

Содержание

Об авторе	15
Об изображении на обложке	15
Предисловие	17
Введение	19
Для кого написана эта книга	19
Что нужно знать	20
Как организована эта книга	20
Подготовка	21
Ответы к упражнениям	25
Соглашения, принятые в этой книге	25
Примеры исходного