справиться со следующей общей задачей.

**8.15.** Задайте функции, преобразующие каждое из перечисленных выше представлений, во каждое из остальных.

## 9. Порядок и сортировка.

A crusader's wife slipped from the garrison
And had an affair with a Saracen.
She was not oversexed,
Or jealous or vexed,
She just wanted to make a `comparison`.[168]

**Задача сортировки** в простейшей постановке состоит в следующем. Имеется список

`{{x1,y1},{x2,y2},...,{xn,yn}}`

состоящий из **записей** $x_1, \ldots, x_n$ и **ключей** $y_1, \ldots, y_n$. Требуется расположить этот список в порядке возрастания ключей. В дальнейшем мы обычно делаем два упрощающих предположения:

• записи совпадают с ключами, иными словами, происходит сортировка списка `{x1,x2,...,xn}`;

• среди ключей нет повторяющихся, иными словами, $x_i \neq x_j$ при $i \neq j$.

Оба эти предположения являются *чисто техническими*, никак не влияют на сложность задачи сортировки и введены исключительно для упрощения обозначений. Все излагаемые далее алгоритмы работают и без этих предположений. Более того, некоторые из них являются **устойчивыми**, в том смысле, что они не меняют порядка записей с одинаковым ключом, по отношению к исходному списку.

В `Mathematica` имеются внутренняя команда, осуществляющая сортировку, `Sort`. Команда `OrderedQ[x]` является сокращением для `Sort[x]==x`.

| | |
|---|---|
| `Order[x,y]` | сравнение $x$ и $y$ |
| `Sort[x]` | сортировка списка $x$ |
| `Sort[x,crit]` | сортировка списка $x$ по критерию `crit` |
| `OrderedQ[x]` | тест упорядоченности $x$ |

Сортировка осуществляется в соответствии с **внутренним порядком**, заданным на множестве всех выражений, подробнее по этому поводу см. Модуль 2 Основы синтаксиса . Функция `Order[x,y]` возвращает $+1$, если $x$ предшествует $y$ относительно внутреннего порядка, $-1$, если $x$ следует за $y$, и 0, если они равны. Вот что достаточно знать о внутреннем порядке начинающему:

• числа предшествуют буквам;

• для вещественных чисел используется обычный порядок;

• комплексные числа сортируются по вещественной части, а при равенстве вещественных частей — по *абсолютной величине* мнимой части (подробнее см. Модуль 1);

---

[168] В сборнике *Лимерики по-русски* этот лимерик переведен так: Чернобровая дева из Азии/ Отдалась молодцу ашкеназии./ Ей не то, чтобы мало/ От своих попадало,/ Но хотелось ей разнообразии. — К сожалению, при этом потеряно ключевое слово `comparison`.

• для символов и выражений используется обычный алфавитный порядок;

• строчные буквы предшествуют прописным;

• короткие выражения предшествуют длинным.

В случае, когда мы хотим осуществить сортировку в соответствии с порядком, отличающимся от внутреннего порядка, применяется **сортировка по критерию** `Sort[x,crit]`. Например, применение `Sort[x,Greater]` или, что то же самое, `Sort[x,#>#2&]`, отсортирует список чисел не в порядке возрастания, а в порядке убывания.

**9.1.** Отсортируйте список натуральных чисел по числу делителей.

**Решение.** Напрашивающееся решение

```
Sort[Range[10],Length[Divisors[#1]]<Length[Divisors[#2]]&]
```

приводит к достаточно странному результату. Правильное решение таково:

```
Reverse[Sort[Range[20],
            Length[Divisors[#1]]>Length[Divisors[#2]]&]]
```

**9.2.** Отсортируйте список пар по второму элементу.

**Решение.** В зависимости от того, что мы под этим понимаем. Если мы хотим отсортировать пары лексикографически, но с обратным порядком индексов, то так

```
newsort[x_]:=Sort[x,OrderedQ[{Reverse[#1],Reverse[#2]}]&]
```

Хотя можно применить трюк с двойным применением `Reverse` — в части, посвященной перестановкам, мы будем *постоянно* пользоваться этим трюком, как с `Reverse`, так и с `Transpose`:

```
newsort[x_]:=Map[Reverse,Sort[Map[Reverse,x]]]
```

Если же мы хотим просто переставить пары в порядке возрастания второго элемента, вообще не обращая внимания на первый элемент, то так:

```
newsort[x_]:=Sort[x,OrderedQ[{Last[#1],Last[#2]}]&]
```

**9.3.** Отсортируйте элементы вложенного списка по возрастанию первых $m$ элементов

**Решение.** Если мы уверены, что все элементы действительно длиннее, чем $m$, то так:

```
inisort[x_,m_]:=Sort[x,OrderedQ[{Take[#1,m],Take[#2,m]}]&]
```

В противном случае нужно вначале раздуть более короткие элементы вправо посредством `-Infinity`.

По умолчанию порядок на вложенных списках таков: вначале списки сортируются по длине, а потом списки *данной длины* сортируются лексикографически. Иными словами, применяется не чисто лексикографический порядок `Lex`, а `LengthLex`.

**9.4.** Задайте функцию, сортирующую вложенные списки чисто лексикографически, не обращая внимания на длину.

**9.5.** Упорядочите числа от 1 до $n$ лексикографически.

**Указание.** Тот, кто читал Модуль 1, знает, что для этого достаточно превратить число в список цифр.

По умолчанию `Sort` вообще не обращает внимания на *глубину* вложенного списка, иными словами, список {{`x,y,z`}} имеющий длину 1, предшествует списку {`x,y`}, имеющему длину 2.

**9.6.** Задайте функцию, сортирующую вложенные списки по глубине, а внутри данной глубины в обычном порядке.

**9.7.** Задайте функцию, сортирующую вложенные списки по глубине, а внутри данной глубины чисто лексикографически.

Функция порядковой статистики `Ordering` является дополнительной к `Sort`. Вызванная без параметров, она возвращает позиции элементов списка `Sort[x]` в исходном списке $x$. Для списка $x$, являющегося перестановкой `Range[n]`, значение `Ordering[x]` это просто **обратная перестановка**. В случае списков с повторяющимися элементами функция `Ordering` возвращает позицию первого такого элемента. Как и функция `Sort`, она может указывать позиции при сортировке по критерию, но первым параметром для нее является отсечка по количеству элементов. Поэтому при задании критерия как *второго* параметра необходимо явно указывать дефолтное значение первого параметра, `All`.

| | |
|---|---|
| `Ordering[x]` | положение элементов `Sort[x]` в $x$ |
| `Ordering[x,n]` | положение первых $n$ элементов `Sort[x]` в $x$ |
| `Ordering[x,All,crit]` | положение элементов `Sort[x,crit]` в $x$ |

Вот как, примерно, используется функция `Ordering`.

**9.8.** Найдите положение $m$-го по порядку элемента в списке $x$.

**Решение.** Для этого нужно вспомнить, как выделяются части и уровни выражений. В данном случае, `Ordering[x,{m}]`.

**9.9.** Задайте функцию, которая сортирует список в соответствии со значениями функции $f$ на элементах этого списка.

**Решение.** Вот решение из книги Вольфрама

```
sortBy[list_,f_]:=list[[Ordering[Map[f,list]]]]
```

## 10. Простая сортировка.[169]

Примерно 400 страниц книги Кнута[170] посвящено детальному изложению *нескольких десятков* алгоритмов сортировки, с доказательством их корректности, вычислением среднего и худшего времени их работы, деталями практической реализации и пр. Ясно, что мы не можем конкурировать

[169] Always glad to share my ignorance — I've got plenty.

[170] Кнут Д. Искусство программирования. Том 3. Сортировка и поиск М. – СПб – Киев: Вильямс, 2000. 822 с.

с Кнутом, да это и не нужно, так как большинство наиболее тонких алгоритмов **внешней сортировки**, разработанных в то время, когда *списки ключей* не помещались оперативную память, имеют сегодня чисто историческое и рекреативное значение. Если не работают обычные алгоритмы внутренней сортировки, рецепт здесь один — BUY A LARGER AND FASTER COMPUTER.

В этом и следующем двух пунктах мы опишем и сравним некоторые простейшие алгоритмы сортировки. В действительности количество шагов алгоритма сортировки мало что говорит о его качестве. Во многих случаях нужно минимизировать другие параметры.

• **Количество сравнений**. Ключи не обязаны быть числами, если ключи сами являются словами, строками, стрингами, их сравнение может быть в высшей степени нетривиальным.

• **Количество перестановок**. Размер записей может быть очень велик. В этих случаях нужно минимизировать количество перемещений.

Сортировка, в которой на каждом шаге происходит сравнение и перестановка двух элементов, называется **простой сортировкой**. Вот три популярных общих метода простой сортировки, выполняющие сортировку *в среднем* за время $O(n^2)$.

• **Простая обменная сортировка = сортировка перестановкой**: на каждом шаге переставляются два элемента $x$ и $y$ образующие **инверсию**, т.е. такие, что $x$ предшествует $y$, но $x > y$.

• **Простая сортировка выбором**: выбирается *минимальный* среди неотсортированных элементов, и ставится на *последнее* место среди отсортированных.

• **Простая сортировка вставкой**: берется *первый* среди неотсортированных элементов и вставляется на *нужное* место среди отсортированных.

Существует много различных вариантов организации вычислений. Например, при обменной сортировке можно начинать либо с самого левого, либо с самого правого элемента, который меньше предыдущего, и продолжать двигать его влево, пока при этом происходит уменьшение количества инверсий. После этого можно либо возвратиться к началу или концу списка, либо — лучше!! — продолжать двигаться в том же направлении по списку в поиске еще одного элемента, который меньше предыдущего. С другой стороны, можно начать с самого левого или самого правого элемента, который больше следующего, и продолжать двигать его вправо, пока при этом происходит уменьшение количества инверсий. При применении первой процедуры из списка {4,3,5,1,2} за один проход получится {1,4,3,5,2}. При применении второй первой тонет 4, в результате чего получается {3,4,5,1,2}, после чего начинает тонуть 5 и, окончательно, за один полный проход мы придем к {3,4,1,2,5}. За еще один полный проход получится {3,1,2,4,5} и т.д. Для наглядности эту ситуацию описывают так, как будто список расположен вертикально и маленькие = легкие элементы поднимаются, а большие = тяжелые — опускаются.

**10.1.** Реализуйте простую обменную сортировку полнопроходным **методом пузырька**: на каждом шаге самый нижний элемент, меньший чем предыдущий, начинает всплывать, и всплывает, пока не наталкивается на меньший элемент, после этого начинает всплывать следующий элемент, меньший, чем предыдущий и т.д. Дойдя до начала списка, мы возвращаемся к его концу и делаем следующий проход.

**Решение.** Вот один шаг рекурсивного алгоритма, в стилистике сурового минимализма:

```
bubble[{}]:={}; bubble[{x_}]:={x};
bubble[x_]:=If[OrderedQ[Last[x],Last[Most[x]]],
           Append[bubble[Append[Most[Most[x]],Last[x]]],
                                              Last[Most[x]]],
           Append[bubble[Most[x]],Last[x]]]
```

Вот тот же шаг в процедурном жанре:

```
bubble[x_]:=Block[{i,y=x},For[i=Length[x],i>1,i--,
           If[y[[i]]<y[[i-1]],
                    {y[[i-1]],y[[i]]}={y[[i]],y[[i-1]]}]];
                              Return[y]]
```

После этого остается лишь положить

```
bubblesort[x_]:=FixedPoint[bubble,x]
```

**10.2.** Реализуйте простую обменную сортировку методом пузырька в следующем варианте: на каждом шаге самый нижний элемент, меньший чем предыдущий, начинает всплывать, и всплывает, пока не наталкивается на меньший элемент, после чего мы снова возвращаемся к концу списка.

**10.3.** Реализуйте простую обменную сортировку полнопроходным методом пузырька в варианте, когда всплытие пузырьков начинается сверху, а не снизу.

**10.4.** Если метод пузырька сортирует список длины 100 за 0.2 секунды, то сколько времени ему понадобится, чтобы отсортировать список длины 500? А теперь проведите эксперимент.

**10.5.** Реализуйте простую обменную сортировку полнопроходным **методом грузика**: на каждом шаге самый верхний элемент, больший, чем следующий, начинает тонуть, и тонет, пока не наталкивается на больший элемент. После этого начинает тонуть следующий элемент и т.д. Дойдя до конца списка, мы возвращаемся к его началу и делаем следующий проход.

**Решение.** Если без фантазии, то

`bubble`$\mapsto$`plummet`, `Last`$\mapsto$`First`, `Most`$\mapsto$`Rest`, `Append`$\mapsto$`Prepend`.

И, конечно, не забудьте заменить неравенство в тесте на противоположное — или переставить исходы в `If`.

**10.6.** Реализуйте простую обменную сортировку методом грузика в следующем варианте: на каждом шаге самый верхний элемент, больший, чем

следующий, начинает тонуть, и тонет, пока не наталкивается на больший элемент, после чего мы снова возвращаемся к началу списка.

**10.7.** Реализуйте простую обменную сортировку **методом шейкера**: на нечетных шагах всплывает пузырек, на четных — тонет грузик. Во сколько раз — и почему — этот метод лучше двух предыдущих?

**10.8.** Реализуйте алгоритм **простой случайной сортировки**: два *случайно* выбранных элемента $x_i$ и $x_j$, $i < j$, списка $x$ переставляются, если $x_i > x_j$.

**10.9.** Экспериментально оцените, сколько раз следует применить случайную сортировку, чтобы список длины $n$ оказался отсортированным с вероятностью 90%.

**10.10.** Реализуйте простую сортировку выбором для списков без повторений.

**Решение.** Если без рекурсии, то, например, так:

```
selectsort[x_]:=Block[{y={},z=x,w},
            Nest[(w=Min[#];y=Append[y,w];z=DeleteCases[z,w])&,
                                    z,Length[x]-1];Return[y]]
```

Здесь $y$ обозначает отсортированную, а $z$ — неотсортированную часть $x$. Зачем мы ввели локальную переменную $w$? Кстати, что произойдет здесь, если список $x$ содержит повторения?

**10.11.** Реализуйте рекурсивный алгоритм простой сортировки с выбором максимума вместо минимума.

**10.12.** Задайте функцию, которая за время $O(\log(n))$ ставит элемент $y$ на нужное место в *отсортированном* списке $x$ длины $n$.

**Решение.** Например, обычным методом деления пополам:

```
locate[{},y_]:={y};
locate[{x_},y_]:=If[OrderedQ[{x,y}],{x,y},{y,x}]
locate[x_,y_]:=Block[{n=Length[x],l=Floor[n/2]},
                  If[OrderedQ[{y,x[[l]]}],
                        Join[locate[Take[x,l],y],Take[x,l-n]],
                        Join[Take[x,l],locate[Take[x,l-n],y]]]]
```

Первые две строки задают начальное условие, а остальные строки — рекуррентную процедуру, которая вызывает функцию `locate` либо для первой, либо для второй половиеы списка. Переменные $l$ и $n$ задекларированы в качестве локальных переменных не столько для того, чтобы вычислять `Length` и `Ceiling` один раз, сколько для того, чтобы сократить текст. При последовательном сравнении $y$ с элементами списка $x$ стартуя с его начала или конца в худшем случае понадобится $n$ сравнений, а вовсе не $\log_2(n)$ + константа.

**10.13.** Реализуйте простую сортировку вставкой для списков без повторений.

**11. Быстрая сортировка.**[171]

Ни один из методов простой сортировки не может дать *в среднем* время лучше, чем $O(n^2)$. Чтобы получить лучшее время, необходимо работать не с индивидуальными элементами, а с подсписками. Вот два простейших метода коллективной сортировки, достаточные для большинства практических целей.

• **сортировка с убывающим смещением**: **алгоритм Шелла** и его варианты *в худшем* случае дают время $O(n^{1.5})$.

• **сортировка разделением**: алгоритм Хоара = быстрая сортировка = `quicksort` дает время $O(n\ln(n))$ *в среднем*, а его версии, использующие нелинейную структуру, такие, как пирамидальная сортировка = сортировка кучками = `heapsort` — даже *в худшем* случае.

При небольших $n$, скажем $n \le 1024$, удобно применять алгоритм Шелла[172]. Чтобы объяснить, в чем он состоит, полезно вначале вспомнить простейший случай **сортировки слиянием**, который предложил фон Нейман в 1945 году[173].

**11.1.** Даны два непересекающихся отсортированных списка $x$ и $y$. Задать функцию, сортирующую `Join[x,y]` за время $O(n)$, где $n$ — максимум длин $x$ и $y$.

**Решение.** Если полностью соблюдать симметрию между $x$ и $y$, то, например, так

```
onion[x_,{}]:=x; onion[{},y_]:=y;
onion[x_,y_]:=If[First[x]<First[y],
                        Prepend[onion[Rest[x],y],First[x]],
                        Prepend[onion[x,Rest[y]],First[y]]]
```

В простейшем варианте **алгоритм Шелла** можно описать так:

• На первом шаге упорядочиваются пары $\{x_i, x_{\lfloor n/2\rfloor+i}\}$.

• На втором шаге упорядочиваются четверки

$$\{x_i, x_{\lfloor n/4\rfloor+i}, x_{\lfloor n/2\rfloor+i}, x_{\lfloor 3n/4\rfloor+i}\}.$$

• На третьем шаге упорядочиваются восьмерки

$$\{x_i, x_{\lfloor n/8\rfloor+i}, x_{\lfloor n/4\rfloor+i}, x_{\lfloor 3n/8\rfloor+i}, x_{\lfloor n/2\rfloor+i}, x_{\lfloor 5n/8\rfloor+i}, x_{\lfloor 3n/4\rfloor+i} x_{\lfloor 7n/8\rfloor+i}, \}.$$

И так далее. Преимущество этого алгоритма по сравнению с простой сортировкой состоит в том, что здесь сохраняется структура, возникшая на предыдущих шагах. А именно, на каждом шаге в каждом блоке происходит слияние двух *упорядоченных* списков одинаковой длины, а мы знаем,

---

[171] Hoare's Law of Large Problems: Inside every large problem there is a small problem struggling to get out.

[172] D.L.Shell, A high-speed sorting procedure. — Comm. Ass. Comp. Mach., 1959, vol.2, N.7, p.30–32.

[173] J. von Neumann, Collected Works, vol.5, MacMillan, N.Y., 1963, p.196–214.

что эта операция может быть осуществлена за линейное время. Детальный анализ этого алгоритма проведен в упомянутой выше книге Кнута[174], стр.102–115, где в частности, доказан (нетривиальный!) результат Паперного и Стасевича, утверждающий, что *в худшем* случае сложность этого алгоритма равна $O(n^{1.5})$.

**11.2.** Реализуйте алгоритм сортировки Шелла.

В действительности, конечно, здесь можно брать *любую* убывающую последовательность смещений, последнее из которых равно 1, совсем не обязательно последовательность $\lfloor n/2 \rfloor, \lfloor n/4 \rfloor, \lfloor n/8 \rfloor, \ldots, 2, 1$. Более того, известно, что некоторые другие последовательности смещений в среднем дают лучший результат, чем приведенная выше последовательность.

**11.3.** Реализуйте алгоритм сортировки Шелла для другой последовательности смещений.

**Указание.** Начать нужно, конечно, с алгоритма слияния более чем двух отсортированных списков.

Однако, для больших $n$ алгоритм Шелла заведомо проигрывает алгоритмам сортировки разделением. Самый старинный этих алгоритмов, **алгоритм Хоара**, названный самим автором `quicksort`[175], по-детски прост. На каждом шаге этого алгоритма выбирается **пивот**[176]. После этого сортируемый список разделяется на два подсписка: элементы, меньшие пивота, и элементы, не меньшие пивота, к каждому из которых отдельно применяется `quicksort`. Сам Хоар выбирал в качестве пивота больший из первого и последнего элемента.

**11.4.** Реализуйте алгоритм Хоара.

**Решение.** Прежде всего, выберем пивот:

```
pivot[x_]:=If[First[x]>Last[x],First[x],Last[x]]
```

А теперь определим рекуррентную процедуру

```
quicksort[{}]={}; quicksort[{x_}]:={x};
quicksort[x_]:=Join[quicksort[Select[x,#<pivot[x]&]],
                                quicksort[Select[x,#>=pivot[x]&]]]
```

**11.5.** Убедитесь, что уже в этом самом примитивном варианте `quicksort` во много десятков раз эффективнее простой сортировки. Сравните время сортировки списков длины 500 или 1000 и максимальную длину списка, который можно отсортировать за 10 секунд.

**11.6.** Напишите *итеративную* программу, реализующую `quicksort`.

---

[174] Кнут Д. Искусство программирования. Том 3. Сортировка и поиск М. – СПб – Киев: Вильямс, 2000. 822 с.

[175] C.A.R.Hoare, Quicksort, Computer J., 1962, vol.5, N.1, p.10–15.

[176] В компьютерной литературе используется *несколько десятков* различных переводов термина `pivot` на русский: опорный элемент, ведущий элемент, средний элемент, главный элемент, центральный элемент, ... Это разнообразие побудило нас вообще отказаться от перевода.

Алгоритм Хоара *в среднем* производит сортировку за время $O(n\ln(n))$, но при неудачном выборе пивота в худшем случае получается время $O(n^2)$. Ясно, что алгоритм Хоара работает тем лучше, чем ближе длина каждого из двух получающихся списков к половине длины исходного списка, иными словами, чем ближе пивот к медиане.

**11.7.** Определить медиану списка из трех элементов

**Решение.** По любому, хотя бы так:

```
medi[{x_,y_,z_}]:=If[x<y<z||x>y>z,y,medi[RotateRight[{x,y,z}]]]
```

**11.8.** Реализуйте `quicksort` в следующем варианте. На каждом шаге в качестве пивота выбирается медиана трех *случайных* элементов списка $x$.

**11.9.** Напишите программу сортировки, которая переключается между `quicksort` и алгоритмом Шелла как только длина $n$ неотсортированного фрагмента становистя $\le 1000$.

Имеются более эффективные, чем `quicksort` алгоритмы сортировки, которые вводят на списке дополнительную нелинейную структуру, используют дополнительную память (скажем, строят вспомогательные массивы указателей или запоминают результаты предыдущих сравнений), используют случайность. Однако, все эти усовершенствования сказываются только на времени сортировки *худшего* случая и на константах, ПОРЯДОК СКОРОСТИ АЛГОРИТМА *в среднем* $O(n\log(n))$ НЕ УЛУЧШАЕТСЯ.

## 12. Порядковая статистика.[177]

Задача **порядковой статистики** состоит в следующем: дан список $x$ длины $n$ и целое число $m$, $1 \le m \le n$, требуется найти элемент $y$, который стоит на $m$-м месте в *отсортированном* списке $x$. Мы уже упоминали функцию `Ordering`, которая решает эту задачу в следующей форме: она указывает *позицию* такого элемента $y$ в исходном списке.

**12.1.** Определите функцию, решающую задачу порядковой статистики.

**Решение.** Скажем, так `orderstat[x_,m_]:=x[[Ordering[x,{m}]]]`

Правда, при этом ответ получается в формате `{y}`.

Частными случаями этой задачи являются задача нахождения минимума – при $m = 1$, максимума – при $m = n$ и **медианы** – при $m = (n+1)/2$, для списков нечетной длины[178]. Все эти функции имплементированы в системе.

| | |
|---|---|
| `Max[x]` | максимум списка $x$ |
| `Min[x]` | минимум списка $x$ |
| `Median[x]` | медиана списка $x$ |
| `Mean[x]` | среднее арифметическое списка $x$ |

[177] The IQ of the group is the lowest IQ of a member of the group divided by the number of people in the group.

[178] Для числовых списков четной длины медианой часто называют среднее арифметическое $n/2$-го и $(n/2+1)$-го элементов отсортированного списка.

Кроме того, можно ставить обратную задачу, найти позицию элемента $y$ в отсортированном списке.

Очевидно, что для того, чтобы найти максимум и минимум списка достаточно произвести $O(n)$ операций.

**12.2.** Найдите минимум списка, не пользуясь командами `Min` и `Max`.

**Решение.** Процедурное решение очевидно:

```
minimum[]=Infinity;
minimum[x_]:=Block[{y=Infinity,n=Length[x],i},
            For[i=n,i>=1,i--,If[x[[i]]<y,y=x[[i]]]];Return[y]]
```

Но, конечно, гораздо элегантней так:

```
minimum[{}]=Infinity; minimum[{x_}]:=x
minimum[x_]:=If[First[x]<Last[x],
                        minimum[Most[x]],minimum[Rest[x]]]
```

**12.3.** Найдите максимум списка, не пользуясь командами `Min` и `Max`.

Гораздо менее очевидно, что для того, чтобы найти медиану списка, достаточно $O(n)$ операций.

**12.4.** Найдите медиану списка, не пользуясь командой `Median`.

**Указание.** Делением списка на две или, лучше на три части.

С использованием команды `Sort` решение задач порядковой статистики тривиально.

**12.5.** Найдите $m$-й элемент отсортированного списка и положение элемента $y$ в отсортированном списке.

**Решение.** Да уж, тут высшего образования не надо: `Sort[x][[m]]` и `Position[Sort[x],y]`.

Настоящая проблема состоит, естественно, в том, чтобы сделать это не сортируя список.

**12.6.** Напишите программу, которая за время $O(n)$ ищет элемент, следующий за $y$ в списке, получающемся из $x$ после сортировки.

**12.7.** Напишите программу, которая за время $O(n)$ ищет элемент, предшествующий $y$ в списке, получающемся из $x$ после сортировки.

**12.8.** Напишите программу, которая за время $O(n)$ ищет элемент, стоящий на $m$-м месте в списке, получающемся из $x$ после сортировки.

**12.9.** Напишите программу, которая за время $O(n)$ ищет позицию элемента $y$ в списке, получающемся из $x$ после сортировки.

В действительности в системе и в пакете `Statistics‘DataManipulation‘` имплементированы многие другие функции порядковой статистики такие, как `Quantile`, `BinCounts`, `RangeCounts`, `CategoryCounts`, `BinLists`, `RangeLists`, `CategoryLists`, и т.д.

*В общем случае* скорость сортировки $O(n\ln(n))$ асимптотически неулучшаема. Это утверждение, однако, перестает быть верным, если

нам — как это часто бывает! — известна априорная информация о строении и вероятностном распределении ключей, например, если ключи являются последовательными натуральными числами, имеют естественную структуру прямого произведения или копроизведения и т.д. При подходящих дополнительных предположениях сложность сортировки мало отличается от сложности порядковой статистики и известны алгоритмы сложности $O(n)$.

• **Сортировка зачерпыванием** = `bucketsort` ≈ **карманная сортировка** = `binsort`: распределение ключей

• **Поразрядная сортировка**: используется, когда ключ состоит из нескольких полей, например, сортировка игральных карт по масти и рангу, сортировка дат по году, месяцу и числу.

• **Сортировка слиянием**: в случае, когда список состоит из небольшого количества уже отсортированных фрагментов.

**12.10.** Напишите программу, которая сортирует трехзначные числа вначале по первой цифре, потом по второй, и, наконец, по третьей.

> "А свиного сала не покупаете?" сказала хозяйка, следуя за ним. "Почему не покупать? Покупаю, только после."
>
> Николай Гоголь, *Мертвые души*

## § 3. Функции

> I have yet to see any problem, however complicated, which if you looked at it in the right way, did not become still more complicated.
>
> Paul Anderson

> To criticize mathematics for its abstraction is to miss the point entirely. Abstraction is what makes mathematics work. If you concentrate too closely on too limited an application of a mathematical idea, you rob the mathematician of his most important tools: analogy, generality, and simplicity. Mathematics is the ultimate in technology transfer.
>
> Ian Stewart, *Does God play dice*?

В этом параграфе мы анализируем основные понятия, связанные с отображениями множеств. Как уже упоминалось во введении, в силу скупости чистой теории множеств в использовании выразительных средств — эвфемически называемой `минимализмом` — все в ней выражается в терминах отображений[179]. В различных контекстах отображения истолковываются как функции, преобразования, движения, действия, операции,

[179] Конечно, признавая `ограниченность ресурсов` чистой теории множеств, мы ни в малейшей степени не хотим поставить под сомнение ее огромную выразительную силу.

списки, наборы, перестановки, симметрии, функционалы, операторы, формы, семейства, последовательности, слова, векторы, матрицы, etc. — все на свете является функцией.

**1. Функции.**

Понятие отображения или, как принято говорить в школьной математике, функции, является одним из основных в математике. **Отображение** $f : X \longrightarrow Y$ сопоставляет каждому элементу $x$ множества $X$ единственный элемент $y$ множества $Y$, обозначаемый обычно $f(x)$ и называемый **образом** элемента $x$ под действием $f$. Равенство $y = f(x)$ записывается также в виде $f : x \mapsto y$ и читается, например, так: $x$ **отображает** $x$ **в** $y$ или $f$ **переводит** $x$ **в** $y$.

Множество $X$ называется **областью определения** отображения $f : X \longrightarrow Y$ и обозначается $D(f)$, от английского `domain`, а множество $Y$ называется **областью значений** отображения $f$ и обозначается $R(f)$, от английского `range`. Область определения и область значений часто называются просто **областью** и **кообластью**.

Говорят также, что $f$ действует из $X$ в $Y$. Напомним, что $D(f)$ и $R(f)$ входят в определение отображения $f$. Иными словами, два отображения $f$ и $g$ называются **равными**, если и только если

- их области совпадают, $D(f) = D(g) = X$,
- их кообласти совпадают, $R(f) = R(g) = Y$,
- для любого $x \in X$ выполнено равенство $f(x) = g(x)$.

Однако, основным в определении отображения является сам *способ*, которым элементам $X$ сопоставляются элементы $Y$. С чисто экстенсиональной точки зрения этот способ тоже должен задаваться некоторым множеством, ведь в ортодоксальной теории множеств вообще не существует ничего, кроме множеств! Подмножество

$$\Gamma(f) = \{(x, f(x)), \mid x \in X\} \subseteq X \times Y,$$

состоящее из всех пар $(x, f(x))$, называется **графиком** отображения $f : X \longrightarrow Y$. В [VH2] мы подробно обсуждаем отличия этого обычного в математике *экстенсионального* определения функции от понятия функции, принятого в компьютерной алгебре.

Пусть $X = \{x_1, \ldots, x_m\}$ и $Y = \{y_1, \ldots, y_n\}$ два конечных множества. Данными какой структуры удобнее всего задавать отображение $f : X \longrightarrow Y$? Ответ на этот вопрос зависит от контекста, решаемых задач и используемых алгоритмов. Пусть $f(x_i) = z_i \in Y$. При этом мы считаем сами множества $X$ и $Y$ фиксированными. Разумеется, в учебных целях мы могли бы задавать отображение как *тройки*, состоящие из области, кообласти и какой-то дополнительной структуры данных. Однако, в реальных вычислениях никто не будет включать область и кообласть в задание функции, так что и мы не будем заниматься подобными глупостями. Вот несколько обычных способов задавать отображения.

• Задать график отображения, {{x1,z1},...,{xm,zm}}, в программировании такой способ задания функции называется **табличным**.

• Перечислить элементы множества $X$ в *каком-то* порядке, а потом их образы, в том же порядке {{x1,...,xm},{z1,...,zm}}, такой способ задания отображения называется **полным**.

• Задать образы {z1,...,zm}, элементов $x_i$, в том порядке, как $x_i$ идут в списке, представляющем $X$ — этот способ называется **сокращенным**.

В параграфе, посвященном перестановкам, мы подробнейшим образом анализируем все эти способы для частного случая $X = Y$ и методы перехода от одной формы записи к другой. Ограничимся поэтому несколькими задачами общего характера.

**1.1.** Задайте функцию, выясняющую, является ли набор пар графиком отображения $f : X \longrightarrow Y$.

**1.2.** Напишите программу, генерирующую все отображения $X \longrightarrow Y$.

**1.3.** Напишите программу, генерирующую случайное отображение $f : X \longrightarrow Y$.

**Указание.** Проще всего породить $m$ случайных элементов множества $Y$.

Пусть $f : X \longrightarrow Y$ и $A \subseteq X$. Тогда

$$f(A) = \{f(x) \mid x \in A\}$$

называется **образом** множества $A$ относительно (или под действием) отображения $f$. Образ $f(X)$ области $X = D(f)$ под действием $f$ обозначается обычно $\mathrm{Im}(f)$, от английского `image`, и называется образом отображения $f$. Иными словами, $\mathrm{Im}(f)$ — это *множество* таких $y \in Y$, для которых существует такое $x \in X$, что $f(x) = y$.

Вообще говоря, для $y \in \mathrm{Im}(f)$ может существовать много $x$ со свойством $f(x) = y$. Любой $x \in X$ такой, что $f(x) = y$ называется **прообразом** $y$ Множество всех прообразов некоторого $y \in Y$ обозначается $f^{-1}(y)$ и называется **полным прообразом** элемента $Y$. Разумеется, если $y \notin \mathrm{Im}(f)$, то $f^{-1}(y) = \emptyset$. Вообще, для любого подмножества $B \subseteq Y$ его **полный прообраз** $f^{-1}(B)$ определяется как

$$f^{-1}(B) = \{x \in X \mid f(x) \in B\} = \bigcup_{y \in B} f^{-1}(y).$$

**1.4.** Задайте функцию, вычисляющую для подмножества $A \subseteq X$ его образ относительно отображения $f : X \longrightarrow Y$.

**1.5.** Задайте функцию, вычисляющую для подмножества $B \subseteq Y$ его прообраз относительно отображения $f : X \longrightarrow Y$.

## 2. Предметы и ящики.

Математик (*вынимая из головы шар*)
Я вынул из головы шар.
Я вынул из головы шар.
Я вынул из головы шар.
Я вынул из головы шар.
Андрей Семенович
Положь его обратно.
Положь его обратно.
Положь его обратно.
Положь его обратно.

Даниил Хармс, *Математик и Андрей Семенович*

Традиционная комбинаторика говорит о **задачах размещения**, в которых требуется разместить $n$ `предметов` по $m$ `ящикам`:

- рассадить $n$ кроликов по $m$ клеткам,
- раскрасить $n$ стран в $m$ цветов,
- разложить $n$ писем по $m$ конвертам,
- опустить $n$ шаров по $m$ урнам,
- распределить $n$ частиц по $m$ энергетическим уровням,

С нашей точки зрения всюду здесь речь идет об **отображениях** $n$-элементного множества $X$ в $m$-элементное множество $Y$, удовлетворяющих определенным ограничениям. При этом элементы множества $X$ соответствуют предметам, элементы множества $Y$ — ящикам, а отображение $f : X \longrightarrow Y$ сопоставляет каждому предмету его ящик.

Множество всех отображений из $X$ в $Y$ обозначается обычно одним из следующих трех образов: $\mathrm{Map}(X, Y)$, от английского `map` или `mapping`, $\mathrm{Mor}(X, Y)$, от английского `morphism`, либо, наконец, просто $Y^X$. Обратите внимание, что именно $Y^X$, а не $X^Y$.

На отображение $f : X \longrightarrow Y$ могут накладываться те или иные условия. Простейшие условия на отображения — инъективность, сюръективность и биективность — обычно формулируются в терминах того, сколько предметов — ни одного, один или несколько — попадают в каждый ящик.

• Отображение называется **инъективным**, если в каждый ящик попадает *не более* одного объекта. Иными словами, если $f(x_1) = f(x_2)$ для некоторых $x_1, x_2 \in X$, то $x_1 = x_2$. Про такое отображение можно сказать, что оно удовлетворяет `принципу запрета Паули`: два фермиона не могут иметь один и тот же набор квантовых чисел. Множество инъективных отображений из $X$ в $Y$ обозначается через $\mathrm{Inj}(X, Y)$.

• Отображение называется **сюръективным**, если в каждый ящик попадает *хотя бы* один объект. Иными словами, для каждого $y \in Y$ *существует* такое $x \in X$, что $f(x) = y$. Множество сюръективных отображений из $X$ в $Y$ обозначается через $\mathrm{Sur}(X, Y)$.

• Отображение называется **биективным**, если в каждый ящик попадает *ровно* один объект, иначе говоря, если оно *одновременно* инъективно и

сюръективно. Таким образом, для каждого $y \in Y$ существует *единственное* такое $x \in X$, что $f(x) = y$. Множество биективных отображений из $X$ в $Y$ обозначается через $\mathrm{Bij}(X, Y)$.

С точки зрения **теории перечисления**, т.е. подсчета отображений данного типа, как предметы, так и ящики могут быть

- **различимыми**,
- **неразличимыми**
- или **частично различимыми**.

Это значит, что мы можем подсчитывать либо число самих отображений $X \longrightarrow Y$ с какими-то ограничениями, либо число *орбит* таких отображений под действием симметрических групп $S_X$ и/или $S_Y$ или каких-либо их подгрупп. При этом мы будем получать существенно различные ответы.

Например, существует $m$ различных способов положить один предмет в $m$ различимых ящиков и *единственный* способ положить этот же предмет в $m$ неразличимых ящиков. С точки зрения индивидуального студента совершенно не все равно, какую оценку он получил на экзамене, но с точки зрения статистики успеваемости важно лишь **сколько** студентов получило на экзамене оценку `почти удовлетворительно` или `весьма хорошо`.

В некоторых разделах комбинаторики, например, в теории графов в этом контексте обычно говорят о **помеченных** и **непомеченных** предметах и ящиках.

Одной из наших первых целей в этом параграфе является получение формул для числа инъективных и сюръективных отображений. Уже простейшая из этих формул, которая утверждает, что $|\mathrm{Inj}(X, Y)| = 0$ при $|X| > |Y|$, и, двойственным образом, $|\mathrm{Sur}(X, Y)| = 0$ при $|X| < |Y|$, оказывается настолько полезной, что мы посвятим ей отдельный пункт.

### 3. Принцип Дирихле.[180]

В наиболее простой форме **принцип Дирихле** утверждает, что если $|X| > |Y|$, то не существует инъективных отображений $f : X \longrightarrow Y$. Иными словами, если нам даны $n$ шаров, которые мы хотим распределить по $m$ ящикам, причем $n > m$, то хотя бы в одном ящике окажется по крайней мере два шара.

В русской учебной литературе этот принцип часто называется также **принципом клеток и кроликов**: если нам дано $n$ кроликов, которых мы хотим рассадить по $m$ клеткам, причем $n > m$, то хотя бы в одной клетке окажется по крайней мере два кролика. В англоязычной литературе этот же принцип обычно называется `pigeonhole principle` = принцип ящика для писем: если имеются $n$ писем, которые нужно распихать по $m$ отделениям ящика для писем, причем $n > m$, то хотя бы в одном отделении окажется по крайней мере два письма.

---

[180] Everybody who reasons carefully about anything is making a contribution to mathematics. ©Richard Feynman, *The Character of Physical Law*

Представляется совершенно невероятным, чтобы столь тривиальное наблюдение приводило к нетривиальным результатам, — но, как замечает по этому поводу Олег Александрович Иванов,[181] откуда и следующие задачи: "однако, . . . "

**3.1.** Легко видеть, что шахматную доску размера $8 \times 8$ можно покрыть костяшками домино $2\times1$ так, чтобы каждая костяшка покрывала две соседние клетки. Можно ли сделать то же самое с доской, из которой вырезаны две клетки на противоположных концах диагонали?

**Решение.** Нет, потому что костяшка домино покрывает одну белую и одну черную клетку, а обе вырезанные клетки одного цвета, при этом неважно даже, что вырезаны клетки на одной из *главных* диагоналей.

**3.2.** Покажите, что для любого покрытия шахматной доски $\times6$ костяшками домино $2 \times 1$ существует такое разрезание доски вертикальной или горизонтальной линией, которое не разрезает ни одной костяшки. Верно ли то же самое для доски $8 \times 8$?

**Указание.** Сколько костяшек может разрезать такая линия?

**3.3.** На белую плоскость брызнули черной краской. Докажите, что найдутся две точки одного цвета на расстоянии 1 метр друг от друга.

**Указание.** Таких пар точек очень много. Воспользуйтесь **принципом Лагранжа** и считайте, что 2 — переменная величина, например, что $2 = 3$. Тогда на белое пространство брызнули красками двух цветов, скажем черной и красной. Вам будет много легче увидеть решение.

Сформулируем теперь **принцип Дирихле** в чуть большей общности, как утверждение о ядре любого отображения $m$-элементного множества в $n$-элементное.

**3.4.** Пусть $m = m_1 + \ldots + m_n - n + 1$ кроликов рассажены по $n$ клеткам. Тогда найдется такой номер $i$, что в $i$-й клетке окажется по крайней мере $m_i$ кроликов.

**Указание.** $+1$.

**3.5.** В классе 30 учеников. Андрюша Крылов сделал в диктанте 13 ошибок, а остальные меньше. Докажите, что найдутся три ученика, сделавшие одинаковое количество ошибок — или не сделавшие ни одной.

**3.6.** Докажите, что у некоторой натуральной степени числа 17 семь последних цифр равны 0000001.

**3.7.** Группа из 21 студента *успешно* сдала сессию из трех экзаменов. Докажите, что по крайней мере 3 студента сдали сессию с одинаковыми оценками.

### 4. Инъективные отображения.

Здесь мы вычислим количество инъективных отображений $n$-элементного множества в $m$-элементное.

---

[181] Иванов О.А. Избранные главы элементарной математики.— СПб.: Изд-во СПбГУ, 1995, pp. 1-223., Гл. 7

**4.1.** Пусть $|X| = m$, $|Y| = n$. Докажите, что $|\operatorname{Inj}(X, Y)| = [n]_m$.

**Решение.** Пусть $X = \{x_1, \ldots, x_m\}$. Образом $x_1$ при инъективном отображении $f : X \longrightarrow Y$ может быть любой из $n$ элементов $Y$. После того как $f(x_1)$ зафиксирован, образом $x_2$ может быть любой из оставшихся $n - 1$ элемента $Y$. После того как $f(x_1)$ и $f(x_2)$ зафиксированы, образом $x_3$ может быть любой из оставшихся $m-2$ элементов $Y$. Продолжая действовать таким образом, на последнем шаге мы заифксировали $f(x_1), \ldots, f(x_{m-1})$, после чего образом $x_m$ может быть любой из оставшихся $n - (m - 1)$ элементов $Y$. Остается применить правило произведения.

**4.2.** Докажите, что $|\operatorname{Bij}(X, Y)| = |X|!$ в случае $|X| = |Y|$ и 0 в противном случае.

Напомним, что, *кроме того*, в § 1 настоящей главы мы вводили понятие *возрастающего* факториала. Она становится интересным, в случае, когда инъективность не имеет места.

**4.3.** Докажите, что число *упорядоченных* размещений $n$ предметов по $m$ ящикам равно $[n]^m$.

**Указание.** Для доказательства нужно лишь заметить, сколькими способами можно разместить $i$-й объект, если $i - 1$ объект уже размещены и применить правило произведения.

**4.4.** Перечислите все упорядоченные размещения трех поросят в трех домиках. С точки зрения волка отнюдь не все равно не только то, в каком именно домике (соломенном, деревянном или каменном) находятся поросята, но и то, в каком порядке съедать поросят, оказавшихся в одном домике. Он может начать с самого жирного или, наоборот, оставить самого жирного на десерт.

Задача о трех поросятах во всевозможных вариантах — неразличимые и различимые поросята и домики, инъективность и неинъективность, сюръективность и несюръективность, случай, когда поросята построили пять домиков, etc. — обсуждается в замечательной книге Мацумаса Анно *Три поросенка*.

Вернемся теперь к инъективным отображениям.

**4.5.** Напишите программу, генерирующую все инъективные отображения $f : X \longrightarrow Y$.

**4.6.** Напишите программу, генерирующую случайное инъективное отображение $f : X \longrightarrow Y$.

**Указание.** Ничем не отличается от задачи генерации произвольного случайного отображения, за исключением того, что на каждом шаге вытянутый элемент выьрасывается.

**4.7.** Докажите, что инъективность отображения $f : X \longrightarrow Y$ эквивалентна тому, что для любого подмножества $A \subseteq X$ выполняется равенство $A = f^{-1}(f(A))$.

**4.8.** Докажите, что инъективность отображения $f : X \longrightarrow Y$ эквивалентна тому, что для любых $A, B \subseteq X$ имеет место равенство $f(A \cap B) = f(A) \cap f(B)$.

**5.Сюръективные отображения.**

Пусть $|X| = m$, $|Y| = n$. Здесь мы получим две формулы — знакопеременную и комбинаторную — для числа $|\operatorname{Sur}(X, Y)|$ сюръективных отображений. Написать знакопеременную формулу совсем просто.

**5.1.** Пусть $|X| = m$, а $|Y| = n$. Докажите, что

$$|\operatorname{Sur}(X, Y)| = n^m - n(n-1)^m + \binom{n}{2}(n-2)^m - \ldots + (-1)^{n-1}\binom{n}{n-1}.$$

**Решение.** Это просто формула решета, примененная к ситуации

$$\operatorname{Sur}(X, Y) = \operatorname{Map}(X, Y) \setminus \bigcup_{i=1}^{n} \operatorname{Map}(X, Y \setminus \{y_i\}).$$

А именно, первое слагаемое это общее количество *всех* отображений $X \longrightarrow Y$, равное $n^m$. Так как каждое из множеств $Y \setminus \{y_i\}$ содержит $n-1$ элемент, а их количество равно $m$, то второе слагаемое это просто сумма порядков $\operatorname{Map}(X, Y \setminus \{y_i\})$. Однако, при этом элементы, попавшие в их попарные пересечения, оказались выброшеными два раза, их количество нужно снова прибавить. Словом, обычное включение-исключение.

Комбинаторная формула потребует некоторой подготовки. Пусть $f :\longrightarrow Y$ — произвольное отображение. Ему можно сопоставить *разбиение* множества $X$, блоками которого являются полные прообразы $f^{-1}(y)$ точек $y \in \operatorname{Im}(f)$. Это разбиение называется **ядром** отображения $f$ и *в комбинаторике* обозначается $\operatorname{Ker}(f)$ или $N(f)$, от английского `kernel` или `nucleus`. Иными словами, ядро — это разбиение $X$, отвечающее отношению эквивалентности $\sim$, определенному **слоями** alias **множествами уровня** отображения $f$. Для этого отношения эквивалентности

$$x_1 \sim x_2 \qquad \Longleftrightarrow \qquad f(x_1) = f(x_2).$$

В алгебре, когда рассматриваются **гомоморфизмы** алгебраических систем, ядро определяет не просто отношение эквивалентности на $X$, а **конгруэнцию** на $X$. Поэтому ядро гомоморфизма полностью определяется заданием одного из своих блоков, обычно прообраза нейтрального элемента, и *в алгебре* обычно именно этот блок называется ядром и обозначается $\operatorname{Ker}(X)$.

В предыдущем пункте мы убедились, что убывающие факториалы служат для перечисления инъективных отображений. А сейчас мы сможем оценить, в чем пафос чисел Стирлинга.

**5.2.** А теперь еще раз посчитайте, сколько существует сюръективных отображений $m$-элементного множества на $n$-элементное?

**Ответ.** Столько же, сколько *упорядоченных* разбиений $n$-элементного множества на $n$ непустых блоков, т.е. $n!\left\{ {m \atop n} \right\}$. В самом деле, ядро $N(f)$ *сюръективного* отображения $f : X \longrightarrow Y$, где $|X| = m$, $|Y| = n$, представляет собой разбиение $X$ на $n$ штук *непустых* блоков. Таким образом, общее количество ядер *сюръективных* отображений $X \longrightarrow Y$ равно $\left\{ {m \atop n} \right\}$. Однако, отображение не полностью определяется своим ядром. Важно еще знать, в какой элемент $Y$ переходят элементы каждого из блоков. Иными словами, чтобы полностью определить отображение $f$, мы должны еще установить биекцию между блоками ядра и элементами множества $Y$. Но, как мы знаем из предыдущего параграфа, количество биекций между двумя $n$-элементыми множествами равно $n!$.

**5.3.** Напишите программу, генерирующую все сюръективные отображения $f : X \longrightarrow Y$.

В отличие от случая произвольных/инъективных отображений следующая задача не совсем очевидна — если, конечно, не делать каких-то заведомых глупостей типа выбора отображения со случайным номером из списка всех сюръективных отображений.

**5.4.** Напишите программу, генерирующую случайное сюръективное отображение $f : X \longrightarrow Y$.

**Указание.** Сделать это, задавая образы элементов, совсем непросто. Проще всего задать случайное разбиение $X$ на $n$ блоков и потом случайную перестановку степени $n$. Имеются и другие эффективные методы сделать это, которые мы здесь не обсуждаем.

**5.5.** Докажите, что сюръективность отображения $f : X \longrightarrow Y$ эквивалентна тому, что для всех подмножеств $B \subseteq Y$ выполняется равенство $B = f(f^{-1}(B))$.

### 6. Чистые и анонимные функции.[182]

Большинство команд функционального программирования требуют обращения к функции по имени. Вот несколько примеров команд и конструкций, в которых необходимо упоминать имя функции.

• Команды применения функции к спискам или элементам списка, команды композиции и итерации такие, как `Apply`, `Map`, `Fold`, `Nest`, и многие другие.

• Команды, использующие критерии такие, как `Select`, `Sort`, `Split` и другие

• Различные конструкции такие, как `PatternTest`.

Конечно, мы всегда можем задать имя функции при помощи конструкции `f[x_]:=rhs`. Однако, что если эта функция требуется нам ровно один раз в определенном контексте для применения команды функционального программирования? Оправдано ли в этом случае присвоение такой функции

[182] All I want is a warm bed, a kind word and unlimited power.

специального имени? В этом параграфе мы рассмотрим форматы чистой и анонимной функции, которые позволяют обращаться по имени к функции, у которой нет обычного имени.

С математической точки зрения команда `Function` представляет собой реализацию **операции абстрагирования** $\lambda$-исчисления Черча. Эта операция реализуется также в некоторых других языках программирования, в частности, в `Lisp`.

| | | |
|---|---|---|
| `&` | `Function[x,y]` | применение чистой функции $x \mapsto y$ |
| `#` | `Slot[]` | аргумент `Function` |
| `#n` | `Slot[n]` | $n$-й аргумент `Function` |
| `##` | `SlotSequence[]` | последовательность аргументов `Function` |
| `##n` | `SlotSequence[n]` | последовательность аргументов, начиная с $n$-го |

Лишь очень немногие функции такие, как `Cos`, `Log`, `Floor`, `PrimeQ` или `Equal`, имеют специальные имена. Многие даже часто встречающиеся функции не имеют обычных имен, не включающих имени переменной. Как, например, *называются* функции $x \mapsto 1 + x^2$ или $x \mapsto 1/(1 + x)$? Так вот, при помощи команды `Function` можно обратиться к любой функции $x \mapsto f(x)$ в формате **чистой функции** `Function[x,f[x]]`. Тем самым, двум упомянутым выше функциям присваиваются имена `Function[x,1+x^2]` и `Function[x,1/(1+x)]`. Теперь в любом контексте мы можем обращаться к этим функциям по имени.

В действительности в большинстве случаев чистые функции вызываются в *операторном* формате или, как принято говорить, в формате **анонимных функций** `f[#]&`. Дело в том, что в большинстве ситуаций не только имя функции, но и имена используемых ей аргументов не имеют никакого значения. В реальной ситуации мы, скорее всего, назовем только что упомянутые функции `(1+#^2)&` и `1/(1+#)&`.

Использованный здесь оператор `#` = `Slot`, обозначает *аргумент* чистой функции, которому мы не хотим присваивать никакого индивидуального имени. Оператор `&` есть просто сокращение для `Function`, и обозначает *применение* чистой функции.

Если у анонимной функции несколько аргументов, то они обозначаются `#1,#2,#3` и так далее. В случае, когда `Function[body]` или `body&` применяется к списку аргументов, #1 заменяется первым аргументом, #2 — вторым и так далее. Мы уже много раз использовали конструкцию анонимной функции в конкретных ситуациях. Рассмотрим одну лингвистическую тонкость в которой мы до сих пор не встречались.

Последовательность слотов `##` = `SlotSequence` заменяет *последовательность* аргументов. Однако, при этом `##n` имеет совершенно другой смысл, чем в `#n`. А именно, `##n` обозначает последовательность всех аргументов анонимной функции, *начиная с n-го*. Команду `SlotSequence` удобно использовать для манипуляции с аргументами функции $f$ даже в тех случаях, когда у нее уже есть имя.

**6.1.** Задайте $f(y,z,x)$ не применяя никаких операций к последовательности аргументов $x,y,z$.

**Решение.** Можно `f[#2,#3,#1]&[x,y,z]`, но лучше `f[##2,#1]&[x,y,z]`.

**6.2.** Задайте функцию $f(a,\dots,z,a,\dots,z,a\dots,z)$ где в качестве аргументов 3 раза повторяются буквы латинского алфавита.

**Решение.** Проще всего так:

```
f[##,##,##]&[Apply[Sequence,
                   ToExpression[CharacterRange["a","z"]]]]
```

**7. Композиции и итерации функций.**

Два отображения $f$ и $g$ такие, что $R(f) = D(g)$ можно **скомпонировать**. Точнее, пусть $f : X \longrightarrow Y$ и $g : Y \longrightarrow Z$ суть два отображения, область значений первого из которых совпадает с областью определения второго. Тогда их **композиция** $g \circ f : X \longrightarrow Z$ задается посредством равенства $(g \circ f)(x) = g(f(x))$, для любого $x \in X$. При этом $g \circ f$ читается как `композиция` $f$ `и` $g$ или $g$ `кружочек` $f$. Обратите внимание на порядок факторов: отображение $f$, действующее первым, записывается вторым. Это связано с тем, что мы пишем функцию слева от аргумента, как $f(x)$. Разумеется, если бы мы использовали обозначение $(x)f$, то и композиция $f$ и $g$ записывалась бы как $f \circ g$, по формуле $(x)(f \circ g) = ((x)f)g$.

Самым важным свойством композиции отображений является ее ассоциативность. А именно, если одно из выражений $(h \circ g) \circ f$ и $h \circ (g \circ f)$ определено, то определено и второе и при этом $(h \circ g) \circ f = h \circ (g \circ f)$. В то же время композиция отображений весьма далека от коммутативности, иными словами, для двух отображений $f$ и $g$, вообще говоря, $f \circ g \neq g \circ f$.

Пусть $X$ — любое множество. Тогда определено отображение $\mathrm{id} = \mathrm{id}_X : X \longrightarrow X$ такое, что $x \mapsto x$ для всех $x \in X$, называемое **тождественным** отображением $X$ на себя. Обозначение $\mathrm{id}_X$ является сокращением от английского `identical`. Тождественные отображения действительно ведут себя как нейтральные элементы для композиции функций, но отсутствие коммутативности накладывает свой отпечаток. А именно, для отображений из $\mathrm{Map}(X,Y)$ тождественное отображение $\mathrm{id}_X$ выступает как **правая единица**, а $\mathrm{id}_Y$ — как **левая единица**. Иными словами, для любого $f \in \mathrm{Map}(X,Y)$ имеем $f \circ \mathrm{id}_X = f = \mathrm{id}_Y \circ f$. По отношению к отображениям $X$ в себя $\mathrm{id}_X$ является уже **двусторонней единицей**.

Самым важным свойством биективных отображений является то, что они обратимы. Иначе говоря, в этом случае сопоставление $y \mapsto f^{-1}(y)$ задает биективное отображение $f^{-1} : Y \longrightarrow X$, называемое отображением, **обратным** к $f$. Определенное так в терминах прообразов отображение $f^{-1}$ действительно является обратным к $f$ по отношению к композиции. При этом $(f^{-1})^{-1} = f$, так что в действительности $f$ и $f^{-1}$ совершенно равноправны.

| | |
|---|---|
| `Identity` | тождественная функция |
| `Composition[f,g,h]` | композиция функций $f, g, h$ |
| `ComposeList[{f,g,...},x]` | список $\{x, f(x), g(f(x)), \dots\}$ |
| `InverseFunction[f]` | обратная к $f$ функция |

Внутренняя функция `Composition[f,g]` представляет собой теоретико-множественную композицию отображений $f$ и $g$. Иными словами, вычисление `Composition[f,g][x]` возвращает `f[g[x]]`. Функция `Composition` имеет две операторных записи.

• Оператор `@` представляет собой теоретико-множественную запись композиции *справа налево*, когда `(f@g@h)[x]===f[g[h[x]]]`.

• Оператор `//` представляет собой теоретико-категорную запись композиции *слева направо*, когда `(f//g//h)[x]===h[g[f[x]]]`.

**7.1.** Убедитесь, что `Mathematica` знает тот факт, что при обращении композиции двух отображений порядок факторов заменяется на противоположный, $(f \circ g)^{-1} = g^{-1} \circ f^{-1}$

Последовательное $n$-кратное применение отображения $f : X \longrightarrow X$ называется $n$-й **итерацией** $f$ и обозначается $f^{\circ n} = f \circ \dots \circ f$ (отображение $f$ применяется $n$ раз). Согласно этому определению, $f^{\circ 0} = \mathrm{id}_X$, $f^{\circ 1} = f$.

Команда `Nest[f,x,n]` вычисляет значение $f^{\circ n}(x)$. Вторая команда, связанная с итерациями функции `NestList[f,x,m]`, генерирует *список* значений $(x, f(x), f^2(2), \dots, f^n(x))$.

| | |
|---|---|
| `Nest[f,x,n]` | вычисление $f^n(x)$ |
| `NestList[f,x,n]` | вычисление списка $\{x, f(x), \dots, f^n(x)\}$ |

Следующие задачи показывают, что, строго говоря, задание этих функций является излишним, но в практических вычислениях итерации встречаются настолько часто, что удобно иметь для них специальные названия.

**7.2.** Выразите функцию `Nest` через `Composition`.

**Указание.** Для этого нужно лишь создать состоящую из $n$ членов *последовательность* $f, \dots, f$.

**7.3.** Выразите функцию `NestList` через `ComposeList`.

Говорят, что $f : X \longrightarrow X$ — отображение **конечного порядка**, если найдется такое $n \in \mathbb{N}$, что $f^{\circ n} = \mathrm{id}_X$. Если такого натурального $n$ не существует, то говорят, что порядок $f$ бесконечен.

**7.4.** Докажите, что если $f^{\circ n} = \mathrm{id}_X$, то $f$ — биекция.

**7.5.** Вычислите $n$-ю итерацию отображения $x \mapsto x/\sqrt{1+x^2}$.

**Ответ.** $x \mapsto x/\sqrt{1+nx^2}$.

Наряду с обычными итерациями для функций от двух аргументов во многих итеративных алгоритмах широко применяется **фолдинг**[183], при котором на каждом следующем шаге в функцию скармливается новое значение одного из аргументов.

| | |
|---|---|
| `Fold[f,x,{a,b,c}]` | вычисление $f(f(f(x,a),b),c)$ |
| `FoldList[f,x,{a,b,...}]` | вычисление списка $\{x, f(x,a), f(f(x,a),b), \dots\}$ |

Типичным примером функций, над которыми особенно часто производится фолдинг, являются ассоциативные алгебраические операции. Например, `Fold[Plus,x,{a,b,c}]` дает $((x+a)+b)+c$. Обратите внимание, что для неассоциативных операций при этом получается *левонормированный* результат.

**7.6.** Даст ли вычисление `Fold[Plus,x,{a,b,c}]` значение $x^{a^{b^c}}$?

**7.7.** Выразите сумму списка в терминах функции `Fold`.

**Решение.** Сумма списка вычисляется посредством `Apply[Plus,x]`. Однако, можно и так `Fold[Plus,First[x],Rest[x]]`.

**7.8.** Выразите произведение списка в терминах функции `Fold`.

**7.9.** Задайте функцию, которая в применении к тройке $f$, $x$, $y$, где $y = \{..., a, b, c\}$ даст *правонормированный* результат $\dots f(a, f(b, f(c, x)))\dots$).

**Решение.** Ну, например, `Fold[f[#2,#1]&,x,Reverse[y]]`.

**8. Траектории и неподвижные точки.**

> All the wonders of our Universe can in effect be captured by simple rules, yet there can be no way to know all the consequences of these rules, except in effect just to watch and see how they unfold.
>
> Stephen Wolfram. A New Kind of Science

Пусть $f : X \longrightarrow X$ — преобразование множества $X$, иными словами, отображение $X$ в себя. Зафиксируем точку $x \in X$ и будем последовательно применять $f$ к этой точке и ее образам. Получившаяся последовательность $x, f(x), f^2(x), f^3(x), \dots$ — или *множество* $\{x, f(x), f^2(x), f^3(x), \dots\}$ — называется **траекторией** точки $x$ под действием $f$. Поведение траекторий функции $f$ называется **динамикой** этой функции.

В отличие от функции `Nest[f,x,n]`, которая вычисляет $f^n(x)$ и на этом успокаивается, функция `NestWhile[f,x,test]` последовательно вычисляет $x, f(x), f^2(x), \dots$, до тех пор, пока `test` дает значение `True` и возвращает *первое* значение $f^n(x)$, для которого тест дает значение `False`. Функция

[183] Альтернативными названиями этой операции являются немецкое `фальцовка` и латинское `флексура`. Мы остановились на английском названии, так как только оно используется в программировании.

`NestWhile[f,x,test]` возвращает *все* полученные до этого момента значения $f^i(x)$.

| | |
|---|---|
| `NestWhile[f,x,test]` | вычисление $f^i(x)$, с условием `test` |
| `NestWhileList[f,x,test]` | вычисление списка $\{x, f(x), \dots, f^i(x)\}$ с условием `test` |

В случае, когда найдется такое $n$, что $f^n(x) = x$, траектория

$$\{x, f(x), f^2(x), \dots, f^{n-1}(x)\}$$

точки $x$ называется ее **орбитой**[184]. Для функции на конечном множестве все траектории конечны. Отсюда легко вывести, что от каждой точки $x$ за конечное число шагов можно дойти до такой точки $y$, для которой найдется такое $n$, что $f^n(y) = y$. Для многих приложений особенно важен следующий частный случай. Точка $x \in X$ такая, что $f(x) = x$ называется **неподвижной точкой** отображения $f$. Орбита неподвижной точки состоит из единственного элемента. Функция `FixedPoint[f,x]` ищет неподвижную точку в траектории $x$ под действием $f$, а функция `FixedPointList[f,x]` возвращает начало траектории до неподвижной точки.

| | |
|---|---|
| `FixedPoint[f,x]` | решение уравнения $f^n(x) = f^{n+1}(x)$ |
| `FixedPointList[f,x]` | список $\{x, f(x), f^2(x), \dots, f^n(x), f^{n+1}(x)\}$ до первого решения $f^n(x) = f^{n+1}(x)$ |

**8.1.** Выразите функцию `FixedPoint` через `NestWhile`.

**Решение.** Естественнее всего так: `NestWhile[f,n,f[#]!=#&]`.

**8.2.** Выразите функцию `FixedPointList` через `NestWhileList`.

Рассмотрим симметризацию

$$\mathrm{rev} : a_1 \dots a_n \mapsto a_1 \dots a_n + a_n \dots a_1.$$

**8.3.** Для каждого числа $n \leq 100$ вычислите, сколько раз нужно применить симметризацию rev, для того, чтобы получился палиндром.

**8.4.** Для каждого числа $n \leq 100$ найдите первый палиндром, который получается при многократном применении к нему симметризации rev.

**8.5.** Сопоставим натуральному числу $n$ разность $f(n)$ наибольшего и наименьшего чисел, получающихся из $n$ перестановкой цифр. Найдите, через сколько применений $f$ мы дойдем до цикла и какова длина этого цикла.

[184] Мы не обсуждаем здесь общее понятие орбиты. В общем случае орбита обратимого преобразования $f$ является объединением начинающихся с одного и того же элемента траекторий $f$ и $f^{-1}$. Однако, орбиты необратимых преобразований определяются достаточно сложно.

Расмотрите следующую арифметическую функцию:

$$f(n) = \begin{cases} 3n+1, & \text{если } n \text{ нечетно,} \\ n/2, & \text{если } n \text{ четно.} \end{cases}$$

**8.6.** Исследуйте динамику этой функции. Для каждого $n \leq 1000$ найдите минимальное количество итераций $m$ такое, что $f^m(n) = 1$.

**8.7.** Что можно сказать об итерациях функции Эйлера $\phi : n \mapsto \phi(n)$?

**8.8.** Зафиксируем натуральное $k$ и рассмотрим арифметическую функцию $f : n \mapsto k\phi(n)$. Исследуйте поведение итераций $f^m$ функции $f$ в зависимости от $k$.

Рассмотрите множество последовательностей длины $n$, состоящих из $+1$ и $-1$ и следующее преобразование этого множества:

$$f : (u_1, u_2, \ldots, u_{n-1}u_n) \mapsto (u_1u_2, u_2u_3, \ldots, u_{n-1}u_n, u_nu_1).$$

**8.9.** Убедитесь, что если $n$ нечетно, а последовательность $u$ отлична от последовательностей $(+1, \ldots, +1)$ и $(-1, \ldots, -1)$, состоящих только из $+1$ или только из $-1$, то не существует натурального $m$ такого, что $f^m(u) = u$.

**8.10.** Убедитесь, что если $n = 2^k$, то для любой последовательности $u$ существует $m \leq n$ такое, что $f^m(u) = u$.

**8.11.** Для произвольного $n$ найдите все последовательности длины $m$ для которых существует натуральное $m$ такое, что $f^m(u) = u$.

Рассмотрим множество последовательностей длины $n$, состоящих из натуральных чисел и следующее преобразование этого множества:

$$g : (x_1, x_2, \ldots, x_{n-1}x_n) \mapsto (|x_1 - x_2|, |x_2 - x_3|, \ldots, |x_{n-1} - x_n|, |x_n - x_1|).$$

**8.12.** Убедитесь, что если $n = 2^k$, то для любой последовательности $x$ существует $m$ такое, что $f^m(x) = (0, \ldots, 0)$.

Рассмотрите преобразование

$$n_1 \ldots n_s \mapsto n_1^3 + \ldots + n_s^3$$

сопоставляющее натуральному числу сумму кубов его цифр.

**8.13.** Найдите неподвижные точки этого отображения, меньшие 100000.

Рассмотрите преобразование

$$n_1 \ldots n_s \mapsto n_1^{n_1} + \ldots + n_s^{n_s}$$

сопоставляющее натуральному числу сумму степеней его цифр, причем каждая цифра возводится в степень равную ей самой.

**8.14.** Это преобразование имеет ровно одну неподвижную точку на множестве натуральных чисел, кроме 1. Найдите эту точку.

**8.15.** Будем строить последовательность по следующему правилу. На первом шаге напишем подряд две единицы. На втором шаге впишем между ними двойку. На третьем шаге впишем 3 между любыми двумя числами, сумма которых равна 3. На четвертом шаге впишем 4 между любыми двумя числами, сумма которых равна 4 и т.д. Сколько раз в эту последовательность будет вписано число $n$?

**Ответ.** Проведя несколько первых итераций, легко заметить, что это в точности последовательность знаменателей чисел Фарея, так что $n$ вписано в нее в точности $\phi(n)$ раз.

## 9. Применение функции к элементам списка.

> Если вы с первого раза сумели написать программу, в которой транслятор не обнаружил ни одной ошибки, сообщите об этом системному программисту. Он исправит ошибки в трансляторе.
>
> Дональд Кнут

Один из важнейших навыков грамотного программирования состоит в том, чтобы тщательнейшим образом различать

• применение функции к самому списку `f[{x,y,z}]`,

• применение функции к *последовательности*, образованной элементами списка `f[x,y,z]`,

• применение функции к каждому элементу списка {`f[x]`,`f[y]`,`f[z]`}.

Нельзя сказать, чтобы понимание этого отличия представляло какую-то трудность. Однако, выработка полного автоматизма в такого рода вещах требует длительной практики, а для вложенных списков даже самый опытный пользователь при потере вигильности может легко допустить ошибку в определении уровня, на котором следует применить функцию. Вот основные команды применения функции к элементам списка. Следует обратить особое внимание на отличие `Map[f,x]` от `Apply[f,x,{1}]` — такое же, как отличие `f[{x,y}]` от `f[x,y]`, но на уровне 1!

| | | |
|---|---|---|
| `f@@x` | `Apply[f,x]` | применение функции к последовательности элементов списка |
| `f/@x` | `Map[f,x]` | применение функции к индивидуальным элементам списка на уровне 1 |
| `f@@@x` | `Apply[f,x,{1}]` | применение функции к последовательностям элементов списка на уровне 1 |
| `f//@x` | `MapAll[f,x]` | применение $f$ ко всем уровням $x$ |

**9.1.** Задайте функцию, превращающую список $\{a, b, c, \dots\}$ в список синглетонов $\{\{a\}, \{b\}, \{c\}, \dots\}$.

**Решение.** Конечно, `Map[List,x]` или, что то же самое, `Map[{#}&,x]`.

Следующая небольшая вариация на эту тему позволит нам в следующем параграфе дать еще одно, замечательно простое, решение задачи о генерации подмножеств.

**9.2.** Задайте функцию, которая превращает список $\{a, b, c, \dots\}$ в список пар $\{\{\emptyset, \{a\}\}, \{\emptyset, \{b\}\}, \{\emptyset, \{c\}\}, \dots\}$.

**9.3.** Задайте функцию, которая возводит элементы списка в квадрат.

Чтобы грамотно использовать эти команды, необходимо понимать, как устроены **спецификации уровня** = `level specification`, см. [VH2] по поводу технических подробностей. По умолчанию `Apply` использует спецификацию уровня `{0}`, а `Map` — спецификацию уровня `{1}`. Однако, в каждой из этих команд можно факультативно задавать в качестве третьего аргумента *любую* легальную спецификацию уровня. Команда `MapAll`, строго говоря, является излишней, так как она дублирует команду `Map[f,x,{0,Infinity}]`.

**9.4.** Примените функцию $f$ к каждому элементу матрицы `{{a,b},{c,d}}`.

**Решение.** Вычисление `Map[f,{{a,b},{c,d}},{2}]` даст правильное

`{{f[a],f[b]},{f[c],f[d]}}`.

Для сравнения укажем, что вычисление `Map[f,{{a,b},{c,d}}]` без явного указания уровня применит $f$ не на втором, а на первом уровне,

`{f[{a,b}],f[{c,d}]}`.

При неправильном указании уровня `Map[f,{{a,b},{c,d}},2]` — или, что то же самое, `Map[f,{{a,b},{c,d}},{1,2}]` — команда `Map` применит $f$ как на первом, так и на втором уровне,

`{f[{f[a],f[b]}],f[{f[c],f[d]}]}`.

**9.5.** Задайте функцию, возвращающую **среднее арифметическое**

$$\frac{x_1 + \ldots + x_n}{n}$$

элементов списка.

**Решение.** Ну, конечно, так `mean[x_]:=Apply[Plus,x]/Length[x]`

**9.6.** Задайте функцию, возвращающую **среднее геометрическое**

$$\sqrt[n]{x_1 \ldots x_n}$$

элементов списка.

**9.7.** Задайте функцию, возвращающую **среднее гармоническое**

$$\frac{n}{\frac{1}{x_1} + \ldots + \frac{1}{x_n}}$$

элементов списка.

Во многих случаях необходимо применить функцию не ко всем элементам списка, а только к элементам, находящимся в определенных позициях. Этому служат команды *селективного* применение функции к элементам списка `MapAt` и `MapIndexed`.

| | |
|---|---|
| `MapAt[f,x,i]` | применение $f$ к $i$-й части $x$ |
| `MapAt[f,x,{i,j}]` | применение $f$ к $(i,j)$-й компоненте $x$ |
| `MapAt[f,x,{{i},{j}}]` | применение $f$ к $i$-й и $j$-й частям $x$ |
| `MapIndexed[f,x]` | применение функции к парам, состоящим из части списка и ее адреса |

Команда `MapAt` действует так же, как `Map`, но при этом применяет функцию $f$ только к частям или компонентам списка, находящимся на указанных позициях. Для вложенных списков `MapAt` использует те же соглашения относительно адресации частей, что и команды `Extract`, `Delete`, `Insert` и `ReplacePart`. Отметим, что команда `MapIndexed` особенно удобна при работе с матрицами.

**9.8.** Задайте функцию, применяющую $f$ к первому элементу списка.

**9.9.** Задайте функцию, применяющую $f$ к последнему элементу списка.

**9.10.** Задайте функцию, применяющую $f$ к последнему элементу первого элемента списка.

**9.11.** Задайте функцию, возводящую в квадрат элементы списка на местах с нечетными номерами.

**9.12.** Задайте функцию, меняющую знак у элементов списка на местах с четными номерами.

**9.13.** Задайте функцию, заменяющую на 0 элементы списка на местах с нечетными номерами.

**9.14.** Задайте функцию, заменяющую на 1 элементы списка на местах с четными номерами.

Совершенно замечательно, что среди адресов в команде `MapAt` могут быть повторяющиеся. В этом случае функция $f$ будет многократно применяться к одному и тому же элементу.

**9.15.** Задайте функцию *два раза* применяющую $f$ ко второму элементу списка.

**Решение.** Например, так `Map[f,x,{{2},{2}}]`.

## 10. Применение функции к элементам нескольких списков.

> Если вы с первого раза сумели написать программу, в которой транслятор не обнаружил ни одной ошибки, сообщите об этом системному программисту. Он исправит ошибки в трансляторе.
>
> Дональд Кнут
>
> To tell between `Murphy's` and `Guinness` is sure not a task for beguinness.

Еще более суровые засады сулит начинающему применение функций нескольких переменных к элементам *нескольких* списков. В настоящем пункте мы лишь совсем коротко рассмотрим основные команды распределения и протаскивания функции через списки.

Основные команды **распределения** `Outer` и `Distribute` уже встречались нам при обсуждении прямых произведений. Область их применения гораздо шире, они могут производить распределение по любому заголовку, но нас здесь интересует только распределение по спискам. Распределение состоит в том, чтобы применить $f$ *ко всем* парам $(a, b)$, первый элемент которых берется из первого списка, а второй — из второго, организовав результат в виде списка, возможно вложенного.

| | |
|---|---|
| `Outer[f,x,y]` | распределить $f$ по компонентам списков $x$ и $y$ на самом глубоком уровне |
| `Outer[f,x,y,n]` | распределить $f$ по компонентам списков $x$ и $y$ на уровне $n$ |
| `Distribute[f[x],List]` | распределить $f$ по входящим в $x$ спискам |

Важнейшей синтаксической особенностью функции `Outer` является то, что по умолчанию она действует не на верхнем уровне, как подавляющее большинство функций работы со списками, а, наоборот, на самом глубоком уровне выражения. Это значит, что для вложенных списков она *всегда* дает не тот результат, который ожидает начинающий.

Функция `Distribute` имеет *один* аргумент, а именно, само выражение, и *четыре* параметра. Ее полный формат таков:

```
Distribute[x,g,f,gg,ff]
```

При вызове `Distribute` в таком формате происходит следующее: функция $g$ распределяется по тем компонентам выражения $x$, которые имеют заголовок $f$ и в получившемся выражении $f$ заменяется на $ff$, а $g$ на $gg$.

**10.1.** А теперь напишите однострочную команду, генерирующую все подмножества множества $X$.

**Решение.** Вот феерическое решение из книги Вольфрама:

```
subsets[x_]:=Distribute[Map[{{},{#}}&,x],List,List,List,Join]
```

Так как первая часть этой команды, перерабатывающая список $\{a, b, c\}$ в список пар $\{\{\emptyset, \{a\}\}, \{\emptyset, \{b\}\}, \{\emptyset, \{c\}\}\}$ уже встречалась нам в предыдущем параграфе, то все дело здесь в выборе параметров команды `Distribute`!!!

Это блестящий пример концептуального программирования, основанного на *полном* контроле над тем, что происходит при вычислении, с точки зрения математики! Мы, конечно, далеки от мысли, что нам удастся в течение полугода научить студента, который до этого никогда серьезно не занимался вычислениями, программировать в таком стиле — и, собственно говоря, не ставим такой цели.

**10.2.** Скажите, не включая компьютера, что проиойдет, если переставить в решении предыдущей задачи два последних параметра местами? Что, например, дает вычисление

```
Distribute[Map[{{},{#}}&,{a,b}],List,List,Join]?
```

**10.3.** Скажите, не включая компьютера, что произойдет, если заменить в том же решении предпоследний параметр на `Join`? Что, например, дает вычисление

```
Distribute[Map[{{},{#}}&,{a,b}],List,List,Join,Join]?
```

Как мы знаем из параграфа, посвященного прямым произведениям, с помощью команды `Outer`, если следить в ней за уровнем и производить выравнивания ответа, обычно удается добиться того же результата, что с помощью команды `Distribute`.

**10.4.** Напишите основанную на той же идее программу генерации подмножеств, использующую `Outer` вместо `Distribute`.

**Указание.** Во-первых, `Distribute` распределяется через любое количество списков, а `Outer` — только через два, поэтому необходимо применить фолдинг. Во-вторых, чтобы получить правильный ответ для вложенных списков, `Outer` нужно применять только к элементам списков на уровне 1. В-третьих, `Outer` создает список с дополнительным уровнем вложенности, который необходимо убрать.

**Решение.** Конечно, с учетом всех упомянутых лингвистических особенностей решение, использующее `Outer`, чуть сложнее решения с `Distribute`, но все равно более, чем убедительно:

```
subsets[x_]:=Fold[Flatten[Outer[Join,#1,#2,1],1]&,
                        {{},{First[x]}},Map[{{},{#}}&,Rest[x]]]
```

А вот основные команды **протаскивания**, `Inner`, `Thread` и `MapThread`. Протаскивание состоит в том, чтобы продеть $f$ через два списка *одинаковой длины* применив $f$ ко всем парам вида $(a_i, b_i)$, компоненты которых находятся на позициях с одинаковыми номерами.

| | |
|---|---|
| `Inner[f,x,y,g]` | протащить $f$ через списки $x$ и $y$, заменив заголовок на $g$ |
| `Thread[f[x,y]]` | протащить $f$ через списки $x$ и $y$ |
| `MapThread[f,x]` | протащить $f$ через входящие в $x$ списки |

Различие между `Thread` и `MapThread` не математическое, а чисто синтаксическое. Как вычисление

```
Thread[f[{a,b,c},{x,y,z}]]
```

так и вычисление

```
MapThread[f,{{a,b,c},{x,y,z}}]
```

возвращает

```
{f[a,x],f[b,y],f[c,z]}
```

**10.5.** Определите функцию `Thread` в терминах других функций применения функций к элементам списков.

**Решение.** Вот несколько совсем очевидных решений, которые работают для линейных списков:

```
Apply[f,Transpose[{{a,b,c},{x,y,z}}],1]
Inner[f,{a,b,c},{x,y,z},List]
Tr[Outer[f,{a,b,c},{x,y,z}],List]
```

Еще одно напрашивающееся решение

```
Map[f,Transpose[{{a,b,c},{x,y,z}}]]
```

приводит к синтаксической ошибке

```
{f[{a,x}],f[{b,y}],f[{c,z}]
```

Однако, стоит ввести вспомогательную функцию `ff[{x_,y_}]:=f[x,y]`, принимающую в качестве аргумента *список* аргументов функции $f$, как оно становится правильным:

```
Map[ff,Transpose[{{a,b,c},{x,y,z}}]]
```

Получить столь же ясные и естественные решения для *вложенных* списков совсем не так просто!!! Дело в том, что между тем, как применяются к вложенным спискам команды `Inner` и `Outer` с одной стороны, и тем, как действуют `Thread` и `MapThread`, имеется *принципиальное* различие. Дело в том, что по умолчанию `Inner` и `Outer` применяется на нижнем уровне, а `Thread` и `MapThread` — на верхнем уровне.

**10.6.** Задайте функцию, которая проверяет, что каждый элемент списка $x$ содержится в соответствующем элементе списка $y$.

**Решение.** Единственная команда, при помощи которой эта задача решается без дальнейших хлопот это `MapThread`:

```
Apply[And,MapThread[MemberQ,{y,x}]]
```

Распределить действие функционального *символа* $f$ при помощи команды `Thread` тоже совсем просто. Вычисление

```
Thread[f[{{a,b},{c,d}},{u,v}]]
```

возвращает нужный нам промежуточный результат

```
{f[{a,b},u],f[{c,d},v]}.
```

Проблема состоит в том, что если подставить сюда вместо $f$ функцию с высоким приоритетом такую, как `MemberQ`, система попытается эвалюировать еще до того, как применит `Thread`, что, естественно, приведет к совершенно бессмысленному ответу. Именно для борьбы с такими явлениями в системе предусмотрены команды **удерживания = подмораживания**, которые не дают провести эвалюацию функций, до тех пор, пока удерживание не будет снято соответствующей командой освобождения. В данной ситуации проще всего воспользоваться общим удерживанием `Hold` и освобождением `Release = ReleaseHold`, примерно так:

```
ReleaseHold[Thread[Hold[MemberQ][y,x]]]
```

Теперь уже, конечно, ясно, как добиться требуемого эффекта с помощью `Inner`, для этого нужно подморозить не только эвалюацию `MemberQ`, но и все подмножества, входящие в состав второго списка, чтобы команда `Inner` не вникала в их внутреннюю структуру, и освободить их только *после* протаскивания `Hold[MemberQ]`. Это можно сделать так:

```
ReleaseHold[Inner[Hold[MemberQ],Map[Hold,y],x,List]]
```

Ясно, что подобный текст находится за пределами того, что разумно требовать от начинающего. Более того, даже в книге Вольфрама в этом месте допущена ошибка, так как определенная там в терминах функции `Inner` функция `cartesianInclude`, решающая следующую задачу, работать не может — и не работает.

**10.7.** Дано множество $X$ состоящее из упорядоченных $n$-ок и список множеств $Y_1, \ldots, Y_n$. Задайте функцию, которая выбирает те элементы $X$, которые лежат в прямом произведении $Y_1 \times \ldots \times Y_n$.

**Решение.** Проще всего при помощи `MapThread`:

```
cartesianInclude[x_,y_]:=Select[x,Apply[And,
                                MapThread[MemberQ,{y,#}]]&]
```

**10.8.** Задайте функцию, которая по любому списку $x$ длины $n$ и списку натуральных чисел $y$ длины $n$ строит список той же длины, $i$-й частью которого является список, состоящий из повторенного $y_i$ раз элемента $x_i$.

**Решение.** Если список невложенный, то хотя бы так:

```
Inner[Table[#1,{#2}]&,x,y,List]
```

А теперь измените этот текст таким образом, чтобы получающаяся команда правильно работала для вложенных списков — а лучше замените `Inner` на `MapThread`.

**10.9.** Задайте функцию, которая строит *список* элементов мультимножества по его носителю и списку кратностей.

> Vail's Second Axiom: The amount of work to be done increases in proportion to the amount of work already completed.

## § 4. Перестановки

Аз есмь Алфа и Омега, начало и конец, первый и последний.

Откровение Святого Иоанна Богослова, Глава 22–13

Двадцать две основные буквы: Бог их нарисовал, высек в камне, соединил, взвесил, *переставил* и создал из них все, что есть, — и все, что будет.

*Сефер Йецира*

Some of the particulars recommended by Abulafia contributed to the aura of magic surrounding Kabbalah: the best hour for meditative permutations (known as `tzeruf`) was midnight. The meditator was to light many candles, wear phylacteries and a prayer shawl, and write out the permutations of the alphabet with ever increasing speed. The resulting ecstatic state accompanied by the desire of the soul to leave the body could be so powerful that there was even the possibility of death. At the peak of ecstatic experience there would be a rush of unintelligible language and the kabbalist had to envision a surrounding circle of angels who could help to decipher the divine message. IT WAS THE SHEER FORCE OF THE LETTERS THEMSELVES WHICH BROUGHT FORTH THE MEANING, since the only link between the `Sephirot` of non-verbal Wisdom and verbal Intelligence was through the letters of the alphabet.

Johanna Drucker, *The Alphabetic Labyrinth*[185]

В русском алфавите, как известно, букв намного больше, чем в древнееврейском, так что возможности для практической каббалистики открываются самые широкие.

Виктор Пелевин, ГКЧП как тетраграмматон

Этот параграф содержит введение в практическую каббалистику. А именно, здесь мы изучаем симметрическую группу, состоящую из всех перестановок $n$-элементного множества. В частности, читатель без труда сможет повторить все основные результаты каббалистов, относящиеся к группе $S_{22}$. Среди алгебраистов широко распространено убеждение, что комбинаторика является разделом теории групп и что вообще не существует никакой комбинаторики, кроме комбинаторики симметрической группы[186]. Однако, даже те кто не готов встать на столь радикальную точку зрения, согласятся с тем, что изучение симметрической группы представляет собой важнейшую часть комбинаторики и естественно возникает *всюду* в математике и ее приложениях — теория перечисления, теория кодирования, задачи сортировки, дискретная оптимизация, etc., etc., etc.

### 1. Запись перестановки.[187]

[185] J.Drucker, The Alphabetic Labyrinth: The Letters in History and Imagination. — Thames and Hudson, London, 1995.

[186] На самом деле, конечно, комбинаторики групп Вейля.

[187] Vidi, Vici, Veni. ©Giulio Cesare

Напомним, что **перестановкой** степени $n$ называется биекция множества $\underline{n} = \{1, \ldots, n\}$ на себя. Множество всех перестановок степени $n$ называется **симметрической группой** степени $n$ и обозначается через $S_n$.

| `Permutations[Range[n]]` | симметрическая группа степени $n$ |
|---|---|

Чаще всего перестановки изображаются двумя строками следующим образом. В первой строке перечисляются элементы множества $\underline{n}$ в *каком-то* — например, в естественном — порядке, а во второй строке под каждым элементом записывается его образ под действием перестановки. Пусть, например, $\pi$ — перестановка, переводящая $j$ в $\pi(j) = i_j$. Тогда пишут $\pi = \begin{pmatrix} 1 & \ldots & n \\ i_1 & \ldots & i_n \end{pmatrix}$ — это так называемая **полная** или **развернутая запись перестановки** $\pi$. Например, тождественная перестановка может быит записана как $\mathrm{id} = \begin{pmatrix} 1 & \ldots & n \\ 1 & \ldots & n \end{pmatrix}$.

Ясно, что если элементы множества $\underline{n}$ приведены в естественном порядке, то при этом $\pi$ вполне определяется своей второй строкой и иногда пишут просто $\pi = (i_1, \ldots, i_n)$, это так называемая **сокращенная запись перестановки**. Специалисты в области комбинаторики и теории групп не любят пользоваться сокращенной записью, из-за конфликта с обозначением циклов. Однако, в программировании обычно удобнее пользоваться сокращенной записью, а разложение на циклы задавать вложенным списком.

Для примера перечислим все элементы симметрической группы $S_3$ степени 3 в полной записи:

$$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}, \quad \begin{pmatrix} 1 & 2 & 3 \\ 1 & 3 & 2 \end{pmatrix}, \quad \begin{pmatrix} 1 & 2 & 3 \\ 2 & 1 & 3 \end{pmatrix},$$

$$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{pmatrix}, \quad \begin{pmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \end{pmatrix}, \quad \begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{pmatrix}.$$

и все элементы симметрической группы $S_4$ степени 4 в сокращенной записи:

| | | | | | |
|---|---|---|---|---|---|
| (1 2 3 4) | (1 2 4 3) | (1 3 2 4) | (1 3 4 2) | (1 4 2 3) | (1 4 3 2) |
| (2 1 3 4) | (2 1 4 3) | (2 3 1 4) | (2 3 4 1) | (2 4 1 3) | (2 4 3 1) |
| (3 1 2 4) | (3 1 4 2) | (3 2 1 4) | (3 2 4 1) | (3 4 1 2) | (3 4 2 1) |
| (4 1 2 3) | (4 1 3 2) | (4 2 1 3) | (4 2 3 1) | (4 3 1 2) | (4 3 2 1) |

Преимущество развернутой записи состоит в том, что при этом не нужно требовать, чтобы элементы первой строки стояли в естественном порядке, что особенно удобно при образовании обратной перестановки. Иными словами, если $j_1, \ldots, j_n$ — любое расположение чисел $1, \ldots, n$, и $\pi(j_h) = k_h$, то перестановка $\pi$ может быть записана и как $\pi = \begin{pmatrix} j_1 & \ldots & j_n \\ k_1 & \ldots & k_n \end{pmatrix}$. Например,

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 1 & 4 & 2 \end{pmatrix} = \begin{pmatrix} 4 & 2 & 1 & 3 \\ 2 & 1 & 3 & 4 \end{pmatrix}.$$

Таким образом, каждая перестановка степени $n$ имеет ровно $n!$ различных полных записей.

В соответствии с только что сказанным, мы будем записывать перестановку $\pi$ одним из трех следующих образов:

- **сокращенная запись** {i1,...,in};
- **полная запись** {{j1,...,jn},{k1,...,kn}};
- **табличная запись** {{j1,k1},...,{jn,kn}}.

Переход от полной записи к табличной и наоборот осуществляется внутренней командой Transpose.

**1.1.** Напишите команды, осуществляющие переход от полной записи перестановки к сокращенной и наоборот.

**Решение.** Перейти от сокращенной записи $x$ к полной совсем просто, нужно лишь предварить сокращенную запись $x$ списком $1,\ldots,n$ нужной длины:

```
{Range[Length[x]],x}
```

Получить сокращенную запись из полной чуть сложнее, так как для этого нужно прежде всего отсортировать первую строку. Проще всего это делается так: мы переходим от полной записи к табличной, сортируем пары (по умолчанию сортировка пар начинается с сортировки по первому элементу) и снова возвращаемся к полной записи. Однако, теперь первая строка отсортирована и отбрасывая ее мы получаем сокращенную запись:

```
Last[Transpose[Sort[Transpose[x]]]]
```

**1.2.** Напишите команду, порождающую по сокращенной записи перестановки список *всех* ее полных записей.

**Указание.** Можно использовать внутреннюю команду Permutations. Еще раз подчеркнем, что в большинстве ситуаций программисты рассматривает именно сокращенную запись перестановки как основную. Например, команда Permutations[list] порождает все перестановки списка list в сокращенной записи.

**1.3.** Породите случайную перестановку степени $n$.

**Решение.** Наивное решение состоит в том, чтобы породить симметрическую группу степени $n$ и выбрать в ней случайный элемент:

```
Permutations[Range[n]][[Random[Integer,{1,n!}]]]
```

Однако, уже для $n = 12$ порождение всех $12! = 479001600$ перестановок *только* для того, чтобы выбрать из них одну, представляет собой в высшей степени сомнительное предприятие.

В 1990 году Джо Кристи в пакете SymmetricGroup[188] предложил следующее изумительное решение, поражающее воображение сочетанием брутальности и изящества:

[188] В настоящее время этот пакет в основном включен в стандартный пакет Combinatorica, входящий в поставку системы.

```
RandomPermutation[n]:=Block[{xxx},
                            xxx=Table[{Random[],i},{i,1,n}];
                            Last[Transpose[Sort[xxx]]]]
```

Здесь происходит следующее. Прежде всего, генерируется список пар, состоящих из *случайного* вещественного числа в интервале $(0,1)$, *с машинной точностью*, и *номера* $i = 1,\dots,n$. После этого список сортируется — вспомним, что по умолчанию сортировка производится по первому (случайному!) элементу. В результате этого вторые элементы пар окажутся случайным образом переставлены. Вероятность того, что два числа, порожденных командой `Random[]`, совпадут, равна примерно $10^{-16}$ и *для всех практических целей* ею можно пренебречь. Порождение перестановки степени $10^6$ при помощи этой процедуры занимает примерно столько же времени, как выбор перестановки из списка всех перестановок степени 10.

Иногда используется еще одна запись перестановки, **матричная запись**. При этом каждой перестановке степени $n$ сопоставляется ортогональная матрица степени $n$, состоящая из 0 и 1. А именно, для перестановки $\pi \in S_n$ обозначим через $(\pi)$ **матрицу перестановки**, элемент которой в позиции $(i,j)$ равен $\delta_{i,\pi(j)}$.

Изобразим для примера матрицы перестановки степени 3:

$$\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix},\quad \begin{pmatrix}1&0&0\\0&0&1\\0&1&0\end{pmatrix},\quad \begin{pmatrix}0&1&0\\1&0&0\\0&0&1\end{pmatrix},$$

$$\begin{pmatrix}0&0&1\\1&0&0\\0&1&0\end{pmatrix},\quad \begin{pmatrix}0&1&0\\0&0&1\\1&0&0\end{pmatrix},\quad \begin{pmatrix}0&0&1\\0&1&0\\1&0&0\end{pmatrix},$$

Мы можем записывать перестановку задавая соответствующую матрицу перестановки — эта запись называется **матричной записью** перестановки. Например, матричная запись перестановки $(5,4,3,1,2)$ равна

```
{{0,0,1,0,0},{0,0,0,1,0},{1,0,0,0,0},{0,1,0,0,0},{0,0,0,0,1}}
```

**1.4.** Напишите команду, порождающую по сокращенной записи перестановки ее матричную запись.

**1.5.** Напишите команду, порождающую по матричной записи перестановки ее сокращенную запись.

**2. Алгебра перестановок.**

В полной записи обратная перестановка получается просто перестановкой первой и второй строк, иными словами, применением функции `Reverse`.

**2.1.** Напишите команду, вычисляющую обратную перестановку по сокращенной записи исходной перестановки.

**Ответ.** Проще всего приписать к сокращенной записи исходной перестановки список нужной длины, только не в начале, а в конце:

```
{x,Range[Length[x]]}.
```

**2.2.** Напишите команду, вычисляющую сокращенную запись обратной перестановки по сокращенной записи исходной перестановки.

**Ответ.** Вообще-то, для перестановок начального отрезка натурального ряда это просто внутренняя функция `Ordering`, которая как раз и возвращает список, состоящий из позиций последовательных натуральных чисел в исходном списке.

В общем случае естественнее всего при помощи двойного применения `Transpose`. А именно, для получения полной записи обратной перестановки достаточно приписать к исходной перестановке единичную перестановку. Чтобы перейти отсюда к обычной (сокращенной) записи можно поступить так: транспонировать полную запись, отсортировать ее по первому элементу, после чего снова транспонировать. При этом получится полная запись обратной перестановки с упорядоченной первой строкой. Чтобы получить из нее сокращенную запись обратной перестановки, достаточно отбросить в получившейся записи первую строку. В коде это выглядит так:

```
inv[x_]:=Last[Transpose[Sort[Transpose[{x,Range[Length[x]]}]]]]
```

Впрочем, вместо второго применения `Transpose` можно просто применить `Last` к каждому элементу

```
inv[x_]:=Map[Last,Sort[Transpose[{x,Range[Length[x]]}]]]
```

**Произведение перестановок** определяется как композиция отображений. В приведенных обозначениях умножение двух перестановок $\sigma$ и $\pi$ осуществляется так: нужно записать первую строку $\sigma$ как вторую строку $\pi$, тогда $\sigma\pi$ — это перестановка, первая строка которой совпадает с первой строкой $\pi$, а вторая строка — со второй строкой $\sigma$ в этой новой записи. Пусть, например,

$$\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 4 & 2 & 1 & 5 & 3 \end{pmatrix}, \qquad \pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 5 & 1 & 2 & 4 & 3 \end{pmatrix},$$

переписывая $\sigma$ в виде $\sigma = \begin{pmatrix} 5 & 1 & 2 & 4 & 3 \\ 3 & 4 & 2 & 1 & 5 \end{pmatrix}$, получим

$$\sigma\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 4 & 2 & 1 & 5 \end{pmatrix}.$$

Обратите внимание, что перестановки умножаются как отображения, а именно, **справа налево**: первой действует правая перестановка, а потом левая.

**2.3.** Напишите команду, вычисляющую произведение перестановок в сокращенной записи.

**Ответ.** Пусть $x$ и $y$ — две перестановки в сокращенной записи. Тогда `x[[y]]` — сокращенная запись их произведения.

**2.4.** Напишите команду, вычисляющую произведение перестановок в полной записи.

**2.5.** Напишите команду, вычисляющую произведение перестановок в табличной записи.

**2.6.** Убедитесь, что матричная запись согласована с алгеброй перестановок, иными словами, умножению перестановок отвечает умножение матриц, а обратная перестановка переходит в обратную — или, что в данном случае то же самое, транспонированную — матрицу.

В этом случае математик сказал бы, что отображение, сопоставляющее перестановку матрицу перестановки, является **гомоморфизмом**: $(\sigma\pi) = (\sigma)(\pi)$.

**3. Генерация перестановок.**[189]

В этом и следующих разделах мы изучим несколько способов построить — или, как говорят программисты, сгенерировать — все перестановки степени $n$. Чаще всего для этого используется рекурсия, т.е. список перестановок степени $n$ строится на основе списка перестановок степени $n-1$.

**3.1.** Напишите код, который порождает все перестановки степени $n$.

**Ответ.** Можно воспользоваться рекурсией. Если мы уже породили все перестановки степени $n-1$, то перестановки степени $n$ получаются врисовыванием $n$ в каждую из них на все позиции с первой по $n$-ю, или, как мы это делаем ниже, с $n$-й по первую:

```
perm[1]={{1}};
perm[n_]:=perm[n]=Flatten[Outer[Insert[#1,n,#2]&,
                          perm[n-1],Reverse[Range[n]],1],1]
```

Индукцию можно начинать и с $n=0$, положив `perm[0]={{}}`.

Конечно, на языке `Mathematica` существуют десятки (сотни?) способов выразить ту же самую мысль. Вот еще один, при котором перестановка степени $n-1$ разбивается на начальную и конечную часть, может быть пустые, между которыми вставляется $n$:

```
permbis[1]={{1}};
permbis[n_]:=permbis[n]=Flatten[Map[
            ReplaceList[#,{x___,y___}->{x,n,y}]&,permbis[n-1]],1]
```

**3.2.** Напишите программу, которая порождает все перестановки элементов списка `list`.

**Ответ.** Точно так же, как в предыдущей задаче:

```
permut[{x_}]:={{x}};
permut[list_]:=Flatten[Outer[Insert[#1,Last[list],#2]&,
      permut[Most[list]],Reverse[Range[Length[list]]],1],1]
```

В книге Кнута описан другой алгоритм генерации перестановок степени $n$. Этот алгоритм столь же прост для реализации, хотя придумать его, пожалуй, несколько труднее. А именно, допишем к каждой перестановке степени $n-1$ полуцелые числа $\frac{1}{2}, \frac{3}{2}, \frac{5}{2}, \ldots, n-\frac{1}{2}$. Например, из перестановки (213) степени 3 получаются четыре списка $(213\frac{1}{2})$, $(213\frac{3}{2})$, $(213\frac{5}{2})$,

[189] I have the simplest tastes. I am always satisfied with the best. ©Oscar Wilde

$(213\frac{7}{2})$. После этого заменим элементы каждого из списков `list` числами $1, 2, \ldots, n$ *сохраняя порядок*. Однако, в действительности чуть проще реализовать версию этого алгоритма, в которой на последнем шаге проделывается *обратная* процедура, а именно, каждый список `list` заменяется на список позиций в `list` на которых появляются элементы `Sort[list]`. Дело в том, что при этом можно воспользоваться внутренней функцией `Ordering`

**3.3.** Напишите программу, которая реализует этот алгоритм генерации перестановок степени $n$.

**Ответ.** Как и в первой задаче этого параграфа, проще всего воспользоваться командой `Outer`, примененной на уровне 1, для того, чтобы дописать поочередно к каждой перестановке степени $n-1$ последовательно каждый из элементов списка $\{\frac{1}{2}, \frac{3}{2}, \frac{5}{2}, \ldots, n-\frac{1}{2}\}$ и указать позиции элементов получившегося списка в порядке возрастания:

```
perm[1]:={{1}}
perm[n_]:=Flatten[Outer[Ordering[Join[#1,{#2}]]&,
                        perm[n-1],1/2+Range[0,n-1],1],1]
```

**3.4.** А теперь напишите программу, которая реализует этот алгоритм генерации перестановок степени $n$ в исходной редакции.

**Ответ.** Нужно лишь дописать в предыдущем коде `inv` поверх `Ordering`.

А вот еще одна литературная обработка *той же самой* идеи. Зафиксируем перестановку степени $n-1$ и какое-то число $m$, $1 \le m \le n$. А теперь прибавим 1 ко всем числам в исходной перестановке, которые $\ge m$. При этом получится перестановка чисел $1, \ldots m-1, m+1, \ldots, n$ и дописав к ней $m$ мы получим перестановку степени $n$.

**3.5.** Напишите программу, которая реализует эту версию алгоритма генерации перестановок степени $n$.

**Указание.** Проще всего отдельно определить функцию `edit[x,m]`, которая проделывает с перестановкой $x$ степени $n-1$ описанное выше преобразование, а потом, как всегда, применить эту функцию к элементам двух списков посредством `Outer`.

**4. Генерация перестановок в лексикографическом порядке.**[190]

Внутренняя команда `Permutations` генерирует перестановки в **лексикографическом порядке**. Например, вычисление `Permutations[3]` возвращает

$$(1\,2\,3), \quad (1\,3\,2), \quad (2\,1\,3), \quad (2\,3\,1), \quad (3\,1\,2), \quad (3\,2\,1).$$

В то же время, описанная в предыдущем параграфе функция `perm` порождает перестановки в другом порядке. Например, `perm[3]` даст

$$(1\,2\,3), \quad (1\,3\,2), \quad (3\,1\,2), \quad (2\,1\,3), \quad (2\,3\,1), \quad (3\,2\,1).$$

[190] Те же тестикулы, но в ортогональной проекции ... ©Николай Фоменко

Конечно, сортировка этого списка как раз и приведет к лексикографическому порядку, но представляется совершенно абсурдным вначале порождать перестановки в *каком-то* порядке с тем, чтобы тут же их сортировать. Посмотрим, удастся ли нам породить их сразу в лексикографическом порядке?

Для этого заметим, что расположение перестановок в лексикографическом порядке обладает следующими свойствами:

• Нумерация начинается с тождественной перестановки $(12\ldots n)$.

• Все перестановки разбиваются на $n$ блоков порядка $(n-1)!$, в соответствии с возрастающим значением элемента $i=1,\ldots,n$ в первой позиции.

• Внутри $i$-го блока перестановки множества $\{1,\ldots,i-1,i+1,\ldots,n\}$ расположены лексикографически.

Ясно, что это является готовой рекуррентной процедурой для построения перестановок $n$-элементного множества в лексикографическом порядке.

**4.1.** Дайте рекуррентную процедуру генерации перестановок в лексикографическом порядке.

**Ответ.** Например, так

```
lex[{}]:={{}}; lex[x_]:=Apply[Join,
           Table[Map[Prepend[#,x[[i]]]&,lex[Delete[x,i]]],
                                    {i,1,Length[x]}]]
```

Обратите внимание, что здесь $x$ произвольный список. Таким образом, чтобы породить перестановки $\{1,\ldots,n\}$ нужно вычислить `lex[Range[n]]`.

Попробуем теперь дать нерекуррентную процедуру генерации перестановок в лексикографическом порядке. Ясно, что для решения этой задачи необходимо уметь находить для каждой перестановки ту перестановку, которая следует за ней в лексикографическом порядке.

**4.2.** Задайте функцию, сопоставляющую перестановке непосредственно следующую за ней в лексикографическом порядке.

**Ответ.** Ограничимся возможным ответом, предоставляя читателю самостоятельно разобраться — как советует Кнут! — в том, что здесь происходит и почему эта программа вообще работает.[191]

```
next[x_]:=Module[{n=Length[x],i,j,t,y=x},i=n-1;
                  While[Order[y[[i]],y[[i+1]]]==-1,i--]; j=n;
                  While[Order[y[[j]],y[[i]]]==1,j--];
                        {y[[i]],y[[j]]}={y[[j]],y[[i]]};
            Join[Take[y,i],Reverse[Drop[y,i]]]]
```

Эта откровенно жульническая программа устроена так, что вычисление `next[{4,3,2,1}]` дает `{1,2,3,4}`, как и задумано. Но почему это происходит?

---

[191] Детали на чуть другом языке можно найти в книге В. Липский. Комбинаторика для программистов – М:"Мир", 1988, 213 С.

**4.3.** Задайте функцию, сопоставляющую перестановке непосредственно предшествующую ей в лексикографическом порядке.

**4.4.** А теперь снова задайте функцию, генерирующую перестановки в лексикографическом порядке.

**Ответ.** Ну чего там, `NestList[next,Range[n],n!-1]`.

Примерно так и работает команда `LexicographicPermutations`, определенная в пакете `Combinatorica`.

Часто удобнее пользоваться не лексикографическим, а антилексикографическим порядком, при котором смотрят не на возрастание элементов начиная слева, а на убывание элементов, начиная справа. Для примера перечислим элементы $S_3$ в антилексикографическом порядке:

$$(1\,2\,3),\quad (2\,1\,3),\quad (1\,3\,2),\quad (3\,1\,2),\quad (2\,3\,1),\quad (3\,2\,1).$$

**4.5.** Дайте рекуррентную процедуру генерации перестановок в антилексикографическом порядке.

**Указание.** Для этого опишите свойства расположения перестановок в антилексикографическом порядке — это и будет почти готовой рекуррентной процедурой.

**Ответ.** Как и выше, `mutatis mutandis`:

```
antilex[{}]={{}}; antilex[x_]:=Apply[Join,
          Table[Map[Append[#,x[[i]]]&,antilex[Delete[x,i]]],
                                  {i,Length[x],1, -1}]]
```

**4.6.** Дайте итеративную процедуру генерации перестановок в антилексикографическом порядке.

**Указание.** Для этого, наверное, полезно определить перестановку, непосредственно следующую за данной в антилексикографическом порядке.

**4.7.** Задайте функцию, сопоставляющую перестановке непосредственно предшествующую ей в антилексикографическом порядке.

### 5. Транспозиции.[192]

В настоящем разделе мы покажем, что симметрическая группа порождается самыми простыми мыслимыми перестановками. Циклы длины 2 называются **транспозициями**. Таким образом, каждая транспозиция имеет вид $w_{ij} = (ij)$, $1 \le i \ne j \le n$, она переставляет элементы $i \ne j$ и оставляет все остальные элементы множества $\underline{n}$ на месте. Ясно, что $w_{ij} = w_{ji}$, так что в действительности различных транспозиций вдвое меньше, чем пар $(i, j)$, $i \ne j$, они отвечают тем парам $(i, j)$, в которых $i < j$. По определению каждая транспозиция является инволюцией, т. е. элементом порядка 2. Иными словами, $w_{ij}^{-1} = w_{ij}$.

---

[192] На практике колокол неудобно сдвигать более, чем на одно место за один раз. Остановить колокол в верхнем положении равновесия — дело довольно тонкое. ©Алиса Дикинсон *Переборы с вариациями: теория и практика* из книги *Узоры симметрии*. — М.: Мир, 1980, с.69–78.

**5.1.** Докажите, что каждая перестановка является произведением транспозиций $(ij)$, $i < j$.

**Решение.** В следующем параграфе мы докажем, что каждая перестановка является произведением *циклов* — даже независимых циклов. Покажем поэтому, что каждый цикл является произведением транспозиций. Мы утверждаем, что каждый цикл длины $l$ является произведением $l-1$ транспозиции. В самом деле, легко видеть, что

$$(i_1 i_2 \dots i_{l-1} i_l) = (i_1 i_2) \dots (i_{l-2} i_{l-1})(i_{l-1} i_l).$$

Назовем **фундаментальной транспозицией** транспозицию двух *соседних* символов,

$$s_i = w_{i,i+1} = (i, i+1), \qquad i = 1, \dots, n-1.$$

**5.2.** Докажите, что каждая перестановка является произведением *фундаментальных* транспозиций $s_1, \dots, s_{n-1}$.

**5.3.** С учетом предыдущей задачи нам достаточно доказать, что любая транспозиция $w_{ij}$, $i < j$, является произведением фундаментальных. Будем вести индукцию по $j - i$. База индукции $j - i = 1$, когда транспозиция $w_{ij}$ сама является фундаментальной. Пусть $j - i \geq 2$, причем для всех транспозиций с меньшей разностью $j - i$ утверждение уже доказано. Так как $j - i \geq 2$, то можно найти такое $h$, что $j < h < j$. Легко видеть, что $w_{ij} = w_{ih} w_{hj} w_{ih}$, причем по индукционному предположению $w_{ih}$ и $w_{hj}$ уже являются произведениями фундаментальных транспозиций. Вот явная формула для $w_{ij}$, которая получается на этом пути:

$$w_{ij} = s_i \dots s_{j-2} s_{j-1} s_{j-2} \dots s_i.$$

**5.4.** Докажите, что группа каждая перестановка является произведением транспозиций (12), (13), ..., $(1n)$.

**5.5.** Докажите, что группу $S_n$ невозможно породить менее, чем $n-1$ транспозицией.

**5.6.** Докажите, что группа $S_n$ порождается транспозицией (12) и длинным циклом $(123 \dots n)$.

**5.7.** Верно ли, что группа $S_n$ порождается длинным циклом $(123 \dots n)$ и *любой* транспозицией? Найдите необходимое и достаточное условие для того, чтобы $S_n$ порождалось длинным циклом $(123 \dots n)$ и транспозицией $(1m)$.

**Решение.** Так как группа $H = \langle (123 \dots n), (1m) \rangle$ содержит все транспозиции $(i, i+m-1)$, то она содержит и все транспозиции вида $(i, i+j(m-1))$, где второй индекс понимается по модулю $n$. В случае, когда $d = \gcd(n, m-1) = 1$ можно выбрать $j$ такое, что $j(m-1) \equiv 1 \pmod{n}$. Тем самым, в этом

случае $H$ содержит (12) и мы оказываемся в условиях предыдущей задачи. Если же $d \geq 2$, то разобьем $\underline{n}$ на $d$ блоков $\{1, 1+d, \ldots, n-d+1\}$, $\{2, 2+d, \ldots, n-d+2\}, \ldots, \{d, 2d, \ldots, n\}$. Ясно, что как $(123\ldots n)$, так и $(1m)$ переставляет блоки, так что никакое их произведение не может отобразить 1 в 2, оставив при этом $1+d$ на месте. Тем самым, $H$ собственная подгруппа в $S_n$. В действительности можно доказать, что порядок $H$ равен $d((n/d)!)^d$. Минимальный пример, когда $d \geq 2$ – это группа $\langle(1234), (13)\rangle \leq S_4$ порядок которой равен 8.

**5.8.** Докажите, что группа $S_n$ порождается транспозицией (12) и циклом $(23\ldots n)$.

## 6. Переборы с вариациями.[193]

При порождении перестановок в лексикографическом порядке каждая вторая получалась из предыдущей одной транспозицией. Но есть и такие, которые требуют $\lfloor (n+1)/2 \rfloor$ транспозиций. Между тем для многих приложений важно уметь порождать перестановки в таком порядке, что каждая следующая отличается от предыдущей одной *фундаментальной* транспозицией — такой порядок перестановок называется `minimum change order`.

Например, при умножении перестановки на фундаментальную транспозицию многие комбинаторные инварианты мало меняются, скажем увеличиваются или уменьшаются на 1. Поэтому на списке перестановок, записанном в `minimum change order`, многие комбинаторные алгоритмы работают значительно быстрее.

Исторически эта задача возникла в связи с вызваниванием **переборов с вариациями** = `change ringing`[194]. Чтобы понять, о чем идет речь, необходимо знать, что традиция английского колокольного звона радикально отличается как от русской, так и от континентальной европейской. В то время как на Руси звонарь ударяет в колокол раскачивая его *язык*, в Англии он поворачивает сам колокол при помощи специального колеса. Это значит, что на английских колоколах *невозможно* вызванивать сложные мелодии. Вызванивание переборов с вариациями начинается с нисходящей гаммы, в которой звучат все колокола начиная с самого маленького до самого большого. После этого вызваниваются всевозможные перестановки восьми колоколов — вариации = `changes`. Максимум того, что физически осуществимо в очередной вариации, это придержать *один* их колоколов с тем, чтобы переставить его с *соседним* колоколом по сравнению с предыдущей вариацией. Это и есть генерация перестановок в `minimum change order`.

---

[193] Если бы ты был младенцем с Криптона, то умел бы, к примеру, летать... — Я не раз летал, — с достоинством ответил Жихарь. — Я целых два раза летал посредством горячего воздуха. Правда, второй раз в полном беспамятстве... — Посредством горячего воздуха нынче только ленивый не летает, — сказал Колобок. — Нет, тебе бы полагалось лететь без пара и без крыльев, одною силою духа. ©Михаил Успенский, *Кого за смертью посылать*

[194] A. T. White, Ringing the cosets. — Amer. Math. Monthly, 1987, October, p.721–746.

Для примера перечислим элементы $S_3$ в обычном кампанологическом порядке, где принято гонять самый маленький колокол:

$$(1\,2\,3),\quad (2\,1\,3),\quad (2\,3\,1),\quad (3\,2\,1),\quad (3\,1\,2),\quad (1\,3\,2).$$

Обратите внимание, что здесь происходит. Вначале первый колокол перемещается на последнюю позицию — на кампанологическом языке это называется `hunting up`, после чего происходит перестановка двух других колоколов и первый колокол снова перемещается на первую позицию — `hunting down`.

При генерации $S_n$ первый колокол попеременно перемещается между первой и последней позицией $(n-1)!$ раз. Эту процедуру генерации перестановок называют **алгоритмом Джонсона–Троттера**[195]. Она реализована в функции `MinimumChangePermutations` пакета `Combinatorica`[196].

С точки зрения удобства программирования несколько естественнее гонять не первый, а последний колокол, иными словами порождать $S_3$ в следующем порядке:

$$(1\,2\,3),\quad (1\,3\,2),\quad (3\,1\,2),\quad (3\,2\,1),\quad (2\,3\,1),\quad (2\,1\,3).$$

**6.1.** Напишите рекуррентную программу, реализующую алгоритм Джонсона—Троттера в варианте с гоном последнего элемента.

**Ответ.** Вот вариант такой процедуры, реализованный с особым цинизмом. На *четных* перестановках $1,\ldots,n-1$ происходит гон $n$ вниз, а на *нечетных* — гон вверх. Вероятно, большинство пользователей вместо `Range[n,1,-1]` предпочли бы использовать `Range[n]` и заменили условие и порядок исходов в условном операторе на противоположные.

```
minperm[0]={{}}; minperm[n_]:=Flatten[Outer[
            If[Signature[#1]==1,
                  Insert[#1,n,#2],Insert[#1,n,n-#2+1]]&,
                        minperm[n-1],Range[n,1,-1],1],1]
```

**6.2.** То же, но с гоном первого элемента.

**6.3.** Напишите итеративную программу, реализующую алгоритм Джонсона—Троттера в варианте с гоном последнего элемента.

**Указание.** При желании можно написать примерно такую же процедуру, как та, которую мы использовали при построении перестановок в лексикографическом порядке.[197].

---

[195] S.M.Johnson, Generation of permutations by adjacent transpositions. – Math. Comput., 1963, vol.17, p.282–285.

[196] Начиная с версии 10, основная функциональность пакета Combinatorica включена в ядро `Mathematica`

[197] Детали изложены в упомянутой выше книге В. Липский. Комбинаторика для программистов – М:"Мир", 1988, 213 С.

**7. Декремент.**[198]

В этом разделе мы построим гомоморфизм $\operatorname{sgn} : S_n \longrightarrow \{\pm 1\}$. Обозначим через $m = |\underline{n}/\sigma|$ число орбит перестановки $\sigma \in S_n$. По определению $m = r + s$, где $r$ — количество истинных циклов перестановки $\sigma$, а $s = |\operatorname{Fix}(\sigma)|$. Разность $\operatorname{decr}(\sigma) = n - m$ называется **декрементом** перестановки $\sigma$. **Знаком перестановки** $\sigma \in S_n$ называется

$$\operatorname{sgn}(\sigma) = (-1)^{\operatorname{decr}(\sigma)} = (-1)^{n-m} = \prod_{i=1}^{m} (-1)^{|X_i|-1},$$

где произведение берется по **всем** орбитам $X_1, \ldots, X_m$ перестановки $\sigma$.

Достоинством этого определения является то, что, с одной стороны, легко доказать его совпадение с обычным определением в терминах транспозиций, к которому мы вернемся в следующих двух параграфах, а с другой стороны, так как знак определен в терминах самой перестановки $\sigma$, вопроса о корректности при этом не возникает. В следующих параграфах мы дадим еще два определения знака, но проверить их корректность и совпадение друг с другом заметно сложнее. В некоторых старых книгах знак перестановки назывался ее **сигнатурой**, в связи чем в `Mathematica` знак перестановки $x$ вычисляется посредством `Signature[x]`.

| `Signature[x]` знак перестановки $x$ |
|---|

**7.1.** Докажите, что $\tau_1, \ldots, \tau_l$ если транспозиции, то $\operatorname{sgn}(\tau_1 \ldots \tau_l) = (-1)^l$.

**Решение.** Индукция по $l$. Случай $l \leq 1$ очевиден. Для индукционного перехода достаточно показать, что знаки $\tau_2 \ldots \tau_l$ и $\tau_1 \ldots \tau_l$ **различны**. Мы покажем, что если $\pi$ — любая перестановка, а $\tau$ — транспозиция, то $\operatorname{sgn}(\tau\pi) = -\operatorname{sgn}(\pi)$. Достаточно показать, что число орбит изменяется на 1. Пусть $\tau = (pq)$. Орбиты $\pi$, не содержащие ни $p$, ни $q$, продолжают оставаться орбитами $\tau\pi$. Поэтому нам нужно рассмотреть следующие два случая: $p, q$ лежат в различных орбитах, $p, q$ лежат в одной орбите.

Если $p, q$ лежат в различных орбитах, то

$$(pq)(pi_2 \ldots i_r)(qj_2 \ldots j_s) = (pi_2 \ldots i_r qj_2 \ldots j_s)$$

так что в этом случае две орбиты сливаются в одну.

Если $p, q$ лежат в одной орбите, то

$$(pq)(pi_2 \ldots i_r qj_2 \ldots j_s) = (pi_2 \ldots i_r)(qj_2 \ldots j_s),$$

так что в этом случае одна орбита распадается на две.

Проанализировав это рассуждение, внимательный читатель может заметить, что оно доказывает *значительно* более точное утверждение.

---

[198] Все, что дается даром, наиболее ценно. Все наиболее ценное должно даваться даром. ©Поль Валери., *Тетради*

**7.2.** Пусть $d = \operatorname{decr}(\sigma)$. Докажите, что $\sigma$ можно представить как произведение $d$ транспозиций, причем $d$ наименьшее число, обладающее этим свойством.

Таким образом, декремент перестановки $\sigma$ есть *в точности* длина этой перестановки по отношению к множеству транспозиций, т. е. *наименьшее* число $d \in \mathbb{N}_0$ такое, что $\sigma$ можно представить в виде произведения $d$ транспозиций. Поскольку $m \geq 1$, декремент принимает наибольшее значение на множестве длинных циклов: длинные циклы и *только* они требуют для своего выражения $n-1$ транспозиции.

**8. Знакопеременная группа.**[199]

**8.1.** Докажите, что отображение $\operatorname{sgn} : S_n \longrightarrow \{\pm 1\}$ является гомоморфизмом.

**Решение.** Если перестановку $\pi$ можно представить как произведение $l$ транспозиций, а перестановку $\sigma$ — как произведения $m$ транспозиций, то, конкатенируя эти представления, мы выразим $\pi\sigma$ как произведение $l+m$ транспозиций. Таким образом,

$$\operatorname{sgn}(\pi\sigma) = (-1)^{l+m} = \operatorname{sgn}(\pi)\operatorname{sgn}(\sigma).$$

Назовем перестановку $\sigma$ **четной**, если $\operatorname{sgn}(\sigma) = 1$, и **нечетной**, если $\operatorname{sgn}(\sigma) = -1$. Например, $l$-цикл требует для своего выражения $l-1$ транспозиции и, тем самым, цикл четной длины *нечетен*, а цикл нечетной длины *четен*. Ядро sgn называется **знакопеременной группой** и обозначается $A_n$, от первой буквы слова `alternating`.

По определению $A_n$ состоит из всех четных перестановок. Множество $S_n \setminus A_n$ всех нечетных перестановок образует смежный класс $S_n$ по $A_n$ в качестве представителя которого можно выбрать, например, любую транспозицию:

$$S_n = A_n \sqcup (S_n \setminus A_n) = A_n \sqcup A_n(12).$$

Для любого $n \geq 2$ группа $A_n$ является подгруппой индекса 2 в $S_n$ и, значит $A_n \trianglelefteq S_n$. Порядок группы $A_n$ равен $n!/2$.

**8.2.** Постройте вложение $S_n$ в $A_{n+2}$. Покажите, что $S_n$ нельзя вложить в $A_{n+1}$.

По определению группа $A_n$ порождается всевозможными произведениями двух различных транспозиций $(ij)(hk)$. Однако обычно удобнее пользоваться другой системой образующих.

**8.3.** При любом $n \geq 3$ группа $A_n$ порождается 3-циклами.

---

[199] При любом числе колоколов ровно половина всех вариаций имеет одну природу и ровно половина — другую. В чем именно заключается эта природа, мне не дано судить, но, как станет мало-помалу ясно, в ней непременно следует разобраться, прежде чем мы сможем постичь науку композиции звонов и их исполнения. ©C.A.W.Troyte, *Change Ringing*

**Решение.** Для доказательства теоремы нам нужно выразить произведение двух транспозиций $(ij)(hk)$ как произведение 3-циклов. Если оказывается $|\{i,j,h,k\}|=3$, то это произведение само является 3-циклом. С другой стороны, если $|\{i,j,h,k\}|=4$, то $(ij)(hk)=(ij)(ih)(ih)(hk)$ является произведением двух 3-циклов.

**8.4.** Докажите, что группа $A_n$ порождается 3-циклами $(123)$, $(124)$, $\ldots$, $(12n)$.

**8.5.** Докажите, что группа $A_n$ порождается 3-циклами $(123)$, $(234)$, $\ldots$, $(n-2,n-1,n)$.

**Указание.** Группа $A_n$ порождается подгруппой $A_{n-1}$, стабилизирующей $n$, и одним (любым) циклом, перемещающим $n$.

**8.6.** Пусть $n$ нечетно, верно ли, что группа $A_n$ порождается 3-циклами $(123)$, $(145)$, $\ldots$, $(1,n-1,n)$?

**8.7.** Докажите, что группа $A_n$ порождается 3-циклом $(123)$, и либо длинным циклом $(12\ldots n)$ при нечетном $n$, либо циклом $(23\ldots n)$ при четном.

**8.8.** Покажите, что при $n\geq 4$ центр группы $A_n$ тривиален.

**Решение.** Для любого $\pi\neq \mathrm{id}$ найдется $i$ такое, что $\pi(i)\neq i$. Так как $n\geq 4$, то найдутся $j,h$ такие, что все четыре индекса $i,\pi(i),j,h$ различны. Тогда $\pi(ijh)\pi^{-1}=(\pi(i),\pi(j),\pi(h))\neq(i,j,h)$.

**8.9.** Докажите, что при $n\geq 5$ все 3-циклы в $A_n$ сопряжены. Верно ли это для $n=4$?

**Решение.** Мы уже знаем, что 3-циклы сопряжены в $S_n$. Пусть $(ijh)$ и $(klm)$ — два любых 3-цикла, а $\pi\in S_n$ – перестановка такая, что $\pi(ijh)\pi^{-1}=(klm)$. Если перестановка $\pi$ четна, мы достигли своей цели. Если перестановка $\pi$ нечетна, но $n\geq 5$, то найдутся такие $r,s$, что все 5 индексов $i,j,h,r,s$ различны. Тогда $(rs)$ коммутирует с $(ijh)$ и, тем самым, $\pi(rs)(ijh)(\pi(rs))^{-1}=(klm)$, причем $\pi(rs)$ четна. С другой стороны, централизатор 3-цикла содержит по крайней мере 3 элемента, поэтому в $A_4$ имеется не более $12/3=4$ циклов, сопряженных с данным. Это значит, что 3-циклы в $A_4$ разбиваются на два класса сопряженности.

## 9. Инверсии.[200]

Забудем про определение из § 9 и дадим другое определение знака. Положим $\mathrm{sgn}(\sigma)=(-1)^l$, если $\sigma$ можно представить как произведение $l$ транспозиций $\sigma=\tau_1\ldots\tau_l$. Это определение эквивалентно нашему основному определению через декремент. Однако, из этого нового определения совершенно неясно, почему знак определен **корректно**, т. е. почему перестановку $\sigma$ нельзя представить в виде

$$\sigma=\tau_1\ldots\tau_l=\rho_1\ldots\rho_m,$$

[200]Мне, конечно, легче сойти с ума, чем им. Я, например, увижу на карте Пакистана: там, где должен быть Исламабад — там оказалось Равалпинди, а там, где прежде было Равалпинди, увижу Исламабад — и все, я сбрендил. А они все даже не заметят. ©Венедикт Ерофеев, *Из записных книжек*

где $l$ и $m$ имеют разную четность? Это можно доказать методом совместной индукции[201], но мы не будем обсуждать доказательство здесь. Вместо этого покажем, что знак вполне характеризуется тем, что это гомоморфизм, переводящий транспозиции в $-1$.

**9.1.** Для любого $n \geq 2$ знак sgn является *единственным* нетривиальным гомоморфизмом $\phi : S_n \longrightarrow \{\pm 1\}$.

**Решение.** В самом деле, пусть $(pq)$, $(rs)$ — две транспозиции в $S_n$, а $\pi$ — любая перестановка такая, что $\pi(p) = r$, $\pi(q) = s$. Тогда $\pi(pq)\pi^{-1} = (rs)$, так что при всех гомоморфизмах $\phi : S_n \longrightarrow \{\pm 1\}$ в абелеву группу $\{\pm 1\}$ транспозиции принимают одно и то же значение. Если это значение равно 1, то гомоморфизм $\phi$ тривиален. Если же оно равно $-1$, то $\phi = \mathrm{sgn}$.

**9.2.** При $n \geq 2$ знакопеременная группа $A_n$ является единственной подгруппой индекса 2 в $S_n$.

Сейчас мы дадим *третье* определение знака перестановки, в терминах длины этой перестановки по отношению к множеству фундаментальных транспозиций. Говорят, что пара $(i, j)$ образует **инверсию** = `inversion` = `Fehlstand` для перестановки $\sigma \in S_n$, если $i < j$, но $\sigma(i) > \sigma(j)$. Обозначим через $\mathrm{inv}(\sigma)$ общее **число инверсий** перестановки $\sigma$, т. е. количество всех пар $(i, j)$, $1 \leq i < j \leq n$, образующих инверсию для $\sigma$.

Заметим, что многие авторы называют инверсией не саму пару $(i, j)$, а ее *образ* $(\sigma(i), \sigma(j))$ под действием $\sigma$. Кнут утверждает, что понятие инверсии ввел в 1750 году Крамер в книге *Introduction à l'analyse des lignes courbes algébriques*, но, конечно, трудно себе представить, чтобы Секи Кова и фон Лейбниц не владели этим понятием лет за 70 до Крамера, когда они одновременно — и, насколько мы в состоянии судить, независимо — ввели понятие определителя. Однако в шпенглеровском смысле понятие инверсии гораздо старше и уже абсолютно отчетливо выступает в китайских текстах III в. до н.э.

**9.3.** Докажите, что для любой перестановки $\mathrm{sgn}(\sigma) = (-1)^{\mathrm{inv}(\sigma)}$

**Решение.** Каждая транспозиция есть произведение нечетного числа фундаментальных транспозиций. Умножение на фундаментальную транспозицию создает или убивает ровно одну инверсию.

В действительности, имеет место *гораздо* более точное утверждение.

**9.4.** Пусть $d = \mathrm{inv}(\sigma)$. Докажите, что $\sigma$ можно представить как произведение $d$ фундаментальных транспозиций, причем $d$ *наименьшее* число, обладающее этим свойством.

Теперь мы в состоянии дать еще одно определение знака перестановки — можно думать, что это *шутка*, но в действительности, это *цитата* взято из учебника `высшей алгебры` Куроша:

$$\mathrm{sgn}(\sigma) = \prod_{1 \leq i < j \leq n} \frac{\sigma(i) - \sigma(j)}{i - j}.$$

[201] А.И.Кострикин, Введение в алгебру, ч.I: Основы алгебры. – М.: Наука, 1994

При всей своей неестественности[202] это определение обладает одним техническим преимуществом, а именно, для этого определения очевидно, что sgn является гомоморфизмом. Для этого нужно лишь заметить, что в действительности произведение в этой формуле берется по $\{i,j\} \in \bigwedge^2(\underline{n})$. В частности, для любой перестановки $\pi$ имеем

$$\prod_{1\le i<j\le n} \frac{\sigma(i)-\sigma(j)}{i-j} = \prod_{1\le \pi(i)<\pi(j)\le n} \frac{\sigma(i)-\sigma(j)}{i-j}$$

Это значит, что следующее свойство получается даром.

**9.5.** Отображение $(-1)^{\mathrm{inv}} : S_n \longrightarrow \{\pm 1\}$, $\pi \mapsto (-1)^{\mathrm{inv}(\pi)}$, является гомоморфизмом.

**Решение.** В самом деле,

$$(-1)^{\mathrm{inv}(\pi\sigma)} = \prod_{1\le i<j\le n} \frac{\pi\sigma(i)-\pi\sigma(j)}{i-j} =$$
$$= \prod_{1\le i<j\le n} \frac{\pi\sigma(i)-\pi\sigma(j)}{\sigma(i)-\sigma(j)} \prod_{1\le i<j\le n} \frac{\sigma(i)-\sigma(j)}{i-j} = (-1)^{\mathrm{inv}(\pi)}(-1)^{\mathrm{inv}(\sigma)}.$$

**9.6.** Дайте еще одно доказательство того, что для любого $n \ge 2$ знак sgn является *единственным* нетривиальным гомоморфизмом $\phi : S_n \longrightarrow \{\pm 1\}$.

**Решение.** Как мы только что показали, отображение $(-1)^{\mathrm{inv}}$ является гомоморфизмом $S_n \longrightarrow \{\pm 1\}$. Этот гомоморфизм нетривиален, так как, например, у транспозиции $\tau = (1,2)$ всего одна инверсия, и, следовательно, $(-1)^{\mathrm{inv}(\tau)} = -1$. В силу единственности знака гомоморфизм $(-1)^{\mathrm{inv}}$ *обязан* совпадать с sgn.

### 10. Принцип инволюций.[203]

Перестановка называется **инволюцией**, если ее квадрат равен id. Между большинством математиков и специалистами по теории групп существует расхождение в понимании того, является ли сама тождественная перестановка id инволюцией. Большинство математиков считают ее тривиальной инволюцией. В то же время в теории групп инволюцией называется только элемент порядка 2.

---

[202] Эпитет **неестественность** относится не к выражению $(-1)^{\mathrm{inv}(\sigma)}$ как таковому, **наоборот**, с точки зрения теории групп Вейля $\mathrm{inv}(\sigma)$ совпадает с $n(\sigma)$, числом положительных корней, которые становятся отрицательными под действием $\sigma$ и, тем самым, с $l(\sigma)$ — длиной $\sigma$ в Коксетеровских образующих. Нет ничего естественнее **этого** определения! Однако выражать $(-1)^{\mathrm{inv}(\sigma)}$ как дурацкое произведение!!! Такое определение было бы уместно в учебнике математического анализа, как часть общей **перверсивно-декадентской** парадигмы. Но цель курса алгебры прямо противоположна — культивировать радость, силу, вкус и привычку к естественности!

[203] Решением Комиссии по переименованиям астероиду номер 35627 возвращено историческое название “астероид номер 27635”. ©ВНиколай Фоменко

**10.1.** Напишите программу, порождающую все инволюции.

**Указание.** Каждая инволюция $\underline{n}$ либо оставляет $n$ на месте и тогда она получается из инволюции $\underline{n-1}$, либо она переставляет $n$ с каким-то $i$, $1 \leq i \leq n-1$, и тогда она получается из такой инволюции $\underline{n-1}$, которая оставляет на месте $i$.

**10.2.** Вычислите количество инволюций в симметрической группе $S_n$.

Под действием инволюции $\pi$ множество $X$ разбивается на *одноэлементные* орбиты $\{x\}$, отвечающие **инвариантным элементам** $x = \pi x$, и двухэлементные орбиты $\{x, y\}$, отвечающие **энантиоморфным парам** $x, y$, где $y = \pi x \neq x$. Обозначим через $X^\pi$ множество неподвижных точек. Только что сказанное означает, что если $\pi, \sigma$ — любые две инволюции на конечном множестве $X$, то порядки множеств $X^\pi$ и $X^\sigma$ сравнимы по модулю 2. В частности, если инволюция $\pi$ имеет на $X$ *нечетное* количество неподвижных точек, то и любая другая инволюция $\sigma$ имеет на $X$ по крайней мере одну неподвижную точку. Это несложное соображение, называемое **принципом инволюций**, имеет далеко идущие следствия.

Стандартное доказательство теоремы Лагранжа о том, что любое целое представляется в виде суммы четырех квадратов, приводимое в любом учебнике теории чисел, состоит из двух частей:

- тождества Эйлера, сводящего все к представимости простого числа,
- (доказанной Эйлером) теоремы Ферма о представлении простых чисел в виде суммы двух квадратов.

При этом обычно считается, что именно вторая часть требует некоторых усилий, скажем построения арифметики целых гауссовых чисел. Surprise!!!

**10.3.** Докажите следующую теорему Ферма: любое простое $p \equiv 1 \pmod 4$ представимо в виде суммы двух квадратов. Иными словами, найдутся такие $m, n \in \mathbb{N}$, что $p = m^2 + n^2$.

**Решение.** Приведем совершенно замечательное доказательство Цагира, основанное на принипе инволюций[204]. Рассмотрим множество

$$X = \{(x, y, z) \mid x, y, z \in \mathbb{N}, x^2 + 4yz = p\}.$$

Легко видеть, что отображение $h : X \longrightarrow X$ заданное посредством

$$(x, y, z) \mapsto \begin{cases} (x + 2z, z, y - x - z), & \text{если } x < y - z, \\ (2y - x, y, x - y + z), & \text{если } y - z < x < 2y, \\ (x - 2y, x - y + z, y), & \text{если } 2y < x, \end{cases}$$

является инволюцией (проверьте!). Ясно, что у этой инволюции ровно одна неподвижная точка на $X$, а именно, $(1, 1, l)$, где $p = 4l + 1$. По принципу инволюций тогда инволюция $g : X \longrightarrow X$, $(x, y, z) \mapsto (x, z, y)$, тоже имеет

[204] D.Zagier, A one-sentence proof that every prime $p \equiv 1 \pmod 4$ is a sum of two squares. — Amer. Math. Monthly, 1990, vol.97, N.2, p.144.

по крайней мере одну неподвижную точку $(m, n, n)$, так что $p = m^2 + 4n^2$, как и утверждалось.

**11. Беспорядки.**

Перестановка $\pi$ называется **беспорядком** = `derangement`, если она не оставляет на месте ни одного символа, иными словами, если $\pi(i) \neq i$ для всех $i$. Задача вычисления количества беспорядков $D_n$ называется **задачей Монмора**. Вот ее простой частный случай.

**11.1.** Сколькими способами можно расставить шахматной доске 8 ладей так, чтобы ни одна из них не била другие и чтобы ни одна не стояла на главной диагонали a8–h1?

**11.2.** Определите функцию, выясняющую, является ли данная перестановка беспорядком.

**Ответ.** Например, вот как это сделано в пакете `Combinatorica`:

```
DerangementQ[x_]:=FreeQ[Range[Length[x]]-x,0]
```

**11.3.** Напишите программу, порождающую все беспорядки степени $n$.

**11.4.** Проведя численный эксперимент, постарайтесь угадать, к чему стремится отношение количества беспорядков порядка $n$ к общему количеству всех перестановок степени $n$ при росте $n$.

**Ответ.** Ответ совершенно удивителен, это отношение чрезвычайно быстро стремится к $1/e$. С точки зрения всех практических целей уже для небольших значений $n$ (порядка нескольких десятков) вероятность того, что случайно выбранная перестановка является беспорядком, не зависит от $n$. Например, если пальто в гардеробе выдаются случайным образом, то вероятность того, что уходя из театра *каждый* зритель возьмет чужое пальто, не зависит от количества зрителей.

А теперь объясним получившийся ответ.

**11.5.** Вычислите количество $D_n$ беспорядков в общем случае.

**Ответ.** По формуле решета получаем, что

$$D_n = n! \left(1 - \frac{1}{1!} + \frac{1}{2!} - \ldots + (-1)^n \frac{1}{n!}\right).$$

Интересно, что формула в скобках – это начало разложения в ряд для $e^{-1}$. Таким образом, для не слишком маленьких $n$ число $e^{-1} > 1/3$ является **очень** хорошим приближением для отношения $D_n/n!$, т.е. вероятности того, что наугад взятая перестановка $n$ символов не оставляет на месте ни одного символа. Совершенно удивительно, что эта вероятность практически не зависит от $n$! Как замечает по этому поводу Райзер, для всех практических целей вероятность *равна* $e^{-1}$ = *для гренладских китов* $\pi = 3$.

В действительности, совсем легко указать и явную рекуррентную формулу для определения количества беспорядков степени $n$. Следующее про-

стое рассуждение взято из книги[205]. Пусть $D_n$ есть количество беспорядков степени $n$.

• Пусть $\pi$ — беспорядок степени $n-1$. Умножая $\pi$ на транспозицию $(i,n)$, для какого-то $1 \le i \le n-1$, мы получим беспорядок степени $n$. Эта процедура даст $(n-1)D_{n-1}$ беспорядков степени $n$.

• Однако, не каждый беспорядок степени $n$ получается таким образом. А именно, пусть $\pi$ перестановка степени $n-1$, оставляющая на месте *ровно один* элемент, а именно, $i$, $1 \le i \le n-1$. Тогда $\pi(i,n)$ также является беспорядком степени $n$.

Ясно, что эти две процедуры дадут нам все беспорядки степени $n$. А именно, для любого беспорядка степени $n$ найдется такое $i$, $1 \le i \le n-1$, что $\pi(i,n)$ оставляет на месте $n$. Но тогда ограничение $\pi(i,n)$ на первые $n-1$ символ *либо* является беспорядком, *либо* имеет единственную неподвижную точку $i$, причем эти возможности являются взаимоисключающими.

**11.6.** Напишите программу, вычисляющую количество беспорядков степени $n$, и сравните ее с `Round[N[n!/E]]`.

**Ответ.** Вот непосредственная реализация описанного выше алгоритма:

```
derang[1]=0; derang[2]=1;
derang[n_]:=derang[n]=(n-1)*derang[n-1]+(n-1)*derang[n-2]
```

**11.7.** Сколько перестановок из $S_n$ оставляют на месте ровно $m$ символов?

**11.8.** Напишите программу, порождающую все перестановки степени $n$, имеющие ровно $m$ неподвижных точек.

**Указание.** Вначале породите все перестановки, оставляющие на месте точки $1,\dots,m$ и только их.

The game never ends, but not the players.

## § 5. Циклы

Наша стратегия состоит в том, чтобы одному биться против десяти, наша тактика — в том, чтобы десяти биться против одного.

Мао Цзе-Дун

### 1. Циклы.[206]

Фиксируем какую-то перестановку $\pi \in S_n$. Определим на множестве $\underline{n} = \{1,\dots,n\}$ отношение $\sim$, полагая $i \sim j$, если $j = \pi^k(i)$ для $k \in \mathbb{Z}$. Очевидно, что отношение $\sim$ является эквивалентностью. Иными словами, оно

• рефлексивно: $i = \mathrm{id}(i) = \pi^0(i)$;

• симметрично: если $j = \pi^k(i)$, то $i = \pi^{-k}(j)$;

[205] J.L.Snell, Introduction to probability theory with computing. — Prentice Hall, 1975.

[206] Revolution is the opiate of the intellectuals. ©Karl Marx

• транзитивно: если $j = \pi^k(i)$ и $h = \pi^l(j)$, то $h = \pi^{k+l}(i)$.

Таким образом, с каждой перестановкой $\pi$ связано разбиение множества $\underline{n}$ на классы эквивалентности $\sim$, эти классы называются **орбитами** $\pi$. Тем самым, $\underline{n} = X_1 \sqcup \ldots \sqcup X_m$, где $X_1, \ldots, X_m$ суть орбиты перестановки $\pi$. Иногда множество орбит перестановки $\sigma$ обозначается через $\underline{n}/\sigma = \{X_1, \ldots, X_m\}$. Орбиты, содержащие более одного элемента, будут называться **нетривиальными**.

С каждой перестановкой $\pi \in S_n$ можно связать два подмножества в $\underline{n}$, а именно, множества

$$\mathrm{Fix}(\pi) = \{i \in \underline{n} \mid \pi(i) = i\}, \qquad \mathrm{Mob}(\pi) = \{i \in \underline{n} \mid \pi(i) \neq i\}$$

**неподвижных** alias **стабильных** точек и **подвижных** alias **мобильных** точек. Иными словами, $\mathrm{Fix}(\pi)$ является объединением всех одноэлементных орбит, в то время как $\mathrm{Mob}(\pi)$ является объединением всех нетривиальных орбит. Ясно, что $\mathrm{Fix}(\pi)$ и $\mathrm{Mob}(\pi)$ устойчивы под действием $\pi$, причем $\underline{n} = \mathrm{Fix}(\pi) \sqcup \mathrm{Mob}(\pi)$.

Перестановки $\pi$ и $\sigma$ называются **независимыми**, если

$$\mathrm{Mob}(\pi) \cap \mathrm{Mob}(\sigma) = \emptyset.$$

Перестановка $\sigma \in S_n$ называется **циклом**, если множество ее мобильных элементов представляет собой одну орбиту под действием $\sigma$. В этом случае $\mathrm{Mob}(\sigma)$ часто называется также **носителем** цикла $\sigma$, а порядок $|\mathrm{Mob}(\sigma)|$ — его **длиной**. Циклы длины 1 называются тривиальными, цикл длины $\geq 2$ называется **истинным циклом**. Обычно, говоря о циклах, математики имеют в виду *истинные* циклы, однако, из программистских соображений нам будет удобнее считать id циклом и писать $\mathrm{id} = (1) = (2) = \ldots = (n)$.

Циклы настолько часто используются в вычислениях, что нам будет удобно ввести для них специальные обозначения. Пусть $i_1, \ldots, i_l \in \underline{n}$ — набор попарно различных символов. Тогда через $(i_1 \ldots i_l)$, обозначается цикл длины $l$ с носителем $\{i_1, \ldots, i_l\}$, под действием которого

$$i_1 \mapsto i_2 \mapsto i_3 \mapsto \ldots \mapsto i_l \mapsto i_1,$$

а все остальные элементы множества $\underline{n}$ остаются на месте. Один и тот же цикл может быть записан по разному:

$$(i_1 i_2 \ldots i_l) = (i_2 \ldots i_l i_1) = \ldots = (i_l i_1 \ldots i_{l-1}).$$

Обычно для записи циклов мы будем пользоваться именно такой записью, которая называется **циклической**.

**1.1.** Напишите программу, возвращающую запись цикла $(i_1 i_2 \ldots i_l)$ как перестановки степени $n$.

**Ответ.** В пакете `Combinatorica` это делается при помощи функции

```
PermutationWithCycle[n_,l_]:=
        FromCycles[Append[({#}&) /@ Complement[Range[n],l],l]]
```

**1.2.** Вычислите обратный к циклу.

**Ответ.** Ясно, что обратный к циклу $(i_1 \dots i_l)$ равен $(i_l \dots i_1)$, так что для нахождения обратного достаточно применить к циклической записи функцию `Reverse`.

**2. Длинные циклы.**

Перестановка $\sigma$ называется **длинным циклом**, если все множество $\underline{n}$ образует одну орбиту под действием $\sigma$. Длинные циклы, называемые также **элементами Коксетера**, представляют собой один из наиболее интересных типов перестановок. Особенно часто используются следующие два взаимнообратных длинных цикла:

$$\texttt{RotateRight} = (1\,2\,3\,\dots\,n), \qquad \texttt{RotateLeft} = (n\;n-1\;n-2\;\dots\;1).$$

**2.1.** Найдите количество длинных циклов в $S_n$.

**Решение.** Запись длинного цикла имеет вид $(\pi(1), \dots, \pi(n))$ для некоторого $\pi \in S_n$. Таким образом, количество различных *записей* длинных циклов равно $n!$. Однако, как было уже замечено, применение `RotateRight` к записи длинного цикла не меняет этот цикл. Так как порядок `RotateRight` равен $n$, то любой длинный цикл допускает ровно $n$ различных записей. Это значит, что количество $n$-циклов на $n$-элементном множестве равно $n!/n = (n-1)!$.

**2.2.** Напишите программу, генерирующую все длинные циклы степени $n$ в циклической записи.

**Ответ.** Среди всех $n$ циклических записей такого цикла ровно одна начинается с 1. Поэтому грубое решение состоит в том, чтобы выбрать те перестановки степени $n$, сокращенная запись которых начинается с 1 и интерпретировать их как запись цикла длины $n$. Для чуть более экономного решения можно записать все перестановки степени $n-1$ и приаппендить к ним $n$. Либо, еще лучше, взять все перестановки $2, \dots, n$ и приписать к ним 1 слева:

```
Map[Prepend[#,1]&,Permutations[Range[2,n]]]
```

Вот, например, все длинные циклы степени 4 в циклической записи:

$$(1\,2\,3\,4)\quad(1\,2\,4\,3)\quad(1\,3\,2\,4)\quad(1\,3\,4\,2)\quad(1\,4\,2\,3)\quad(1\,4\,3\,2).$$

**2.3.** Напишите программу, осуществляющую переход от циклической записи длинного цикла к обычной.

**Решение.** Перейти от циклической записи к полной записи совсем просто. Нужно лишь приписать к циклу $x$ тот же цикл, сдвинутый на одну позицию влево {`x,RotateLeft[x]`}. Можно поступить и наоборот, а именно, предварить цикл тем же циклом, сдвинутым на одну позицию вправо,

{RotateRight[x],x}. Теперь обычная конструкция с Transpose и Sort позволяет найти сокращенную запись.

**2.4.** Напишите программу, генерирующую все длинные циклы степени $n$ в сокращенной записи.

**Ответ.** Нужно лишь скомбинировать результаты двух предыдущих задач посредством Map. Например, так

```
long[n_]:=Map[Last[Transpose[
                        Sort[Transpose[{#,RotateLeft[#]}]]]]&,
                  Map[Prepend[#,1]&,Permutations[Range[2,n]]]]
```

Вот, например, все длинные циклы степени 4 в сокращенной записи:

$$(2\,3\,4\,1)\quad(2\,4\,1\,3)\quad(3\,4\,2\,1)\quad(3\,1\,4\,2)\quad(4\,3\,1\,2)\quad(4\,1\,2\,3).$$

### 3. Каноническое разложение на циклы.[207]

В соответствии с общим определением два цикла называются **независимыми**, если их носители дизъюнктны. Иными словами, циклы $(i_1,\dots,i_l)$ и $(j_1,\dots,j_m)$ независимы, если $\{i_1,\dots,i_l\}\cap\{j_1,\dots,j_m\}=\emptyset$.

Следующий результат, известный как **разложение на независимые циклы**, лежит в основе всей теории групп перестановок. Каждую перестановку можно представить как произведение попарно независимых истинных циклов. Такое представление единственно *с точностью до перестановки сомножителей*.

Следующий способ позволяет сделать разложение на циклы *единственным*. А именно, обозначим через $\pi_1$ цикл, содержащий наименьший элемент из $\mathrm{Mob}(\pi)$, через $\pi_2$ – цикл, содержащий наименьший элемент из $\mathrm{Mob}(\pi)$, который не попал в $\mathrm{Mob}(\pi_1)$, через $\pi_3$ — цикл, содержащий наименьший элемент из $\mathrm{Mob}(\pi)$, который не попал в $\mathrm{Mob}(\pi_1)\cup\mathrm{Mob}(\pi_2)$, и т. д. Полученное в результате разложение называется **каноническим разложением $\pi$ на циклы**.

Для того, чтобы единственным было не только само разложение на независимые циклы, но и его *запись*, математики обычно начинают запись каждого цикла с *наименьшего* элемента. Например, при этом соглашении

$$\begin{pmatrix}1&2&3&4&5&6&7&8&9\\3&5&1&8&9&4&7&6&2\end{pmatrix}=(13)(259)(486)$$

При вычислениях вручную тривиальные циклы обычно опускаются, но при реализации эффективных компьютерных алгоритмов часто естественно явно выписывать и все тривиальные циклы и мы обычно так и будем поступать. Например, к приведенному выше разложению следовало бы дописать тривиальный цикл (7).

---

[207] Это вам так показалось. Ведь я знаю, что они на рынке покупают. Купит вон тот каналья повар, что выучился у француза, кота, обдерет его, да и подает на стол вместо зайца. ©Николай Гоголь, *Мертвые души*

Сейчас мы определим функции, восстанавливающие по перестановке ее каноническое разложение и по каноническому разложению исходную перестановку в обычной записи.

**3.1.** Напишите программу, возвращающую по перестановке ее каноническое разложение на циклы.

**Решение 1 (функциональное программирование).** Следующий текст является вариацией на тему определения команды `ToCycles`, определенной в пакетах `SymmetricGroup` и `Combinatorica`. Вот как вычисляется цикленный тип перестановки $x$ степени $n$:

```
toCycles[x_]:=Block[{i},Fold[If[MemberQ[Flatten[#1],#2],#1,
        Append[#1,NestWhileList[x[[#]]&,i=#2,x[[#]]!=i&]]]&,
                {},Range[Length[x]]]
```

Эта программа работает следующим образом. Два слота `#1` и `#2` заполняются соответственно

- списком циклов, начиная с пустого списка циклов {},
- текущим номером $i$ от 1 до $n$.

Если индекс $i$ использован в каком то из уже образованных циклов, программа не меняет список циклов. В противном случае она начинает образовывать список, состоящий из образов $i$ под действием перестановки $x$ до тех пор, пока снова не получится $i$. В этот момент она прекращает выполнение команды `NestWhileList` и добавляет получившуюся орбиту $i$ к списку циклов.

**Решение 2 (процедурное программирование).** Для циклоидов приведем процедурную программу, решающую ту же задачу:

```
toCycles[x_]:=Block[{n=Length[x],list={},cycle,test,i,j},
                                    test=Table[True,{n}];
          For[i=1,i<=n,i++,If[test[[i]],For[j=i;cycle={},
                    test[[j]],AppendTo[cycle,j];j=x[[j]],
                                    test[[j]]=False];
          AppendTo[list,cycle]]];Return[list]]
```

Здесь происходит следующее. Прежде всего, мы вводим шесть локальных переменных — степень $n$ перестановки $x$, список циклов `list`, текущий цикл `cycle`, тест свежести индекса `test`, начало текущего цикла $i$ и текущий индекс в текущем цикле $j$. Тест `test[[j]]` дает значение `True`, если индекс $j$ свежий, иными словами, еще не учтен ни в перечисленных ранее циклах, ни в составе текущего цикла. В противном случае тест `test[[j]]` дает значение `False`, в этом случае $j$ не может ни служить началом нового цикла, ни аппендиться к текущему циклу. Изначально ни один индекс не учтен, поэтому первое присваивание в теле блока говорит, что *все* индексы свежие. Далее, в первом цикле мы по порядку перебираем все индексы $i$ от 1 до $n$ и, если индекс $i$ свежий, мы начинаем с него новый текущий цикл `cycle={}`, текущий элемент которого обозначается $j$. До тех пор, пока

элемент $j$ является свежим, мы добавляем его к циклу `cycle`, делаем его несвежим и присваиваем $j$ следующее значение `j=x[[j]]`.

В качестве примера приведем вычисленный при помощи этой функции канонические разложения перестановок степени 4.

| | | |
|---|---|---|
| $(1\,2\,3\,4) = (1)(2)(3)(4)$ | $(1\,2\,4\,3) = (1)(2)(3\,4)$ | $(1\,3\,2\,4) = (1)(2\,3)(4)$ |
| $(1\,3\,4\,2) = (1)(2\,3\,4)$ | $(1\,4\,2\,3) = (1)(2\,4\,3)$ | $(1\,4\,3\,2) = (1)(2\,4)(3)$ |
| $(2\,1\,3\,4) = (1\,2)(3)(4)$ | $(2\,1\,4\,3) = (1\,2)(3\,4)$ | $(2\,3\,1\,4) = (1\,2\,3)(4)$ |
| $(2\,3\,4\,1) = (1\,2\,3\,4)$ | $(2\,4\,1\,3) = (1\,2\,4\,3)$ | $(2\,4\,3\,1) = (1\,2\,4)(3)$ |
| $(3\,1\,2\,4) = (1\,3\,2)(4)$ | $(3\,1\,4\,2) = (1\,3\,4\,2)$ | $(3\,2\,1\,4) = (1\,3)(2)(4)$ |
| $(3\,2\,4\,1) = (1\,3\,4)(2)$ | $(3\,4\,1\,2) = (1\,3)(2\,4)$ | $(3\,4\,2\,1) = (1\,3\,2\,4)$ |
| $(4\,1\,2\,3) = (1\,4\,3\,2)$ | $(4\,1\,3\,2) = (1\,4\,2)(3)$ | $(4\,2\,1\,3) = (1\,4\,3)(2)$ |
| $(4\,2\,3\,1) = (1\,4)(2)(3)$ | $(4\,3\,1\,2) = (1\,4\,2\,3)$ | $(4\,3\,2\,1) = (1\,4)(2\,3)$ |

Обратите внимание, что запись $(ijhk)$ в левой и в правой частях имеет совершенно разный смысл. Слева это приведенная запись перестановки, а справа — ее каноническое разложение на циклы. Иными словами, в левой части $(1\,2\,3\,4)$ обозначает тождественную перестановку, а в правой части $(1\,2\,3\,4)$ обозначает длинный цикл $1 \mapsto 2 \mapsto 3 \mapsto 4 \mapsto 1$.

**3.2.** Напишите программу, возвращающую по каноническому разложению перестановки на циклы ее полную запись.

**Решение.** Ну, это совсем просто, из предыдущего параграфа мы знаем, как превратить в полную запись один цикл, теперь мы должны просто сделать то же самое для всех циклов, входящих в цикленную запись $z$ перестановки, а потом убрать лишний уровень вложенности:

```
{Flatten[z],Flatten[Map[RotateLeft,z]]}
```

или, что то же самое,

```
{Flatten[Map[RotateRight,z]],Flatten[z]}
```

**3.3.** Напишите программу, возвращающую по каноническому разложению перестановки на циклы ее сокращенную запись.

**Ответ.** Нужно просто совместить функцию, построенную в предыдущей задаче, с функцией перехода от полной записи к сокращенной. Например, так

```
fromCycles[z]:=Last[Transpose[Sort[Transpose[
                  {Flatten[z],Flatten[Map[RotateLeft,z]]}]]]]
```

Именно так, по существу, и устроена функция `FromCycles`, определенная в пакете `Combinatorica`.

Сейчас мы предложим задачу, которая объясняет, почему программисты называют каноническим разложением перестановки на циклы *реверсированное* каноническое разложение. Рассмотрим отображение $f$ сопоставляющее перестановке $\pi$ перестановку, которая получается из нее следующим образом: мы записываем каноническое разложение $\pi$ на циклы, а потом стираем все внутренние скобки в получившемся разложении и интерпретируем его как сокращенную запись перестановки $f(\pi)$. Например, стирая

внутренние скобки в канонических разложениях (1)(2)(3), (12)(3), (1)(23) (123) мы *каждый раз* получим (123), а стирая внутренние скобки в (13)(2) и (132), мы получим перестановку (132), каноническое разложение которой равно (1)(23).

**3.4.** Изучите динамику функции $f$.

**Ответ.** Можно, конечно, провести компьютерный эксперимент применив `Flatten[toCycles[x]]` ко всем перестановкам небольших степеней, но каждому и так ясно, что однократное применение функции $f$ к *любой* перестановке приводит к перестановке, для которой 1 является неподвижной точке. Тем самым, ее двухкратное применение приводит к перестановке, для которой неподвижны 1 и 2, etc. Это значит, что не более чем за $n-1$ шагов от каждой перестановки мы дойдем до тождественной перестановки, которая является *единственной* неподвижной точкой функции $f$.

**3.5.** Докажите, что сушествует перестановка, от которой применением $f$ нельзя дойти до тождественной перестановки меньше, чем за $n-1$ шаг. Опишите все такие перестановки.

**Ответ.** Имеется ровно $(n-1)!$ такая перестановка степени $n$, а именно, это все перестановки $\pi$ такие, что $\pi(1) = n$. В самом деле, разложение такой перестановки на циклы начинается с $(1n\dots)$ и, значит, $f(\pi) = (1n\dots)$. Иными словами, разложение $f(\pi)$ на циклы начинается с $(1)(2n\dots)$ и, значит, $f^2(\pi) = (12n\dots)$. При каждом дальнейшем применении $f$ индекс $n$ будет сдвигаться вправо на одну позицию.

## 4. Реверсированное каноническое разложение на циклы[208]

Программисты обычно называют каноническим разложением перестановки ее **реверсированное каноническое разложение**, т.е. разложение, получающееся записью циклов ее канонического разложения в обратном порядке. Кроме того, они, конечено, явно выписывают все тривиальные циклы. Вот, для примера, реверсированное каноническое разложение перестановки, рассмотренной в предыдущем пункте:

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 3 & 5 & 1 & 8 & 9 & 4 & 7 & 6 & 2 \end{pmatrix} = (7)(486)(259)(13).$$

Таким образом, по прежнему каждый цикл начинается с наименьшего элемента, но теперь циклы расположены не в порядке возрастания, а в порядке *убывания* своих наименьших элементов. Ясно, что реверсированное каноническое разложение перестановки получается из ее канонического разложения применением функции `Reverse`. С другой стороны, поскольку входящие в реверсированное каноническое разложение циклы по прежнему независимы, исходная перестановка восстанавливается по нему при помощи той же самой функции `fromCycles`.

---

[208] Never make anything simple and efficient when a way can be found to make it complex and wonderful. *The Tao of Real Programming*

Важным техническим преимуществом реверсированного канонического разложения является то, что — в отличие от обычного канонического разложения! — в нем можно вообще не ставить внутренние скобки, так как их положение *однозначно* восстанавливается. А именно, если стереть в реверсированном каноническом разложении все внутренние скобки, то, чтобы вернуться к исходному разложению, достаточно вставить пару скобок )( перед каждым элементом, слева от которого нет ни одного меньшего элемента. Например, (8 5 9 2 6 4 3 1 7) может получиться *только* из разложения (8)(5 9)(2 6 4 3)(1 7).

**4.1.** Напишите программу, восстанавливающую внутренние скобки в реверсированном циклическом разложении перестановки.

Это типичный формальный экзерсис, относительно которого абсолютно все равно, *как именно* добиваться поставленной цели. Наоборот, в таких случаях на занятиях мы обычно поощряли студентов найти решение при помощи тех функций, с которыми они еще не встречались, или, наоборот, не используя тех функций, использование которых они уже хорошо понимают.

**Решение 1.** Вот как, примерно, это делается в пакете `Combinatorica`. Прежде всего, составим список, состоящий из позиций тех элементов, слева от которых нет меньших элементов:

```
nodes[x_]:=Module[{m=Infinity},DeleteCases[
            Table[If[x[[i]]<m,m=x[[i]];i,0],{i,1,Length[x]}],0]
```

Например, вычисление `nodes[{4,10,8,2,5,9,3,1,6,7}]` вернет `{1,4,8}`. Осталось расставить скобки в исходном списке *перед* номерами, содержащимися в этом списке. Это можно сделать, *например*, при помощи команды `Take`. Вначале мы добавляем $n+1$ к списку узлов, потом разбиваем получившийся список номеров на подсписки длины 2 с отступом 1. Для приведенного выше примера при этом получится `{{1,4},{4,8},{8,11}}`. Каждый такой подсписок состоит из номера позиции, в которой начинается текущий цикл, и номера позиции, в которой начинается следующий цикл — в том числе несуществующий цикл, начинающийся с $n+1$. Наконец, мы выбираем в $x$ фрагменты с соответствующим началом и концом:

```
RevealCycles[x_]:=Map[Take[x,{First[#],Last[#]-1}]&
            Partition[Append[nodes[x],Length[x]+1],2,1]]
```

**Решение 2.** Сейчас мы продемонстрируем несколько другое решение, использующее `Split` для той цели, для которой в первом решении применяется `Take`. Прежде всего, определим список, где на $i$-м месте стоит номер *начала* того цикла перестановки $x$, в который попадает $i$-й элемент. Эту функцию было бы естественно назвать `runs`, но, поскольку это имя уже занято, мы назовем ее `corsa`:

```
corsa[x_]:=Module[{m=Infinity,j=1},
            Table[If[x[[i]]<m,m=x[[i]];j=i,j],{i,1,Length[x]}]]
```

Для того примера, который рассмотрен в предыдущем решении, вычисление `corsa[{4,10,8,2,5,9,3,1,6,7}]` дает `{1,1,1,4,4,4,4,8,8,8}`. А теперь проделаем обычный трюк с двойным транспонированием, который мы

уже много раз использовали с функцией `Sort`. Впрочем, имеются два отличия. Во-первых, `Sort` *по умолчанию* сортирует пары по первому элементу, а для `Split` нужно *явно* задать тест, определяющий эквивалентность на множестве пар, в данном случае совпадение элементов списка `corsa[x]` с соответствующими номерами. Во-вторых, после применения `Sort` мы получали табличную запись самой перестановки, а после применения `Split` мы получим список, состоящий из табличных записей составляющих ее циклов и, значит, `Last[Transpose[#]]` нужно применять не к списку в целом, а к его элементам. Таким образом, мы получаем примерно следующий текст:

```
revealCycles[x_]:=Map[Last[Transpose[#]]&,
          Split[Transpose[{Range[10],x}],
              corsa[x][[First[#1]]]==corsa[x][[First[#2]]]&]]
```

Рассмотрим теперь отображение $f$, сопоставляющее перестановке $\pi$ перестановку, которая получается из нее следующим образом: мы записываем *реверсированное* каноническое разложение $\pi$ на циклы, а потом стираем все внутренние скобки в получившемся разложении и интерпретируем его как сокращенную запись перестановки $f(\pi)$. Действие этого отображения значительно интереснее, чем действие аналогичного отображения, рассматривавшегося в предыдущем параграфе. Если в предыдущем случае тождественная перестановка была неподвижной точкой, то теперь тождественная перестановка $(1,\dots,n)$ переходит в реверсию $(n,\dots,1)$.

**4.2.** Найдите все неподвижные точки отображения $f$.

**Ответ.** При $n \geq 2$ у этого отображения нет неподвижных точек.

**4.3.** Постарайтесь понять динамику отображения $f$.

**Ответ.** Здесь без компьютерного эксперимента, использующего функцию `Flatten[Reverse[toCycles[x]]]` обойтись не удастся. Перестановки степени 3 образуют одну орбиту под действием $f$:

$$(1\,2\,3)\quad (3\,2\,1)\quad (2\,1\,3)\quad (3\,1\,2)\quad (1\,3\,2)\quad (2\,3\,1)$$

Перестановки степени 4 образуют две орбиты, при этом орбита тождественной перестановки состоит из 15 элементов:

$$(1\,2\,3\,4)\quad (4\,3\,2\,1)\quad (2\,3\,1\,4)\quad (4\,1\,2\,3)\quad (1\,4\,3\,2)\quad (3\,2\,4\,1)\quad (2\,1\,3\,4)\quad (4\,3\,1\,2)$$
$$(1\,4\,2\,3)\quad (2\,4\,3\,1)\quad (3\,1\,2\,4)\quad (4\,1\,3\,2)\quad (3\,1\,4\,2)\quad (1\,3\,4\,2)\quad (2\,3\,4\,1)$$

Вторая орбита состоит из 9 элементов:

$$(1\,2\,4\,3)\quad (1\,3\,2\,4)\quad (2\,1\,4\,3)\quad (2\,4\,1\,3)\quad (3\,2\,1\,4)$$
$$(3\,4\,1\,2)\quad (3\,4\,2\,1)\quad (4\,2\,1\,3)\quad (4\,2\,3\,1)$$

Еще интересней картина для степени 5. Здесь тоже имеется две орбиты, одна из них трехэлементная:

$$(2\,1\,3\,5\,4)\quad (4\,5\,3\,1\,2)\quad (3\,2\,5\,1\,4)$$

В то же время орбита тождественной перестановки состоит из остальных 117 перестановок степени 5.

Дальше эта закономерность сохраняется: почти все перестановки данной степени попадают в орбиту тождественной перестановки, например, для степени 6 это 694 перестановки из 720, для степени 7 это 4804 перестановки из 5040. Остальные элементы образуют несколько совсем маленьких орбит. Скажем, для степени 6 порядки этих орбит равны 17, 5 и 4, соответственно.

**5. Умножение циклов**

Чтобы разложение на циклы стало удобным интсрументом, нужно выразить все основные конструкции над перестановками в терминах циклов. Мы уже убедились, что обратная перестановка чрезвычайно естественно выражается в терминах циклов — и еще раз вернемся к этой теме в следующем параграфе. А вот как в терминах циклов ищется произведение перестановок?

**5.1.** Задайте функцию, вычисляющую каноническое разложение на циклы двух перестановок, заданных своими каноническими разложениями на циклы.

**Ответ.** Нет ничего проще, нужно просто перейти от цикленной записи к обычной, перемножить перестановки и снова вернуться к цикленной записи:

```
cycleProduct[x_,y_]:=ToCycles[FromCycles[x][[FromCycles[y]]]]
```

в общей алгебре это называется **перенос структуры**.

Конечно, это совсем не то, что имелось в виду! Цикленная запись не только компактнее полной, но — при некотором навыке — удобнее для вычислений. Чтобы убедиться в этом, рассмотрим более общую задачу. Пусть $\sigma = \sigma_1 \dots \sigma_s$ — *произвольное* произведение циклов. Как вычислить обычную и/или цикленную запись $\sigma$? Разумеется, как и выше, мы можем просто превратить каждый из этих циклов в обычную запись перестановки степени $n$ и перемножить все эти перестановки.

**5.2.** Задайте функцию, вычисляющую каноническое разложение на циклы перестановки, заданной как *какое-то* произведение циклов.

Перестановку $\sigma$ можно вычислить следующим образом. Чтобы определить образ $i$ под действием $\sigma$, начнем с того, что найдем самый правый цикл, скажем, $\sigma_p$, в запись которого входит фрагмент вида $(\dots ij \dots)$ либо $(j \dots i)$, этот цикл переводит $i$ в $j$. Если $i$ не входит в запись ни одного из циклов $\sigma_p$, то $\sigma(i) = i$. Найдем теперь самый правый цикл $\sigma_q$ левее $\sigma_p$, в запись которого входит фрагмент вида $(\dots jh \dots)$ либо $(h \dots j)$, этот цикл переводит $j$ в $h$. Если $j$ не входит в запись ни одного из циклов $\sigma_q$, $q < p$, то $\sigma(i) = j$. В противном случае, продолжая действовать таким же образом, мы в конце концов найдем $\sigma(i)$.

Для иллюстрации этого алгоритма рассмотрим перестановку

$$\sigma = (1753)(162)(46)(3574).$$

Вычислим, для примера, $\sigma(7)$. Цикл (3574) переводит 7 в 4, цикл (46) переводит 4 в 6, цикл (162) переводит 6 в 2, наконец, цикл (1753) оставляет 2 на месте. Поэтому $\sigma(7) = 2$. В действительности, вычисляя теперь образ $\sigma(2)$, мы видим, что $\sigma(2) = 7$, так что в разложение $\sigma$ на независимые циклы входит цикл (27). Прежде, чем переходить к следующему упражнению, мы предлагаем читателю довести это вычисление до конца и записать каноническое разложение $\sigma$ на независимые циклы.

**5.3.** Напишите программу, реализующую описанный выше алгоритм умножения циклов.

**5.4.** Как изменится каноническое разложение перестановки на циклы при умножении на некоторую транспозицию?

**6. Цикленный тип.**

Пусть $\pi \in S_n$ и

$$\underline{n}/\pi = \{X_1, \ldots, X_t\}.$$

Обозначим через $n_i = |X_i|$ порядок орбиты $X_i$. **Цикленным типом** перестановки $\pi \in S_n$ называют *набор* $[n_1, \ldots, n_t]$ порядков ее орбит. Обычно этот набор принято записывать как тупель $(n_1, \ldots, n_t)$, располагая $n_i$ в порядке убывания. Обратите внимание, что говорят о **цикленном** или, изредка, **цикловом**, (но **не** циклическом!) типе. Впрочем, часто эпитет `цикленном` опускают и говорят просто о **типе** перестановки.

Смысл введения цикленного типа состоит в том, что он полностью описывает классы сопряженности в симметрической группе. Иными словами, две перестановки в $S_n$ тогда и только тогда сопряжены, когда их цикленные типы совпадают.

Ясно, что $n_1 + \ldots + n_t = n$, так что с точки зрения комбинаторики $(n_1, \ldots, n_t)$ определяет **разбиение** числа $n$. Порядок длин здесь безразличен, поэтому $n_i$ принято располагать не в том порядке, в котором они идут в каноническом разложении на циклы, а в порядке убывания.

**6.1.** Задайте функцию, сопоставляющую перестановке ее цикленный тип.

**Ответ.** С учетом введенной в предыдущем параграфе функции `toCycles` это легче всего сделать так:

```
cycleType[x_]:=Map[Length,toCycles[x]]
```

**6.2.** Вычислите все возможные цикленные типы перестановок степени $\leq 10$.

Количество всевозможных цикленных типов перестановок степени $n$ равно количеству не??упорядоченных разбиений числа $n$ на натуральные слагаемые и вычисляется внутренней функцией `PartitionsP`.

**6.3.** Перечислите все возможные типы перестановок на 5,6,7,8 и 9 символах.

**6.4.** Докажите, что две перестановки в $S_n$ тогда и только тогда сопряжены, когда их цикленные типы совпадают.

**Решение.** Пусть вначале $\pi, \sigma \in S_n$ — две сопряженные перестановки, скажем, $\pi = \rho\sigma\rho^{-1}$ для некоторого $\rho \in S_n$. Ясно, что если $\sigma(i) = j$, то $\pi\rho(i) = \rho(j)$. Тем самым, если $X_1, \dots, X_t$ — орбиты $\sigma$, то $\rho(X_1), \dots, \rho(X_t)$ — орбиты $\pi$, а так как $\rho \in S_n$, то $|\rho(X_h)| = |X_h|$, так что цикленные типы сопряженных перестановок совпадают.

Обратно, предположим, что

$$\pi = (i_{11} \dots i_{1l}) \dots (i_{t1} \dots i_{tm}),$$

$$\sigma = (j_{11} \dots j_{1l}) \dots (j_{t1} \dots j_{tm}),$$

две перестановки одинакового цикленного типа. Мы утверждаем, что $\pi = \rho\sigma\rho^{-1}$, где

$$\rho = \begin{pmatrix} i_{11} & \dots & i_{1l} & \dots & i_{t1} & \dots & i_{tm} \\ j_{11} & \dots & j_{1l} & \dots & j_{t1} & \dots & j_{tm} \end{pmatrix}.$$

В самом деле, посмотрим, во что $h$-й элемент $r$-го цикла переходит под действием перестановки $\rho\sigma\rho^{-1}$: $i_{rh} \mapsto j_{rh} \mapsto j_{r,h+1} \mapsto i_{r,h+1}$. Поскольку то же верно для всех элементов, перестановка $\rho\sigma\rho^{-1}$ совпадает с $\pi$. Но это и означает, что две перестановки одинакового цикленного типа сопряжены.

**7. Количество перестановок степени $n$ с $m$ циклами.**

Мы уже знаем, что количество длинных циклов, которые можно образовать из $n$ символов, равно $(n-1)! = \begin{bmatrix} n \\ 1 \end{bmatrix}$. Это простейший частный случай следующего результата, который, собственно, и объясняет, в чем состоит комбинаторный смысл чисел Стирлинга первого рода.

**7.1.** Докажите, что количество перестановок $n$ символов, в разложение которых входит ровно $m$ циклов, равно $\begin{bmatrix} n \\ m \end{bmatrix}$.

**Решение.** Обозначим количество перестановок $n$ символов, в разложение которых входит ровно $m$ циклов, через $x(n, m)$. Ясно, что $x(0, 0) = 1$, $x(n, 0) = 0$ для всех $m \geq 1$ и $x(n, m) = 0$ для всех $m > n$. Поэтому для того, чтобы убедиться в том, что $x(n, m) = \begin{bmatrix} n \\ m \end{bmatrix}$, достаточно доказать, что $x(n, m)$ удовлетворяет тому же рекуррентному соотношению, что и числа Стирлинга первого рода. В самом деле, рассмотрим перестановку $\pi \in S_n$ и сфокусируемся на символе $n$. Имеется следующая альтернатива:

• $n$ образует отдельную орбиту $\pi$. Убирая эту орбиту мы получаем перестановку $\sigma \in S_n$ с $m-1$ орбитой, так что количество таких перестановок равно $x(n-1, m-1)$.

• $n$ входит в какой-то цикл длины $\geq 2$. В этом случае вычеркивая в цикленной записи перестановки $\pi$ символ $n$ мы получим перестановку $\sigma \in S_{n-1}$, у которой по прежнему $m$ орбит, причем каждая такая перестановка получится из некоторой перестановки $\pi \in S_n$ с $m$ орбитами. Обратно, рассмотрим произвольную перестановку $\sigma \in S_{n-1}$ с $m$ циклами длин $l_1, \dots, l_m$. Чтобы восстановить из нее перестановку $\pi$, мы должны врисовать символ $n$ в какой-то цикл. Имеется ровно $l+1$ способов врисовать $n$ в

цикл $(i_1 \dots i_l)$ длины $l$, но $(ni_1 \dots i_l) = (i_1 \dots i_l n)$ являются просто разными записями одного и того же цикла. Таким образом, среди этих $l+1$ способов ровно $l$ различных. Так как мы можем врисовать $n$ в каждый из $m$ циклов перестановки $\sigma$, это значит, что общее количество перестановок $\pi$, превращающихся в $\sigma$ после вычеркивания $n$, равно $l_1 + \ldots + l_m = n-1$, и не зависит от $\sigma$. Таким образом, общий вклад этого случая равен $(n-1)x(n-1,m)$

Суммируя полученные результаты, получаем

$$x(n,m) = (n-1)x(n-1,m) + x(n-1,m-1),$$

как и утверждалось.

**7.2.** Докажите, что $\left\{ {n \atop m} \right\} \leq \left[ {n \atop m} \right]$

**Решение.** Число Стирлинга второго рода $\left\{ {n \atop m} \right\}$ совпадает с количеством всевозможных разбиений $X = \underline{n}$ на $m$ орбит, и для каждого такого разбиения имеется по крайней мере одна перестановка с данным разбиением на орбиты. В случае, когда имеется хотя бы одна трехэлементная орбита, существует более одной такой перестановки. Поэтому как правило $\left\{ {n \atop m} \right\} < \left[ {n \atop m} \right]$.

Следующая задача легко решается от руки.

**7.3.** Перечислите все перестановки четырех символов, представимые в виде произведения двух циклов.

**Решение.** Так как $\left[ {4 \atop 2} \right] = 11$, то существует 11 таких перестановок, а именно

$$(1)(234), \quad (1)(243), \quad (2)(134), \quad (2)(143), \quad (3)(124), \quad (3)(142),$$
$$(4)(123), \quad (4)(132), \quad (12)(34), \quad (13)(24), \quad (14)(23).$$

**7.4.** Напишите программу, порождающую все перестановки степени $n$ представимые как произведения $m$ циклов.

**8. Статистика циклов.**

В этом разделе мы вычислим, сколько циклов *в среднем* имеет перестановка степени $n$.

**8.1.** Чему равно общее число циклов в канонических разложениях всех перестановок степени $n$?

**Ответ.** Следующая функция дает общее количество циклов в перестановках степени $n$:

```
grandtotal[n_]:=Total[Map[Length[toCycles[#]]&,Permutations[n]]
```

Вот несколько первых значений этой функции

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 3 | 11 | 50 | 274 | 1764 | 13068 | 109584 | 1026576 | 10628640 |

Секундное наблюдение наводит на мысль, что числа во второй строке представляют собой $n!H_n$.

**8.2.** Вычислите, сколько элементных циклов длины $m$ содержат все перестановки степени $n$ и докажите полученный в предыдущей задаче результат.

**Решение.** Прежде всего заметим, что каждый цикл длины $m$ входит в $(n-m)!$ перестановок, так как их действие на элементах, не входящих в цикл, может быть совершенно произвольным. Посчитаем теперь количество возможных циклов длины $m$. В качестве первого элемента такого цикла может фигурировать любой из $n$ элементов, в качестве второго — любой из оставшихся $n-1$ элемента, в качестве третьего — любой из оставшихся $n-2$ элементов и т.д. Наконец, в качестве последнего элемента может фигурировать любой из оставшихся $n-m+1$ элемента. Таким образом, общее количество *записей* циклов длины $m$ равно $[n]_m$, но при этом каждый цикл будет представлен $m$ раз. Таким образом, количество возможных циклов равно $[n]_m/m$ и, значит, общее количество $m$-циклов во всех перестановках равно $(n-m)![n]_m/m = n!/m$. Складывая получившиеся значения по всем $m$, мы как раз и получим

$$n!\left(1+\frac{1}{2}+\frac{1}{3}+\ldots+\frac{1}{n}\right)=n!H_n.$$

**8.3.** Сколько *в среднем* циклов имеет перестановка степени $n$?

**Решение.** Естественно, $H_n$, для этого нужно лишь разделить вычисленное в предыдущей задаче общее количество циклов во всех перестановках степени $n$ на общее количество перестановок.

Поставим теперь вопрос в другую сторону.

**8.4.** С какой вероятностью перестановка степени $n$ имеет $m$ циклов?

**Решение.** Естественно, $\left[\begin{matrix} n \\ m \end{matrix}\right]/n!$, для этого нужно разделить вычисленное в предыдущем параграфе количество перестановко степени $n$ с $m$ циклами на общее количество перестановок.

**8.5.** Чему равна *средняя* длина цикла, входящего в перестановку степени $n$?

**Решение.** Ответ на этот вопрос зависит от того, как конкретно определяется *средняя* длина — что именно считается равновероятным? Если считать, что равновероятны *сами циклы*, то, естественно, $n/H_n$, для этого нужно просто разделить степень перестановки на среднее количество циклов. Однако, если случайным образом выбирать символ от 1 до $n$, то с большей вероятностью он будет содержаться в более длинном цикле.

### 9. Наибольший порядок элемента $S_n$.

Рассмотрим **стандартную карточную колоду** $P = \texttt{pack}$ из 52 карт. Элементы симметрической группы $S(P) \cong S_{52}$ принято называть **тасовками** $= \texttt{shuffles}$. Сколько раз нужно повторить *фиксированную* тасовку,

чтобы порядок карт совпал с первоначальным? Какое наибольшее число раз нам может при этом понадобиться стасовать колоду? В настоящем параграфе мы установим, что это число равно 180180, см., например,[209].

С математической точки зрения речь здесь идет о нахождении наибольшего порядка перестановки $\pi \in S_n$, который мы обозначим через $G(n)$. Что можно сказать о функции $n \mapsto G(n)$? Ясно, что для $n = 2, 3, 4$ наибольший порядок в группе $S_n$ имеет соответствующий длинный цикл. Но для $n = 5$ это уже не так, порядок длинного цикла равен 5, в то время принадлежащий $S_5$ элемент цикленного типа $(3, 2)$ имеет порядок $3 \cdot 2 = 6$. Еще более замечательные явления происходят для $n = 6$, в этом случае как длинный цикл, так и элемент цикленного типа $(3, 2, 1)$ имеют порядок 6. В то же время, как мы установили в предыдущем параграфе, они не сопряжены в $S_6$. Однако, сюрпризы здесь не кончаются!

**9.1.** Найдите количество 6-циклов и количество перестановок цикленного типа $(3, 2, 1)$ в $S_6$.

**Решение.** Как мы знаем, количество 6-циклов в группе $S_6$ равно $5! = 120$. Количество же элементов типа $(3, 2, 1)$ там равно $2!\binom{6}{3}\binom{3}{2} = 120$. Тем самым, порядки обоих классов элементов максимального порядка в $S_6$ равны. В действительности, у группы $S_6$ существует внешний автоморфизм, переставляющий эти два класса. С этим явлением связано исключительное поведение групп $S_5$ и $S_6$, которое накладывает отпечаток на всю теорию симметрических групп.

Как уже было сказано, в случае $n \leq 4$ наибольший порядок в $S_n$ имеет длинный цикл, так что $G(n) = n$. Однако, начиная с $n = 5$ жизнь становится веселее. Следующий экзерсис проделывал каждый, кто начинал изучать теорию групп.

**9.2.** Вычислите $G(n)$ при $n \leq 11$.

**Ответ.** Ограничимся ответом. В рубрике `cycle type` указаны цикленные типы элементов наибольшего порядка:

| $n$: | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|
| $G(n)$: | 6 | 6 | 12 | 15 | 20 | 30 | 30 |
| `cycle type:` | $(3,2)$ | $(3,2,1)$ или $(6)$ | $(4,3)$ | $(5,3)$ | $(5,4)$ | $(5,3,2)$ | $(5,3,2,1)$ или $(6,5)$ |

Вот еще несколько значений, которые легко вычисляются от руки:

| $n$: | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
|---|---|---|---|---|---|---|---|
| $G(n)$: | 60 | 60 | 84 | 105 | 140 | 210 | 210 |
| `cycle type:` | $(5,4,3)$ | $(5,4,3,1)$ | $(7,4,3)$ | $(7,5,3)$ | $(7,5,4)$ | $(7,5,3,2)$ | $(7,5,3,2,1)$ или $(7,6,5)$ |

[209] M.B.Nathanson, On the greatest order of an element of the symmetric group. — Amer. Math. Monthly, 1972, vol.79, p.500–501.

Уильям Миллер[210] обрывает свою таблицу в этом месте. Сейчас мы вычислим еще несколько значений $G(n)$, так как начиная с $n = 19$ они становится интереснее в нескольких отношениях.

Как вычислять эти значения? Пусть $\pi$ — перестановка цикленного типа $(m_1, \ldots, m_s)$. Ее порядок $m = \mathrm{lcm}(m_1, \ldots, m_2)$ является *наименьшим общим кратным* длин циклов. Сейчас мы убедимся в том, что единственный интересный случай с точки зрения максимизации порядка — это случай *примарных* $m_1, \ldots, m_s$. Следующие соображения приводятся на страницах 500–501 работы Миллера.

**9.3.** Пусть $m = \mathrm{lcm}(m_1, \ldots, m_s)$ — наименьшее общее кратное натуральных чисел $m_1, \ldots, m_s \in \mathbb{N}$, а $m = {p_1}^{l_1} \ldots {p_s}^{l_s}$ — его каноническое разложение на простые. Докажите, что тогда

$$\mathrm{spread}(m) = {p_1}^{l_1} + \ldots + {p_s}^{l_s} \leq m_1 + \ldots + m_s.$$

**Решение.** По уровню и типу это образцовая задача районной олимпиады для 6-го класса, поэтому ограничимся следующим комментарием. Достаточно показать, что если $m_1, \ldots, m_s$ *не являются* степенями попарно различных простых, то существует другая последовательность натуральных чисел $(n_1, \ldots, n_t)$ такая, что $m = \mathrm{lcm}(n_1, \ldots, n_t)$, но при этом $n_1 + \ldots + n_t < m_1 + \ldots + m_s$.

**9.4.** В группе $S_n$ тогда и только тогда существует элемент порядка $m$, когда $\mathrm{spread}(m) \leq n$.

**9.5.** Если в группе $S_n$ существует элемент порядка $m$, то в ней существует элемент порядка $m$, все длины нетривиальных циклов которого являются степенями попарно различных простых.

В частности, это относится к элементам наибольшего порядка. Тем самым мы получаем следующую характеризацию $G(n)$, при помощи которой на компьютере легко вычислить первые *несколько десятков* значений этой функции.

**9.6.** Имеет место равенство $G(n) = \max(m)$, где максимум берется по всем $m \leq n!$ таким, что $\mathrm{spread}(m) \leq n$.

**9.7.** Напишите программу, вычисляющую $G(n)$ и вычислите первые несколько десятков значений этой функции.

**Решение.** Вот следующие пять значений с указанием *какого-то* класса наибольшего порядка:

| $n$: | 19 | 20 | 21 | 22 | 23 |
|---|---|---|---|---|---|
| $G(n)$: | 420 | 420 | 420 | 420 | 840 |
| `cycle type:` | $(7,5,4,3)$ | $(7,5,4,3,1)$ | $(7,5,4,3,1,1)$ | $(7,5,4,3,1,1,1)$ | $(8,7,5,3)$ |

[210] W.Miller, The maximum order of an element of a finite symmetric group. — Amer. Math. Monthly, 1987, June–July, p.497–506.

Сразу бросаются в глаза два обстоятельства. Во-первых, появляется длинная серия $G(19) = G(20) = G(21) = G(22) = 420$ когда $G(n)$ не растет. Во-вторых, начиная с $n = 21$ известный нам контрпример, связанный с существованием двух классов сопряженности элементов максимального порядка, перестает быть единственным. Например, ясно, что элемент цикленного типа $(7, 5, 4, 3, 2)$ имеет тот же порядок, что элемент цикленного типа $(7, 5, 4, 3, 1, 1)$ — кстати, это служит контрпримером к заявлению на странице 501 упомянутой выше статьи Миллера.

Приведем еще несколько значений $G(n)$, уже без указания соответствующих элементов:

$$G(24) = 840, \quad G(25) = G(26) = 1260, \quad G(27) = 1540, \quad G(28) = 2310,$$
$$G(29) = 2520, \quad G(30) = G(31) = 4620, \quad G(32) = G(33) = 5460,$$
$$G(34) = G(35) = 9240, \quad G(36) = G(37) = 13860, \quad G(38) = G(39) = 16380$$

Ну и, как уже отмечалось в самом начале раздела, $G(52) = 180180$. Хотя вычисление этого значения в дуболомном стиле, используя разложение на простые множители *всех* чисел до нескольких миллионов, займет несколько минут машинного времени.

Итак, вычисление $G(n)$ сводится к теоретико-числовой задаче. Однако, получение явной формулы для $G(n)$ представляется довольно сомнительным предприятием. Лучшее, на что мы можем надеяться, — это асимптотические оценки. Первый нетривиальный результат в этом направлении был получен Эдмундом Ландау[211,212] А именно, он доказал, что $\ln(G(n)) \sim \sqrt{n \ln(n)}$. Первоначальное доказательство Ландау в полном объеме использовало теорему о распределении простых.

**10. Участки подъема.**

В первом параграфе настоящей главы мы уже определили число Эйлера $\left\langle {n \atop m} \right\rangle$ как количество перестановок $(x_1 \ldots x_n)$ степени $n$ с $m$ **участками подъема**, т.е. таких, для которых существует ровно $m$ индексов $i$ таких, что $x_i < x_{i+1}$. Композиция с реверсией $(n\, n-1 \ldots 2\, 1)$ показывает, что мы могли бы определять числа Эйлера в терминах **участков спуска** $x_i > x_{i+1}$.

Рассмотрим теперь **восходящие серии**, состоящие из последовательных участков подъема, ограниченных двумя участками спуска и/или концами перестановки:

$$x_{h-1} > x_h < x_{h+1} < \ldots < x_{k-1} < x_k > x_{k+1}.$$

Ясно, что количество восходящих серий на 1 больше, чем количество участков спуска. Точно так же количество **нисходящих серий**

$$x_{h-1} < x_h > x_{h+1} > \ldots < x_{k-1} > x_k < x_{k+1}$$

---

[211] E.Landau, Über die Maximalordnung der Permutation gegebenen Grades. — Archiv der Math. u. Phys., Ser.3, 1903, Bd.5, S.92–103.

[212] E.Landau, Handbuch der Lehre von der Verteilung der Primzahlen, Bd. I, 2nd ed., Chelsea, N.Y., 1953, S.222–229.

на 1 больше, чем количество участков подъема. Поэтому мы могли бы определять числа Эйлера в терминах восходящих или нисходящих серий.

**10.1.** Напишите программу, которая разбивает перестановку на восходящие серии.

**Решение.** Вот как эта функция определяется в пакете `Combinatorica`:

```
runs[x_]:=Map[Take[x,{First[#]+1,Last[#]}]&,
                  Partition[Join[{0},Select[Range[Length[x]-1],
                       x[[#]]>x[[#+1]]&],{Length[x]}],2,1]]
```

**10.2.** Напишите программу, которая ищет в перестановке самую длинную восходящую серию.

**10.3.** Напишите программу, которая разбивает перестановку на нисходящие серии.

При нашей новой интерпретации легко получить комбинаторные доказательства основных соотношений для чисел Эйлера.

**10.4.** Докажите рекуррентное соотношение для чисел Эйлера.

Согласно нашей новой интерпретации число Эйлера $\left\langle {n \atop m} \right\rangle$ представляет собой количество перестановок степени $n$ с $m+1$ восходящей серией. Как может получиться такая перестановка из перестановок степени $n-1$? Рассмотрим произвольную перестановку степени $n-1$ и будем врисовывать $n$ во все возможные места.

• Если исходная перестановка содержала $m+1$ серию, то врисовывание $n$ в конец серии не меняет количество серий. Таким образом, имеется ровно $m+1$ способ получить из перестановки степени $n-1$ с $m+1$ серией перестановку степени $n$ с $m+1$ серией.

• С другой стороны, врисовывание $n$ в любое другое место увеличивает количество серий на 1. Таким образом, второй способ получить перестановку с $m+1$ состоит в том, чтобы взять перестановку степени $n-1$ с $m$ сериями и врисовать $n$ в любую позицию, *кроме* концов уже имеющихся серий. Это и есть искомое рекуррентное соотношение.

> Gauls! We have nothing to fear; except perhaps that the sky may fall on our heads tomorrow. But as we all know, tomorrow never comes!!
>
> Adventures of Asterix

# Глава 9. МНОГОЧЛЕНЫ И МАТРИЦЫ

Математика слишком обширна и овладеть ею всей невозможно. Теорфизикой же можно овладеть всей.

Лев Ландау

Если нет обмана, это уже не теорфизика, а математика.

Александр Ахиезер

Человек, не знающий и не желающий знать основных законов науки, в мирное время вреден, а в военное — опасен.

Аркадий Мигдал

Sometimes attaining the deepest familiarity with a question is our best substitute for actually having the answer.

Brian Greene, *The Elegant Universe*

It is most important to have a beautiful theorem. And if the proofs don't support it, don't be too distressed, but wait a bit and see if some error in the proofs doesn't show up.

Paul Dirac

A mathematician's life would be a happy one if he had only to discover and never to write proofs.

Daniel Gorenstein

Just because something happens to be true, it does not follow that it should be taken for granted.

Raymond M. Smullyan. *Is God a Taoist*?

I thought to address my lectures to the most intelligent in the class and to make sure, if possible, that even the most intelligent student was unable to completely encompass everything that was in the lectures.

Richard Feynman, *Lectures on physics*

What is written with effort is in general read without pleasure.

Samuel Johnson

Старый мэтр называл алгебру музыкой души.

Николай Гумилев, *Скрипка Страдивариуса*

. . . Февраль. Достать чернил и паркер!

Тимур Кибиров

Набор литературных цитат, которые автивно помнит человек, образует неотъемлемую составляющую его личности. Эта составляющая используется и в процессе коммуникации между людьми,

поскольку любая коммуникация основана на некоторой общности исходных знаний. Так, Тимур Кибиров в своей поэзии обращается к тем, кто замечает и понимает все разбросанные по его стихам явные и косвенные цитаты — или хотя бы значимую часть этих цитат.

Владимир Успенский, ... *и лесные сраки*

Согласны ли Вы с тем, что каждый фильм должен иметь начало, середину и конец?
Конечно, но не обязательно в этом порядке.

Жан-Люк Годар

Мы далеки от мысли. От всякой мысли.

Виктор Черномырдин

Все-таки есть кое-что приятное в постоянстве законов истории: вот, например, сейчас нормальному человеку из языков, если говорить честно, ничего, кроме русского и английского, не требуется — и в прежние времена в общем-то так и было.

Андрей Зализняк

Все это вздорные аллегории.

Варвара Петровна Ставрогина

## § 1. Многочлены

Polynomials and power series.
May they forever rule the worls.

Shreeram S. Abhyankar

### 1. Анатомия многочленов.[213]

| | |
|---|---|
| `PolynomialQ[f,{x,y,z}]` | тест полиномиальности |
| `Variables[f]` | входящие в $f$ переменные |
| `Exponent[f,x]` | степень $f$ по $x$ |

В настоящем разделе мы рассматриваем многочлены от одной переменной. Каждый такой многочлен представляется в виде

$$f = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0,$$

где $x$ переменная, а $a_n, a_{n-1}, \ldots, a_1, a_0$ — коэффициенты.

Многочлен $f \in K[x]$ называется **четным**, если в него входят только четные степени $x$. Таким образом, четные многочлены — это в точности элементы $K[x^2]$. Аналогично, многочлен $f \in K[x]$ называется **нечетным**, если в него входят только нечетные степени $x$. Каждый многочлен единственным образом представляется в виде суммы четной и нечетной частей.

[213]Кипарис фалличен. Пальма — вагинальна. ©Тимур Кибиров, *Образы Италии*

Четный многочлен представляется в виде $f(x) = g(x)^2$, а нечетный — в виде $f(x) = xg(x)^2$, для некоторого многочлена $g \in K[x]$. В частности, для четного многочлена имеет место равенство $f(-x) = f(x)$, а для нечетного — равенство $f(-x) = -f(x)$.

**1.1.** Напишите процедуру, устанавливающую четность/нечетность многочлена $f$ и, в случае положительного ответа, находящую многочлен $g$.

Следующий пример показывает, что наивные представления об операциях над многочленами могут нас обманывать.

**1.2.** Существует ли многочлен $f \in \mathbb{Z}[x]$, квадрат которого $f^2$ имеет меньше ненулевых коэффициентов, чем сам $f$? Организуйте поиск такого многочлена. Удалось ли Вам его найти?

**Ответ.** Вы будете смеяться, но такие $f$ существуют. Вот только найти его без шансов! Приведем наименьший пример[214], который очевидно находится далеко за пределами непосредственного поиска на домашнем компьютере:

$$f = 13750x^{12} + 5500x^{11} - 1100x^{10} + 440x^9 - 220x^8 + 220x^7 \\ - 15x^6 - 50x^5 + 10x^4 - 4x^3 + 2x^2 - 2x - 1.$$

Квадрат этого многочлена равен

$$f^2 = 189062500x^{24} + 151250000x^{23} + 2662000x^{19} + 2685100x^{18} - \\ 2217600x^{17} - 83595x^{12} + 2820x^{11} - 660x^7 + 286x^6 + 44x^5 + 4x + 1.$$

Исходный многочлен имеет 13 ненулевых коэффициентов, а его квадрат — всего лишь 12. Заметим, что степень 12 минимальна, многочленов с таким свойством степени $\leq 11$ не существует.

**2. Коэффициенты многочленов.**

| | |
|---|---|
| `Coefficient[f,h]` | коэффициент при $h$ в $f$ |
| `Coefficient[f,x,n]` | коэффициент при $x^n$ в $f$ |
| `CoefficientList[f,x]` | список коэффициентов многочлена $f$ |
| `CoefficientArrays[f,x]` | список коэффициентов малочлена $f$ |

Следующие задачи взяты из книги[215]. Разумеется, там не предполагалось, что они будут решаться с использованием компьютера.

**2.1.** Найдите сумму коэффициентов многочлена $(x^2 + 3x - 3)^{317}$.

**2.2.** Найдите коэффициенты при $x^{17}$ и $x^{18}$ в выражении $(1 + x^5 + x^7)^{20}$.

---

[214] M.Kreuzer, L.Robbiano, Computational commutative algebra. I. — Springer-Verlag, Berlin et al., 2000.

[215] А.М.Яглом, И.М.Яглом, Элементарные задачи в неэлементарном изложении. — М., ГИТТЛ, 1954, с.1–543.

**2.3.** В котором из многочленов $(1 + x^2 - x^3)^{1000}$ или $(1 - x^2 + x^3)^{1000}$ коэффициент при $x^{20}$ больше?

**2.4.** Найдите коэффициент при $x^{50}$ в многочлене

$$(1 + x)^{1000} + x(1 + x)^{999} + x^2(1 + x^{998}) + \ldots + x^{1000}.$$

**2.5.** Найдите коэффициент при $x^{50}$ в многочлене

$$(1 + x) + 2(1 + x)^2 + 3(1 + x)^3 + \ldots + 1000(1 + x)^{1000}.$$

### 3. Значения многочленов.

Казалось бы, вычисление значения многочлена $f$ в точке $z$ требует выполнения $2n-1$ умножений: $n-1$ умножения для нахождения степеней $z$ и $n$ умножений на коэффициенты. В действительности, как было известно в древнем Китае, здесь можно обойтись $n$ умножениями. Скажем, многочлен

$$f = ax^4 + bx^3 + cx^2 + dx + e$$

четвертой степени можно переписать в виде

$$f = (((ax + b)x + c)x + d)x + e,$$

где производится только четыре умножения. В Европе такая схема вычислений известна как схема Руффини–Горнера.

**3.1.** Реализуйте вычисление значения многочлена $f$ по схеме Руффини–Горнера.

**Решение.** Проще всего при помощи команды `Fold`, примерно так:

```
Fold[#1*x+#2&,0,{a,b,c,d,e}]
```

Обратите внимание на то, первым аргументом команды `Fold` является функция от *двух* аргументов

| `HornerForm[f,x]` форма Горнера многочлена $f$ |
|---|

При этом для вычисления значения многочлена степени $n$ схема Руффини—Горнера требует $n$ умножений и $n$ сложений. Эта схема является схемой **без предварительной обработки коэффициентов**. Иными словами, на каждом шаге каждый операнд является либо $x$, либо одним из коэффициентов многочлена, либо абсолютной константой. Легко доказать, см., например[216], что схема Руффини—Горнера является *неулучшаемой* в этом классе, иными словами, $n + n$ это *минимальное* число операций, которое возможно для схемы без предварительной обработки коэффициентов.

[216] В.Я.Пан, О способах вычисления значений многочленов. — Успехи Мат. Наук, 1966, т.21, N.1, с.103–134. Теорема 1.1.

В действительности, если требуется многократно вычислять значения одного и того же многочлена (скажем при приближенном вычислении значений функции при помощи отрезков ряда Тэйлора), как правило, выгодно произвести **предварительную обработку коэффициентов**, т.е. выразить многочлен в другой форме, так что потом для вычисления значения многочлена требуется меньшее число алгебраических операций.

Вот простейший пример такой схемы, предложеный Тоддом. Выделим квадрат в многочлене четвертой степени:

$$f = a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0 = a_4((x(x+b)+c)(x(x+b)+x+d)+e),$$

где коэффициенты $b, c, d, e$ ищутся по формулам

$$b = \frac{-a_4 - a_3}{2a_4}, \quad c = \frac{a_1 - ba_2}{a_4} + b^2(b+1), \quad d = \frac{a_2}{a_4} - b(b+1), \quad e = \frac{a_0}{a_4} - cd.$$

Ясно, что теперь вычисление индивидуального значения $f(x)$ требует лишь 3 умножения и 5 сложений.

Пусть теперь $f$ — многочлен степени $n$. В упражнениях 12.35 и 12.36 в книге А.Ахо, Дж.Хопкрофта, Дж.Ульмана приведен алгоритм вычисления $f(x)$, использующий $\lfloor n/2 \rfloor + 2$ умножений. С другой стороны, Теорема 12.4 той же книги утверждает, что не существует схемы, вычисляющей значение произвольного многочлена степени $n$ менее, чем за $\lfloor n/2 \rfloor$ умножений.

**3.2.** Пусть $f(t) = at^2 + bt + c$ — квадратичный многочлен, Проверьте, что

$$f(1) + f(4) + f(6) + f(7) = f(2) + f(3) + f(5) + f(8).$$

Как доказывается это утверждение?

**Ответ.** В данном случае проверка является доказательством. А именно, доказываемое утверждение состоит в том, что

$$1^0 + 4^0 + 6^0 + 7^0 = 2^0 + 3^0 + 5^0 + 8^0,$$
$$1^1 + 4^1 + 6^1 + 7^1 = 2^1 + 3^1 + 5^1 + 8^1,$$
$$1^2 + 4^2 + 6^2 + 7^2 = 2^2 + 3^2 + 5^2 + 8^2.$$

Обратите внимание, что это ровно тот тип *абсолютно бесполезных* задач о суммах степеней, который рассматривается в аддитивной теории чисел!

**3.3.** Разбейте начальный отрезок натурального ряда $1, 2, \ldots, 16$ на две части так, чтобы сумма значений любого кубического многочлена на одной из этих частей равнялась сумме значений на другой. Единственно ли такое разбиение?

**3.4.** Докажите, что вообще для любого $n$ можно разбить начальный отрезок натурального ряда от 1 до $2^{n+1}$ на две части так, чтобы сумма значений любого многочлена степени $n$ на одной из этих частей равнялась сумме значений на другой.

**3.5.** Обобщите предыдущую задачу. А именно, докажите, что для любых $m$ и $n$ можно разбить начальный отрезок натурального ряда от 1 до $m^{n+1}$ на $m$ попарно дизъюнктных частей так, чтобы все суммы значений любого многочлена степени $n$ на этих частях совпадали.

Однако, мы не сделали все возможное в направлении обобщения нашей исходной задачи. Мы доказали, что для любого многочлена степени $n$ найдутся два различных $s$-элементных множества, где $s = 2^n$, суммы значений многочлена на которых совпадают. Является ли это $s$ наименьшим возможным? Следующая задача показывает, что нет.

**3.6.** Пусть $f(t) = at^2 + bt + c$ — квадратичный многочлен, Проверьте, что

$$f(1) + f(5) + f(6) = f(2) + f(3) + f(7).$$

**3.7.** Можно ли понизить оценку $s = 2^n$ для многочленов степени $n = 3, 4, 5, 6$?

**4. Деление многочленов с остатком.**

| | |
|---|---|
| `PolynomialQuotient[f,g,x]` | неполное частное при дедении $f$ на $g$ |
| `PolynomialRemainder[f,g,x]` | остаток при делении $f$ на $g$ |
| `PolynomialMod[f,g]` | ibid., другим манером |
| `PolynomialGCD[f,g]` | наибольший общий делитель |
| `PolynomialLCM[f,g]` | наименьшее общее кратное |

**4.1.** Найдите остаток от деления многочлена $x + x^3 + x^9 + x^{27} + x^{81} + x^{243}$ на $x^2 - 1$.

**4.2.** Найдите коэффициент при $x^{14}$ в неполном частном, получающемся при делении $x^{1951}$ на $x^4 + x^3 + 2x^2 + x + 1$.

**4.3.** Делится ли многочлен

$$x^{9999} + x^{8888} + x^{7777} + \ldots + x^{1111} + 1$$

на $x^9 + x^8 + x^7 + \ldots + x + 1$?

**4.4.** Найдите наименьшее $m \geq 1000$, для которого многочлен $(x + 1)^m - x^m - 1$ делится на $(x^2 + x + 1)^2$.

**4.5.** Найдите наименьшие $l \geq 1000$, $m \geq 2000$, $n \geq 3000$, для которых многочлен $x^l - x^m + x^n$ делится на $x^2 - x + 1$.

**4.6.** Найдите наименьшие $l \geq 1000$, $m \geq 2000$, $n \geq 3000$, для которых многочлен $x^l + x^m + x^n$ делится на $x^2 + x + 1$.

**5. Структурные манипуляции.**

| | |
|---|---|
| `Expand[f]` | раскрыть все скобки |
| `Collect[f,x]` | расписать $f$ по степеням $x$ |
| `Collect[f,{x,y,z}]` | расписать $f$ по одночленам от $x, y, z$ |

| | |
|---|---|
| `Factor[f]` | разложить многочлен $f$ на множители |
| `FactorSquareFree[f]` | вынести квадратичные множители |
| `FactorTerms[f,x]` | вынести множители, не зависящие от $x$ |
| `FactorList[f]` | список неприводимых множителей |
| `FactorSquareFreeList[f]` | список квадратичных множителей |
| `FactorTermsList[f]` | список множителей, в зависимости от $x$ |

**5.1.** Разложите на множители первые 20 многочленов вида $x^i - y^i$.

**Ответ.** Проще всего так:

```
For[i=1,i<20,i++,Print[x^i-y^i,Factor[x^i-y^i]]]
```

**5.2.** Упростите сумму $1 + x + 2x^2 + \ldots + nx^n$.

**5.3.** А теперь проверьте следующие формулы (Кнут I-1.2.3.20):

$$9\cdot 1+2=11,\quad 9\cdot 12+3=111,\quad 9\cdot 123+4=1111,\quad 9\cdot 1234+5=11111,\ldots$$

и объясните их с помощью результата предыдущей задачи.

**Ответ.** Небольшой эксперимент убедит Вас в том, что $n$ в выражении

```
9*FromDigits[Table[i,{i,1,n}]]+(n+1)
```

совершенно не обязано быть цифрой. Оно может быть любым числом и даже символом или выражением. Кстати, только теперь мы понимаем, как работает `FromDigits`!

С вычислительной точки зрения многочлен — это просто число в базе $x$ (или, если угодно, число это многочлен от неизвестной 10).

**5.4.** Задайте многочлен $f = a_n x^n + \ldots + a_1 x + a_0$ с помощью команды `FromDigits`.

**Ответ.** Ну конечно, `FromDigits[{an,...,a1,a0},x]`.

**5.5.** Можно ли разложить многочлен

$$x^{2222} + 2x^{2220} + 4x^{2218} + \ldots + 2218x^4 + 2220x^2 + 2222$$

в произведение двух многочленов с целыми коэффициентами?

**5.6.** Можно ли разложить многочлен

$$x^{250} + x^{249} + x^{248} + \ldots + x^2 + x + 1$$

в произведение двух многочленов с целыми коэффициентами?

**6. Неразложимые многочлены.**

| | |
|---|---|
| `Decompose[f,x]` | разложить многочлен $f$ в композицию |

Разложение многочленов в композицию не единственно.

$$(x^2+x+1)\circ(x^2-1) = x^4-x^2+1 = (x^2-x+1)\circ x^2.$$

$$(x^2+x+1)\circ(x^2) = x^4+x^2+1 = (x^2-x+1)\circ(x^2+1).$$

**7. Алгебраические уравнения.**

| | |
|---|---|
| `Solve[f==0,x]` | решить уравнение $f(x)=0$ |
| `Roots[f==0,x]` | решить уравнение $f(x)=0$ |

| | |
|---|---|
| `NSolve[f==0,x]` | численное решение уравнения $f(x)=0$ |
| `FindRoot[f==0,{x,x0}]` | корень уравнения $f(x)=0$ вблизи $x_0$ |

**8. Корни многочленов.**

| | |
|---|---|
| `Root[f,i]` | $i$-й корень многочлена $f$ |
| `RootReduce[g]` | преобразовать выражение $g$ к `Root` |
| `ToRadicals[g]` | преобразовать входящие в $g$ объекты типа `Root` в радикалы |
| `RootSum[f,g]` | сумма $g(x_i)$ по корням многочлена $f$ |

## § 2. Запись матриц

**1. Общий элемент.** В данном пункте познакомимся с самым простым и важным способом задания матриц. Этот способ состоит в явном указании общего элемента матрицы при помощи команды `Table`. А именно, матрица $x=(x_{ij})$ размера $m\times n$, у которой элемент в позиции $(i,j)$ равен $x_{ij}$ задается посредством `Table[x[i,j],{i,1,m},{i,1,n}]`

| | |
|---|---|
| `Table[f[i,j],{i,1,m},{j,1,n}]` | матрица значений функции $f$ |
| `Array[f,{m,n}]` | массив значений функции $f$ |

В системе есть внутренняя функция `DiagonalMatrix`, которая сопоставляет списку $\{x_1,\dots,x_n\}$ диагональную матрицу с элементами $x_1,\dots,x_n$ по главной диагонали. Тем не менее, в качестве практики на использование `Table`, решите следующую задачу.

**1.1.** Задайте диагональную матрицу с элементами $x_1,\dots,x_n$.

**Ответ.** Например, так

```
diag[x_,n_]:=Table[If[i==j,x[i],0],{i,1,n},{j,1,n}]
```

**1.2.** Линейные комбинации $xe + y\texttt{test}$ единичной и пробной матриц называются **комбинаторными матрицами**. Изобразим для примера комбинаторную матрицу степени 4:

$$\begin{pmatrix} x+y & y & y & y \\ y & x+y & y & y \\ y & y & x+y & y \\ y & y & y & x+y \end{pmatrix}$$

Задайте комбинаторную матрицу порядка $n$, убедитесь, что для ненулевых $x, x+ny$ она обратима и найдите обратную к ней.

**1.3.** Следующая матрица, известная как **матрица Вандермонда**, десятки раз возникает в нашем учебнике по самым разным поводам:

$$V(x_1, \ldots, x_n) = \begin{pmatrix} 1 & 1 & \ldots & 1 \\ x_1 & x_2 & \ldots & x_n \\ x_1^2 & x_2^2 & \ldots & x_n^2 \\ \ldots & \ldots & \ldots & \ldots \\ x_1^{n-1} & x_2^{n-1} & \ldots & x_n^{n-1} \end{pmatrix}$$

Задайте матрицу Вандермонда и угадайте формулу для обратной к ней.

**Ответ.** Непосредственно по определению:

```
vandermonde[x_,n_]:=Table[x[j]^i,{i,0,n-1},{j,1,n}]
```

Обратите внимание на порядок итераторов!

**1.4.** Задайте **матрицу Коши**:

$$\begin{pmatrix} \dfrac{1}{x_1+y_1} & \dfrac{1}{x_1+y_2} & \ldots & \dfrac{1}{x_1+y_n} \\ \dfrac{1}{x_2+y_1} & \dfrac{1}{x_2+y_2} & \ldots & \dfrac{1}{x_2+y_n} \\ \vdots & \vdots & \ddots & \vdots \\ \dfrac{1}{x_n+y_1} & \dfrac{1}{x_n+y_1} & \ldots & \dfrac{1}{x_n+y_n} \end{pmatrix}.$$

Найдите обратную к ней для небольших $n$.

**Ответ.** И здесь непосредственно по определению:

```
cauchy[x_,y_,n_]:=Table[1/(x[i]+y[j]),{i,1,n},{j,1,n}]
```

**1.5.** Задайте матрицу **одномерного преобразования**:

$$\begin{pmatrix} 1+x_1y_1 & x_1y_2 & \ldots & x_1y_n \\ x_2y_1 & 1+x_2y_2 & \ldots & x_2y_n \\ \vdots & \vdots & \ddots & \vdots \\ x_ny_1 & x_ny_1 & \ldots & 1+x_ny_n \end{pmatrix}.$$

Найдите обратную к ней для небольших $n$.

**Ответ.** Снова без всяких хитростей:

```
dilation[x_,y_,n_]:=
                  Table[If[i==j,1,0]+x[i]*y[j],{i,1,n},{j,1,n}]
```

Стоит обратить внимание, что геометрически название не совсем точно. Конечно, *в общем положении*, когда $yx \neq 0$, это преобразование действительно будет дилацией (псевдоотражением). Однако, в случае, когда $yx = 0$, эта формула задает трансвекцию, а не псевдоотражение.

**1.6.** Задайте следующую матрицу:

$$\begin{pmatrix} 1+x_1y_1 & 1+x_1y_2 & \dots & 1+x_1y_n \\ 1+x_2y_1 & 1+x_2y_2 & \dots & 1+x_2y_n \\ \vdots & \vdots & \ddots & \vdots \\ 1+x_ny_1 & 1+x_ny_1 & \dots & 1+x_ny_n \end{pmatrix}.$$

Будет ли она обратимой для каких-то $n$, $x_i$ и $y_i$?

**1.7.** Задайте следующую матрицу:

$$\begin{pmatrix} -x_1 & 1 & 0 & \dots & 0 & 0 \\ 1 & -x_2 & 1 & \dots & 0 & 0 \\ 0 & 1 & -x_3 & \ddots & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \ddots & -x_{n-1} & 1 \\ 0 & 0 & 0 & \dots & 1 & -x_n \end{pmatrix}.$$

Выясните, будет ли она обратимой и, если да, то найдите обратную к ней для небольших $n$.

**1.8.** Задайте следующую матрицу:

$$\begin{pmatrix} x_1 & 1 & 0 & \dots & 0 & 0 \\ -1 & x_2 & 1 & \dots & 0 & 0 \\ 0 & -1 & x_3 & \ddots & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \ddots & x_{n-1} & 1 \\ 0 & 0 & 0 & \dots & -1 & x_n \end{pmatrix}.$$

Выясните, будет ли она обратимой и, если да, то найдите обратную к ней для небольших $n$.

**1.9.** Задайте следующую матрицу:

$$\begin{pmatrix} y_1 & x_1 & 0 & \dots & 0 & 0 \\ 1 & y_2 & x_2 & \dots & 0 & 0 \\ 1 & 1 & y_3 & \ddots & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 1 & 1 & 1 & \ddots & y_n & x_n \\ 1 & 1 & 1 & \dots & 1 & y_{n+1} \end{pmatrix}.$$

Выясните, будет ли она обратимой и, если да, то найдите обратную к ней для небольших $n$.

**2. Переключатели.**

Матрица $x = (x_{ij}) \in M(n, R)$ вида

$$\begin{pmatrix} a_0 & a_1 & a_2 & \dots & a_{n-1} \\ a_{n-1} & a_0 & a_1 & \dots & a_{n-2} \\ a_{n-2} & a_{n-1} & a_0 & \dots & a_{n-3} \\ \dots & \dots & \dots & \dots & \dots \\ a_1 & a_2 & a_3 & \dots & a_0 \end{pmatrix}.$$

называется **циркулянтом**. Иными словами, $x_{ij}$ зависит не от самих индексов $i, j$, а только от вычета их разности $i - j$ по модулю $n$.

**2.1.** Для всех $n$ задайте циркулянт и для нескольких небольших $n$ найдите матрицу, обратную к нему.

**Решение.** Очевидный способ состоит в том, чтобы явно задать общий элемент циркулянта:

```
circ[a_,n_]:=Table[a[Mod[j-i,n]],{i,1,n},{j,1,n}]
```

Вот еще один изящный способ задавать циркулянт с помощью команды `MapIndexed`:

```
MapIndexed[RotateRight,Table[{a[1],a[2],...,a[n-1],a[0]},{5}]]
```

Требуемый сдвиг по отношению к исходному списку можно обеспечить либо предварительным применением `RotateLeft`

```
MapIndexed[RotateRight,Table[RotateLeft[x],{Length[x]}]]
```

Либо изменением на 1 степени `RotateRight`

```
MapIndexed[RotateRight[#1,#2-1]&,Table[x,{Length[x]}]]
```

Матрица $x = (x_{ij}) \in M(n, R)$ вида

$$\begin{pmatrix} a_0 & a_1 & a_2 & \dots & a_{n-1} \\ -a_{n-1} & a_0 & a_1 & \dots & a_{n-2} \\ -a_{n-2} & -a_{n-1} & a_0 & \dots & a_{n-3} \\ \dots & \dots & \dots & \dots & \dots \\ -a_1 & -a_2 & -a_3 & \dots & a_0 \end{pmatrix}.$$

называется **антициркулянтом**. Иными словами, $(-1)^{\mathrm{sign}(j-i)} x_{ij}$ зависит не от самих индексов $i, j$, а только от вычета их разности $i - j$ по модулю $n$.

**2.2.** Для всех $n$ задайте антициркулянт и для нескольких небольших $n$ найдите матрицу, обратную к нему.

Матрица $x = (x_{ij}) \in M(n, R)$ вида

$$\begin{pmatrix} a_0 & a_1 & a_2 & \dots & a_{n-1} \\ da_{n-1} & a_0 & a_1 & \dots & a_{n-2} \\ da_{n-2} & da_{n-1} & a_0 & \dots & a_{n-3} \\ \dots & \dots & \dots & \dots & \dots \\ da_1 & da_2 & da_3 & \dots & a_0 \end{pmatrix}.$$

называется **квазициркулянтом**.

**2.3.** Для всех $n$ задайте квазициркулянт и для нескольких небольших $n$ найдите матрицу, обратную к нему.

**2.4.** Матрица $x = (x_{ij})$ называется **теплицевой**, если ее элемент $x_{ij}$ в позиции $(i, j)$ зависит не от самих $i$ и $j$, а только от их разности $i - j$. Традиционно считают, что для любого $h = -(n-1), \dots, -1, 0, 1, \dots, n-1$ определен элемент $x_h \in K$ и полагают $x_{ij} = x_{i-j}$. Изобразим для примера теплицеву матрицу степени 5:

$$\begin{pmatrix} x_0 & x_{-1} & x_{-2} & x_{-3} & x_{-4} \\ x_1 & x_0 & x_{-1} & x_{-2} & x_{-3} \\ x_2 & x_1 & x_0 & x_{-1} & x_{-2} \\ x_3 & x_2 & x_1 & x_0 & x_{-1} \\ x_4 & x_3 & x_2 & x_1 & x_0 \end{pmatrix}$$

Задайте теплицеву матрицу произвольного размера и вычислите ее определитель для нескольких небольших $n$.

**2.5.** Матрица $x = (x_{ij})$ называется **ганкелевой**, если ее элемент $x_{ij}$ в позиции $(i, j)$ зависит не от самих $i$ и $j$, а только от их суммы $i + j$. Иными словами, для любого $h = 2, \dots, 2n$ определен элемент $x_h \in K$ и $x_{ij} = x_{i+j}$. Впрочем, обычно нумерацию $x_h$ начинают не с $x_2$, а с $x_0$, так что $x_{ij} = x_{i+j-2}$. Изобразим для примера ганкелеву матрицу степени 5:

$$\begin{pmatrix} x_0 & x_1 & x_2 & x_3 & x_4 \\ x_1 & x_2 & x_3 & x_4 & x_5 \\ x_2 & x_3 & x_4 & x_5 & x_6 \\ x_3 & x_4 & x_5 & x_6 & x_7 \\ x_4 & x_5 & x_6 & x_7 & x_8 \end{pmatrix}$$

Задайте ганкелеву матрицу произвольного размера и вычислите ее определитель для небольших $n$.

**3. Симметрия.** В данном пункте мы изучим применение к матрице преобразований `Reverse` и `Transpose`.

| | |
|---|---|
| `Reverse[x]` | инвертировать список $x$ |
| `Transpose[x]` | транспонировать список $x$ |

Преобразование `Reverse` переставляет начало и конец списка, в данном случае первую строчку с последней, вторую с предпоследней и т.д. По умолчанию `Transpose` меняет первые два уровня вложенности списка. Тем самым, для матриц оно переставляет строки и столбцы. Эти преобразования порождают диэдральную группу $D_4$, таким образом, комбинируя их мы можем получить восемь различных преобразований матриц.

**3.1.** Задайте матрицу, транспонированную к матрице $x$ относительно побочной диагонали.

**Ответ.** Проще всего так:

```
Reverse[Transpose[Reverse[x]]]
```

**3.2.** Задайте матрицу, получающуюся из матрицы $x$ отражением относительно горизонтальной оси симметрии.

**Ответ.** Просто `Reverse[x]`.

**3.3.** Задайте матрицу, получающуюся из матрицы $x$ отражением относительно вертикальной оси симметрии.

**Ответ.** Можно превратить строки в столбцы, отразить относительно горизонтальной оси симметрии и снова превратить столбцы в строки:

```
Transpose[Reverse[Transpose[x]]]
```

Эта конструкция будет встречаться нам настолько часто, что в дальнейшем обычно мы приводим только решение для строк, подразумевая, что дважды применяя `Transpose`, мы получим решение для столбцов.

**3.4.** Задайте матрицу, получающуюся из матрицы $x$ поворотом относительно центра по часовой стрелке на $90^o$.

**Ответ.** Вначале обращаем порядок строк, потом превращаем строки в столбцы:

```
Transpose[Reverse[x]]
```

При этом первая строка становится последним столбцом.

**3.5.** Задайте матрицу, получающуюся из матрицы $x$ поворотом относительно центра против часовой стрелке на $90^o$.

**Ответ.** Естественно, `Reverse[Transpose[x]]`. Здесь мы сначала превращаем строки в столбцы, а потом образаем порядок строк. При этом первая строка становится первым столбцом.

Теперь каждый, кто слышал, что такое диэдральная группа, готов завершить процесс.

**3.6.** Задайте матрицу, получающуюся из матрицы $x$ отражением относительно центра.

**Ответ.** Естественно,

```
Reverse[Transpose[Reverse[Transpose[x]]]]
```

или, *что то же самое*,

```
Transpose[Reverse[Transpose[Reverse[x]]]]
```

Соберем теперь вместе все восемь элементов лиэдральной группы $D_4$:

```
dihedral={Identity,Transpose,Reverse,
          Composition[Reverse,Transpose],
          Composition[Transpose,Reverse],
          Composition[Transpose,Reverse,Transpose],
          Composition[Reverse,Transpose,Reverse],
          Composition[Reverse,Transpose,Reverse,Transpose]}
```

Их применение к матрице $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ посредством

```
Map[MatrixForm,Through[dihedral[{{a,b},{c,d}}]]]
```

даст следующие восемь матриц:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \quad \begin{pmatrix} a & c \\ b & d \end{pmatrix} \quad \begin{pmatrix} c & d \\ a & b \end{pmatrix} \quad \begin{pmatrix} b & d \\ a & c \end{pmatrix}$$

$$\begin{pmatrix} c & a \\ d & b \end{pmatrix} \quad \begin{pmatrix} b & a \\ d & c \end{pmatrix} \quad \begin{pmatrix} d & b \\ c & a \end{pmatrix} \quad \begin{pmatrix} d & c \\ b & a \end{pmatrix}$$

**4. Окаймление.**

**4.1.** Задайте следующую матрицу:

$$\begin{pmatrix} 1 & x_1 & \dots & x_n \\ y_1 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ y_n & 0 & \dots & 1 \end{pmatrix}.$$

Выясните, будет ли она обратимой и, если да, то найдите обратную к ней для небольших $n$.

**4.2.** Задайте квадратную матрицу порядка $n$, в которой натуральные числа от 1 до $n^2$ расположены по спирали в направлении *по часовой стрелке*, начинающейся в *левом верхнем* углу. Вот примеры таких матриц для $n = 2, 3, 4, 5$:

$$\begin{matrix} 1 & 2 \\ 4 & 3 \end{matrix} \qquad \begin{matrix} 1 & 2 & 3 \\ 8 & 9 & 4 \\ 7 & 6 & 5 \end{matrix} \qquad \begin{matrix} 1 & 2 & 3 & 4 \\ 12 & 13 & 14 & 5 \\ 11 & 16 & 15 & 6 \\ 10 & 9 & 8 & 7 \end{matrix} \qquad \begin{matrix} 1 & 2 & 3 & 4 & 5 \\ 16 & 17 & 18 & 19 & 6 \\ 15 & 24 & 25 & 20 & 7 \\ 14 & 23 & 22 & 21 & 8 \\ 13 & 12 & 11 & 10 & 9 \end{matrix}$$

Будет ли эта матрица обратимой и, если да, то найдите обратную к ней для небольших $n$.

**Ответ.** Например, при помощи следующего рекуррентноого определения. Здесь мы явно окаймляем матрицу порядка $n-2$.

```
spiral[1]={{1}}; spiral[2]={{1,2},{4,3}};
spiral[n_]:=Join[{Range[n]},
                 Transpose[Join[{Reverse[Range[3n-1,4n-4]]},
                   Transpose[spiral[n-2]+4*n-4],
                   {Range[n+1,2*n-2]}]],
                 {Reverse[Range[2*n-1,3*n-2]]}]
```

Если Вы правильнов ввели этот код, то теперь вычисление `spiral[10] // TableForm` должно дать

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 11 |
| 35 | 64 | 65 | 66 | 67 | 68 | 69 | 70 | 45 | 12 |
| 34 | 63 | 84 | 85 | 86 | 87 | 88 | 71 | 46 | 13 |
| 33 | 62 | 83 | 96 | 97 | 98 | 89 | 72 | 47 | 14 |
| 32 | 61 | 82 | 95 | 100 | 99 | 90 | 73 | 48 | 15 |
| 31 | 60 | 81 | 94 | 93 | 92 | 91 | 74 | 49 | 16 |
| 30 | 59 | 80 | 79 | 78 | 77 | 76 | 75 | 50 | 17 |
| 29 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 18 |
| 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 |

Можно, конечно, было бы окаймлять матрицу порядка $n-1$, но тогда эти матрицы должны попеременно начинаться в левом верхнем и правом нижнем углу, соответственно, `as follows`.

**4.3.** Задайте квадратную матрицу порядка $n$, в которой натуральные числа от 1 до $n^2$ расположены по спирали в направлении *по часовой стрелке*, начинающейся в *правом нижнем* углу. Вот примеры таких матриц для $n = 2, 3, 4, 5$:

| | |
|---|---|
| 3 | 4 |
| 2 | 1 |

| | | |
|---|---|---|
| 5 | 6 | 7 |
| 4 | 9 | 8 |
| 3 | 2 | 1 |

| | | | |
|---|---|---|---|
| 7 | 8 | 9 | 10 |
| 6 | 15 | 16 | 11 |
| 5 | 14 | 13 | 12 |
| 4 | 3 | 2 | 1 |

| | | | | |
|---|---|---|---|---|
| 9 | 10 | 11 | 12 | 13 |
| 8 | 21 | 22 | 23 | 14 |
| 7 | 20 | 25 | 24 | 15 |
| 6 | 19 | 18 | 17 | 16 |
| 5 | 4 | 3 | 2 | 1 |

**4.4.** Задайте квадратную матрицу порядка $n$, в которой натуральные числа от 1 до $n^2$ расположены по спирали в направлении *против часовой стрелки*, начинающейся *правом нижнем* углу. Вот примеры таких матриц для $n = 3, 4, 5$:

| | |
|---|---|
| 3 | 2 |
| 4 | 1 |

| | | |
|---|---|---|
| 5 | 4 | 3 |
| 6 | 9 | 2 |
| 7 | 8 | 1 |

| | | | |
|---|---|---|---|
| 7 | 6 | 5 | 4 |
| 8 | 15 | 14 | 3 |
| 9 | 16 | 13 | 2 |
| 10 | 11 | 12 | 1 |

| | | | | |
|---|---|---|---|---|
| 9 | 8 | 7 | 6 | 5 |
| 10 | 21 | 20 | 19 | 4 |
| 11 | 22 | 25 | 18 | 3 |
| 12 | 23 | 24 | 17 | 2 |
| 13 | 14 | 15 | 16 | 1 |

Будет ли эта матрица обратимой и, если да, то найдите обратную к ней для небольших $n$.

**4.5.** Рассмотрите аналогичную задачу для пяти осташихся случаев:

- направление *против часовой стрелки*, начало в *левом нижнем* углу;
- направление *по часовой стрелки*, начало в *правом верхнем* углу;
- направление *против часовой стрелки*, начало в *правом верхнем* углу;
- направление *по часовой стрелки*, начало в *левом нижнем* углу;
- направление *против часовой стрелки*, начало в *левом нижнем* углу.

**5. Вычеркивание.**

**5.1.** Задайте матрицу, получающуюся из $x$ вычеркиванием $i$-й строки и $j$-го столбца.

**Ответ.** Ясно, что $i$-я строка вычеркивается при помощи `Drop`, а вычеркивание $j$-го столбца осуществляется точно так же, как и вычеркивание $j$-й строки, в сочетании с двойным применением `Transpose`:

```
coma[i_,j_][x_]:=Transpose[Drop[Transpose[Drop[x,{i}]],{j}]]
```

**6. Блочные матрицы.**

**6.1.** Напишите программу, которая по двум квадратным матрицам $a, b \in M(n, K)$ строит матрицу

$$\begin{pmatrix} e & a \\ -b & 0 \end{pmatrix}$$

размера $2n$, где $e \in M(n, K)$ — единичная матрица.

**6.2.** Напишите программу, конвертирующую блочную матрицу в обычную матрицу.

**Решение.** Аллан Хайес на форуме `MathGroup` предложил следующие два решения этой задачи:

```
Apply[Join,Apply[Join,Transpose[mat,{1,3,2}]],1]
Flatten[Map[Flatten,Transpose[mat,{1,3,2}],{2}],1]
```

Обратите внимание, что оба эти решения используют одну и ту же перестановку уровней.

**6.3.** Задайте матрицу

$$\begin{pmatrix} \begin{pmatrix} a & a & a \\ a & a & a \\ a & a & a \end{pmatrix} & \begin{pmatrix} b & b & b \\ b & b & b \\ b & b & b \end{pmatrix} & \begin{pmatrix} c & c & c \\ c & c & c \\ c & c & c \end{pmatrix} \\ \begin{pmatrix} d & d & d \\ d & d & d \\ d & d & d \end{pmatrix} & \begin{pmatrix} e & e & e \\ e & e & e \\ e & e & e \end{pmatrix} & \begin{pmatrix} f & f & f \\ f & f & f \\ f & f & f \end{pmatrix} \\ \begin{pmatrix} g & g & g \\ g & g & g \\ g & g & g \end{pmatrix} & \begin{pmatrix} h & h & h \\ h & h & h \\ h & h & h \end{pmatrix} & \begin{pmatrix} i & i & i \\ i & i & i \\ i & i & i \end{pmatrix} \end{pmatrix}$$

**Решение.** Нам кажется, что проще всего применить `Table[#,{3},{3}]` к элементам матрицы `{{a,b,c},{d,e,f},{g,h,i}}`:

```
Map[Table[#,{3},{3}]&,
Partition[ToExpression[CharacterRange["a", "i"]],3],{2}]
```

**6.4.** А теперь без компьютера скажите, что получится при вычислении

```
Map[Table[#,{3},{3}]&,
Partition[ToExpression[CharacterRange["a","i"]],3]]
```

без указания уровня?

**Решение.** По умолчанию `Map` применит `Table` не на уровне 2, а на уровне 1, иными словами, не к элементам, а к строчкам. Это значит, что 9 раз повторятся не элементы исходной матрицы, а ее строчки:

$$\begin{pmatrix} \begin{pmatrix} a & b & c \\ a & b & c \\ a & b & c \end{pmatrix} & \begin{pmatrix} a & b & c \\ a & b & c \\ a & b & c \end{pmatrix} & \begin{pmatrix} a & b & c \\ a & b & c \\ a & b & c \end{pmatrix} \\ \begin{pmatrix} d & e & f \\ d & e & f \\ d & e & f \end{pmatrix} & \begin{pmatrix} d & e & f \\ d & e & f \\ d & e & f \end{pmatrix} & \begin{pmatrix} d & e & f \\ d & e & f \\ d & e & f \end{pmatrix} \\ \begin{pmatrix} g & h & i \\ g & h & i \\ g & h & i \end{pmatrix} & \begin{pmatrix} g & h & i \\ g & h & i \\ g & h & i \end{pmatrix} & \begin{pmatrix} g & h & i \\ g & h & i \\ g & h & i \end{pmatrix} \end{pmatrix}$$

Выше мы задали кронекеровское произведение матрицы $\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}$ с пробной матрицей:

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \otimes \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}.$$

**6.5.** Задайте кронекеровское произведение

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \otimes \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}.$$

Обратимся теперь к заданию кронекерова произведения в общем случае. Задать кронекеровское произведение

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \otimes \begin{pmatrix} e & f \\ g & h \end{pmatrix} = \begin{pmatrix} a\begin{pmatrix} e & f \\ g & h \end{pmatrix} & b\begin{pmatrix} e & f \\ g & h \end{pmatrix} \\ c\begin{pmatrix} e & f \\ g & h \end{pmatrix} & d\begin{pmatrix} e & f \\ g & h \end{pmatrix} \end{pmatrix}$$

*как блочную матрицу* совсем легко.

**6.6.** Задайте кронекерово произведение двух матриц как блочную матрицу.

**Решение.** Ровно для этого и служит внутренняя функция `Outer`! Естественно, это просто `Outer[Times,x,y]`.

Попытаемся теперь задать кронекерово произведение как обычную матрицу, без дополнительных уровней вложенности.

**6.7.** Задайте кронекерово произведение двух матриц $x$ и $y$ как обычную матрицу.

**Решение.** Можно конечно, найти их произведение как блочную матрицу, а потом убрать лишние уровни вложенности так, как мы делали это выше. Но в действительности, так как теперь матрица $x$ имеется в нашем распоряжении с самого начала, чуть проще протранспонировать ее сразу, чтобы потом не приходилось судорожно считать уровни:

```
KroneckerProduct[x_,y_]:=Flatten[
          MapThread[Join,Outer[Times,Transpose[x],y],2],1]
```

**6.8.** Задайте кронекерову сумму двух матриц $x$ и $y$.

**7. Диагонали.**

**7.1.** Задайте квадратную матрицу порядка $n$, в которой натуральные числа от 1 до $n^2$ расположены сверху вних по диагоналям, параллельным побочной. Вот примеры таких матриц для $n = 3, 4, 5$:

$$\begin{matrix} 1 & 2 & 4 \\ 3 & 5 & 7 \\ 6 & 8 & 9 \end{matrix} \qquad \begin{matrix} 1 & 2 & 4 & 7 \\ 3 & 5 & 8 & 11 \\ 6 & 9 & 12 & 14 \\ 10 & 13 & 15 & 16 \end{matrix} \qquad \begin{matrix} 1 & 2 & 4 & 7 & 11 \\ 3 & 5 & 8 & 12 & 16 \\ 6 & 9 & 13 & 17 & 20 \\ 10 & 14 & 18 & 21 & 23 \\ 15 & 19 & 22 & 24 & 25 \end{matrix}$$

Будет ли эта матрица обратимой и, если да, то найдите обратную к ней для небольших $n$.

**Ответ.** В следующем определении мы организуем цикл по номеру диагонали $h$, а внутри него цикл по номеру строки $i$:

```
strips[n_]:=Block[{m=1,glist=Table[0,{n},{n}]},
          For[h=2,h<=2*n,h++,For[i=1,i<=n,i++,
              If[1<=h-i<=n,glist[[i,h-i]]=m;m=m+1]]];glist]
```

**7.2.** Задайте квадратную матрицу порядка $n$, в которой натуральные числа от 1 до $n^2$ расположены в соответствии с диагональным процессом Коши ("змейкой"). Вот примеры таких матриц для $n = 3, 4, 5$:

$$\begin{matrix} 1 & 2 & 6 \\ 3 & 5 & 7 \\ 4 & 8 & 9 \end{matrix} \qquad \begin{matrix} 1 & 2 & 6 & 7 \\ 3 & 5 & 8 & 13 \\ 4 & 9 & 12 & 14 \\ 10 & 11 & 15 & 16 \end{matrix} \qquad \begin{matrix} 1 & 2 & 6 & 7 & 15 \\ 3 & 5 & 8 & 14 & 16 \\ 4 & 9 & 13 & 17 & 22 \\ 10 & 12 & 18 & 21 & 23 \\ 11 & 19 & 20 & 24 & 25 \end{matrix}$$

Будет ли эта матрица обратимой и, если да, то найдите обратную к ней для небольших $n$.

**Ответ.**

```
zigzag[n_]:=Block[{h,i,m=1,glist=Table[0,{n},{n}]},
            For[h=2,h<=2*n,h++,If[OddQ[h],
            For[i=1,i<=n,i++,
                If[1<=h-i<=n,glist[[i,h-i]]=m;m=m+1]],
            For[i=n,i>=1,i--,
                If[1<=h-i<=n,glist[[i,h-i]]=m;m=m+1]]]];glist]
```

## § 3. Определители

Из тяжести недоброй и я когда-нибудь прекрасное создам.

Осип Мандельштам

**1. Определение, свойства и вычисление.**[217,218]

Определителем называется функция от столбцов матрицы $x \in M(n,R)$

$$\det : M(n,R) = R^n \times \ldots \times R^n \longrightarrow R,$$

со свойствами полилинейности, антисимметричности и нормированности.

Сформулируем явно, что значат эти условия.

• **Линейность по $i$-му столбцу** означает, что определитель **аддитивен по столбцу** (при фиксированных остальных)

$$\det(x_1, \ldots, y_i + z_j, \ldots, x_n) = \det(x_1, \ldots, y_i, \ldots, x_n) + \det(x_1, \ldots, z_i, \ldots, x_n).$$

и, кроме того, **однороден степени 1**:

$$\det(x_1, \ldots, \lambda x_i, \ldots, x_n) = \lambda \det(x_1, \ldots, x_i, \ldots, x_n).$$

• **Антисимметричность** означает, что если две строки определителя равны, то определитель равен 0:

$$\det(x_1, \ldots, x_i, \ldots, x_i, \ldots, x_n) = 0$$

• **Нормированность** означает, что определитель единичной матрицы равен 1:

$$\det(e_1, \ldots, e_n) = 1.$$

---

[217] Very many of the familiar processes of mathematics, such as multiplication of large numbers or computation of determinants, can be computed far more expiditiously than allowed by the usual "school" algorithms. ©Peter Borwein, T.Erdelyi

[218] Chemistry is a branch of physics, but it is sufficiently extensive and profound to deserve its traditional role as an independent subject. ©Robert Vein, Paul Dale *Determinants and their applications in mathematical physics.* — Springer-Verlag, Berlin et al., 1999.

В наивных изложениях вместо антисимметричности обычно фигурирует **кососимметричность**:

$$\det(x_1, \dots, x_i, \dots, x_j, \dots, x_n) = -\det(x_1, \dots, x_j, \dots, x_i, \dots, x_n).$$

Ясно, что кососимметричность следует из антисимметричности и линейности. Для этого достаточно разложить определитель

$$\det(x_1, \dots, x_i + x_j, \dots, x_i + x_j, \dots, x_n)$$

воспользовавшись линейностью по $i$-му и $j$-му столбцам. Однако, обратное, вообще говоря, неверно.

Следующее свойство выражает самую суть понятия определителя, его `raison d'être`. Определитель $\det : M(n, R) \longrightarrow R$ является мультипликативным гомоморфизмом. Иными словами, для любых двух матриц $x, y \in M(n, R)$ имеет место равенство

$$\det(xy) = \det(x)\det(y).$$

**1.1.** Напишите рекуррентную программу вычисления определителя по Лапласу, с разложением по строке.

**1.2.** Напишите рекуррентную программу вычисления определителя по Лапласу, с разложением по столбцу.

**Рекомендации по вычислению определителя.** Встретив незнакомый определитель проделайте следующие действия — примерно в таком порядке!

• Вычислите несколько первых значений определителя на компьютере, постарайтесь угадать ответ.

Здесь стоит отметить, что в настоящее время существует несколько замечательных программ для `Mathematica` и `Maple` таких, как `Rate` и `Guess`, которые *угадывают* вид определителя! Разумеется, на самом деле, конечно, в большинстве случаев это угадывание носит вполне прозаический характер. Скажем, эти программы вычисляют значения определителя при различных значениях параметров, а потом восстанавливают общий вид интерполяцией.

• Посмотрите, не является ли определитель частным случаем какого-то из известных определителей.

• Посмотрите, не имеет ли определитель специального вида, такого как симметрический, ганкелев или теплицев, для которого применимы специальные методы.

• Постарайтесь получить рекуррентное соотношение понижая степень определителя при помощи элементарных преобразований над строками и столбцами.

• Попробуйте представить определитель как *сумму* двух определителей более простого вида раскладывая строку или столбец в сумму двух.

• Постарайтесь получить рекуррентное соотношение раскладывая определитель по Лапласу, по одной или нескольким строкам/столбцам.

• Примените метод выделения множителей.

• Попробуйте представить определитель как *произведение* двух определителей более простого вида — скажем, треугольных.

• Постарайтесь получить рекуррентное соотношение при помощи метода конденсации.

• Посмотрите, удовлетворяет ли Ваш определитель простому дифференциальному/разностному уравнению.

Если ни один из этих методов не работает, у нас проблема. В этом случае следует либо немедленно обратиться к доктору, либо рассмотреть более общий определитель — который часто гораздо проще вычислить! — либо вернуться к преобразованиям определителя с геометрической точки зрения. В этом случае профессионал обычно действует следующим образом.

• Вводит дополнительные параметры и смотрит, применяются ли перечисленные выше методы к этому более общему определителю.

• Истолковывает матрицу определителя как матрицу линейного преобразования и старается угадать базис, в котором видны собственные числа этого преобразования, например, его действие треугольно.

• В совершенно безнадежном случае ищет определитель в трактате Мюира[219] — он там есть!!!

**2. Альтернанты.**

Примерно 80% всех определителей, встречающихся в реальных вычислениях, являются альтернантами. Альтернанты производятся в двух основных модификациях, простые и двойные.

Пусть $f_1, \dots, f_n$ — любые функция. Определитель вида

$$\det \begin{pmatrix} f_1(x_1) & f_2(x_1) & \dots & f_n(x_1) \\ f_1(x_1) & f_2(x_1) & \dots & f_n(x_1) \\ \dots & \dots & \dots & \dots \\ f_1(x_n) & f_2(x_n) & \dots & f_n(x_n) \end{pmatrix}$$

называется **простым альтернантом**. Самым известным примером простого альтернанта является определитель Вандермонда.

Пусть $f$ — любая функция. Определитель вида

$$\det \begin{pmatrix} f(x_1, y_1) & f(x_1, y_2) & \dots & f(x_1, y_n) \\ f(x_2, y_1) & f(x_2, y_2) & \dots & f(x_2, y_n) \\ \dots & \dots & \dots & \dots \\ f(x_n, y_1) & f(x_n, y_2) & \dots & f(x_n, y_n) \end{pmatrix}$$

[219] T.Muir, The theory of determinants in the historical order of development. — Macmillan, London, vol. I, Up to 1841. — 1906; vol.II, 1841–1860. — 1911; vol.III, 1861–1880. — 1920; vol.IV, 1880–1900. — 1923.

называется **двойным альтернантом**. Самым известным примером двойного альтернанта является определитель Коши.

Ясно, что двойные альтернанты можно рассматривать как простые, по отношению к парциальным функциям $f_i = f(\_, y_i)$. Единственное различие здесь психологическое и состоит в том, трактуются $y_i$ как параметры или как переменные.

**3. Определитель Вандермонда.**

Самый важный определитель, хочется даже сказать, *единственный* реально возникающий в приложениях определитель — это знаменитый определитель Вандермонда

$$V(x_1, \dots, x_n) = \begin{vmatrix} 1 & 1 & \dots & 1 \\ x_1 & x_2 & \dots & x_n \\ x_1^2 & x_2^2 & \dots & x_n^2 \\ \dots & \dots & \dots & \dots \\ x_1^{n-1} & x_2^{n-1} & \dots & x_n^{n-1} \end{vmatrix}$$

Собственно, не будет большим преувеличением сказать, что вся теория определителей построена главным образом для анализа этого примера, возникающего при решении алгебраических уравнений, в задаче интерполяции, преобразовании Фурье и далее везде.

Формула для определителя Вандермонда хорошо известна. Сейчас мы докажем ее тремя методами: при помощи рекуррентных соотношений, при помощи выделения множителей и при помощи треугольной факторизации:

**3.1.** Докажите, что для любых $x_1, \dots, x_n \in R$ имеет место равенство

$$V(x_1, \dots, x_n) = \prod_{1 \le j < i \le n} (x_i - x_j).$$

**Решение 1.** Доказываем это утверждение индукцией по $n$. В качестве базы индукции возьмем $n = 1$; в этом случае обе части доказываемого равенства равны 1. Для того, чтобы осуществить шаг индукции вычтем в определителе Вандермонда $V(x_1, \dots, x_n)$ из каждой строки, начиная с последней, предыдущий столбец, умноженный на $x_n$. В результате мы получим определитель, последний столбец которого совпадает с $e_1$. Раскладывая получившийся определитель по последнему столбцу, мы видим, что $V(x_1, \dots, x_n)$ равен

$$(-1)^{n-1} \begin{vmatrix} x_1 - x_n & x_2 - x_n & \dots & x_{n-1} - x_n \\ x_1(x_1 - x_n) & x_2(x_2 - x_n) & \dots & x_{n-1}(x_{n-1} - x_n) \\ \dots & \dots & \dots & \dots \\ x_1^{n-2}(x_1 - x_n) & x_2^{n-2}(x_1 - x_n) & \dots & x_{n-1}^{n-2}(x_{n-1} - x_n) \end{vmatrix}$$

Вынося из $i$-го столбца этого определителя $x_i - x_n$, мы получаем рекуррентное соотношение

$$V(x_1, \dots, x_n) = \prod_{1 \le i \le n-1} (x_n - x_i) V(x_1, \dots, x_{n-1}).$$

Осталось вспомнить, что по индукционному предположению

$$V(x_1,\dots,x_{n-1}) = \prod_{1\le j<i\le n-1} (x_i - x_j)$$

что вместе с только что полученным рекуррентным соотношением и дает нам требуемую формулу для $V(x_1,\dots,x_n)$.

**Решение 2.** Обычно она доказывается при помощи элементарных преобразований над строчками и рекуррентных соотношений. На самом деле, профессиональные алгебраисты обычно доказывают ее гораздо проще, примерно следующим образом. В самом деле, определитель Вандермонда является многочленом степени $\le 1+2+\ldots+(n-1) = n(n-1)/2$ от $x_1,\dots,x_n$. С другой стороны, если $x_i = x_j$, то определитель Вандермонда имеет два одинаковых столбца и, таким образом, обращается в 0. Это значит, что он делится на $x_j - x_i$. Так как многочлены $x_j - x_i$, $1 \le i < j \le n$, попарно взаимно просты, то он делится на их произведение $\prod x_j - x_i$, $1 \le i < j \le n$, также имеющую степень $n(n-1)/2$. Это значит, что нам остается лишь сравнить коэффициенты этих многочленов при каком-то одночлене, скажем при $x_1^0 x_2^1 x_3^2 \dots x_n^{n-1}$.

Проанализируем приведенное доказательство. Оно состоит из следующих этапов.

- Сведение определителя к полиномиальному виду.
- Выделение неприводимых множителей.
- Сравнение степеней.
- Сравнение коэффициентов.

В практических ситуациях часто этим шагам должен предшествовать еще один шаг, а именно,

- введение новых переменных или параметров.

Определитель с большим количеством параметров вычислить обычно проще или *гораздо* проще. Всегда вводите независимые переменные .

**Решение 3.** Вандермонд методом $LU$-факторизации.

$$\begin{pmatrix} 1 & 1 \\ x & y \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ x & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 0 & y-x \end{pmatrix}$$

$$\begin{pmatrix} 1 & 1 & 1 \\ x & y & z \\ x^2 & y^2 & z^2 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ x & 1 & 0 \\ x^2 & x+y & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 \\ 0 & y-x & z-x \\ 0 & 0 & (z-x)(z-y) \end{pmatrix}$$

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ w & x & y & z \\ w^2 & x^2 & y^2 & z^2 \\ w^3 & x^3 & y^3 & z^3 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ w & 1 & 0 & 0 \\ w^2 & w+x & 1 & 0 \\ w^3 & w^2+wx+x^2 & w+x+y & 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & x-w & y-w & z-w \\ 0 & 0 & (y-w)(y-x) & (z-w)(z-x) \\ 0 & 0 & 0 & (z-w)(z-x)(z-y) \end{pmatrix}$$

В действительности, как правило важна не явная формула для определителя Вандермонда, а только следующий факт.

**3.2.** Если $x_1, \dots, x_n$ — попарно различные элементы области целостности $R$, то $V(x_1, \dots, x_n) \neq 0$.

**4. Определитель Коши.**

Следующий определитель, известный как **(двойной) альтернант Коши**, — а среди широких народных масс просто как **определитель Коши** — играет большую роль в различных проблемах теории аппроксимации, в частности, в теореме Мюнтца:

$$\det \begin{pmatrix} \dfrac{1}{x_1+y_1} & \dfrac{1}{x_1+y_2} & \dots & \dfrac{1}{x_1+y_n} \\ \dfrac{1}{x_2+y_1} & \dfrac{1}{x_2+y_2} & \dots & \dfrac{1}{x_2+y_n} \\ \dots & \dots & \dots & \dots \\ \dfrac{1}{x_n+y_1} & \dfrac{1}{x_n+y_2} & \dots & \dfrac{1}{x_n+y_n} \end{pmatrix}.$$

**4.1.** Вычислите несколько альтернантов Коши и угадайте общую формулу.

**Ответ.** Ограничимся ответом для $n = 2, 3$:

$$\det \begin{pmatrix} \dfrac{1}{x+u} & \dfrac{1}{x+v} \\ \dfrac{1}{y+u} & \dfrac{1}{y+v} \end{pmatrix} = \frac{(u-v)(x-y)}{(u+x)(v+x)(u+y)(v+y)}.$$

$$\det \begin{pmatrix} \dfrac{1}{x+u} & \dfrac{1}{x+v} & \dfrac{1}{x+w} \\ \dfrac{1}{y+u} & \dfrac{1}{y+v} & \dfrac{1}{y+w} \\ \dfrac{1}{z+u} & \dfrac{1}{z+v} & \dfrac{1}{z+w} \end{pmatrix} = \frac{(u-v)(u-w)(v-w)(x-y)(x-z)(y-z)}{(u+x)(v+x)(w+x)(u+y)(v+y)(w+y)(u+z)(v+z)(w+z)}.$$

Теперь ясно, что этот определитель легко выразить в замкнутой форме.

**4.2.** Докажите, что

$$\det_{1\le i,j\le n} \left(\frac{1}{x_i+y_j}\right) = \frac{\prod\limits_{i<j}(x_j-x_i)(y_j-y_i)}{\prod\limits_{i,j}(x_i+y_j)}.$$

**Решение 1.** Вычтите последнюю строку из всех остальных строк, вынесите общие множители из строк и столбцов, после чего вычтите последний столбец из всех остальных столбцов.

**Решение 2.** Приведите элементы матрицы Коши к общему знаменателю, который очевидно равен $\prod_{i,j}(x_i+y_j)$. Ясно, что определитель Коши обращается в 0 при $x_i=x_j$, так как в этом случае у него две равные строчки, и при $y_i=y_j$, так как в этом случае у него два равных столбца. Так как все эти многочлены взаимно просты, то определитель Коши делится на $\prod_{i<j}(x_j-x_i)(y_j-y_i)$. Осталось заметить, что степени левой и правой части равны $-n$ и сравнить какие-нибудь коэффициенты.

В статье[220] проведено вычисление определителя, который является совместным обобщением двойного альтернанта Коши и определителя Вандермонда.

В статье[221] вычисляется следующий определитель.

$$\det\begin{pmatrix} \dfrac{1}{(x_1-y_1)^2} & \dfrac{1}{(x_1-y_2)^2} & \cdots & \dfrac{1}{(x_1-y_n)^2} \\ \dfrac{1}{(x_2-y_1)^2} & \dfrac{1}{(x_2-y_2)^2} & \cdots & \dfrac{1}{(x_2-y_n)^2} \\ \cdots & \cdots & \cdots & \cdots \\ \dfrac{1}{(x_n-y_1)^2} & \dfrac{1}{(x_n-y_2)^2} & \cdots & \dfrac{1}{(x_n-y_n)^2} \end{pmatrix}$$

**4.3.** Убедитесь, что это определитель равен

$$\frac{\prod_{i<j}(x_j-x_i)(y_j-y_i)}{\prod_{i,j}(x_i-y_j)}\,\mathrm{per}_{1\le i,j\le n}\left(\frac{1}{x_i-y_j}\right).$$

Важнейшим частным случаем матрицы Коши является **матрица Гильберта**, для которой общий элемент равен $1/(i+j-1)$. Матрицы Гильберта совершенно замечательны тем, что их определители являются египетскими дробями, а обратные к ним целочисленные.

---

[220] E.L.Basor, P.J.Forrester, Formulas for the evaluation of Toeplitz determinants with rational generating functions. — Math. Nachrichten, 1994, vol.170, p.5–18.

[221] C.W.Borchardt, Bestimmung der symmetrischen Verbindungen ihrer erzeugenden Funktion. – Crelle J., 1855, Bd.53, S.193–198.

# ЛИТЕРАТУРА

> Read, read, read, read, my unlearned reader! read, — or, by the knowledge of the great saint Paraleipomenon — I tell you beforehand, you had better throw down this book at once; for without *much reading*, by which your Reverence knows I mean *much knowledge*, you will not be able to penetrate the moral of the next marbled page (motley emblem of my work!)
>
> Laurence Sterne, *Tristram Shandy*


[1] Вавилов Н.А., Халин В.Г., `Mathematica` *5.* для нематематика. Вып.* 1-2., ОЦЭиМ, СПб, 2005, pp. 1–317.

[2] Вавилов Н.А., Иванов О.А., Лушникова Г.А., Халин В.Г., *Уроки математики при помощи* `Mathematica`., ОЦЭиМ, СПб, 2008, pp. 1–146.

[3] Волков В.А., Халин В.Г., Черняев П.К., и д.р., *Учебные и контрольные задания по математике. Математический анализ. Учебное пособие. 3 изд. испр. и доп.*, ЭФ СПбГУ, СПб, 2010, pp. 1–112.

[4] Вавилов Н.А., Халин В.Г., *Дополнительные задачи по курсу Математика и компьютер.–Учебное издание.* , ОЦЭиМ, СПб, 2006, pp. 1–172.

[5] Вавилов Н.А., Халин В.Г., *Задачи по курсу Математика и компьютер,–Выпуск 1. Арифметика и теория чисел. Учебное издание.*, ОЦЭиМ, СПб, 2006, pp. 1–180.

[6] Вавилов Н.А., Халин В.Г., *Задачи по курсу Математика и компьютер,–Выпуск 2. Комбинаторика и дискретная математика. Учебное издание.*, ОЦЭиМ, СПб, 2007, pp. 1–207.

[7] Вавилов Н.А., Халин В.Г., *Задачи по курсу Математика и компьютер,–Выпуск 3. Алгебра многочленов. Учебное издание.*, ОЦЭиМ, СПб, 2008, pp. 1–204.

[8] Вавилов Н.А., Семенов А.А., Халин В.Г., *Задачи по алгебре. Линейная алгебра.*, ОЦЭиМ, СПб, 2003, pp. 1–52.

[9] Вавилов Н.А., *Не совсем наивная линейная алгебра. II. Алгебра матриц.*, ОЦЭиМ, СПб, 2006, pp. 1–232.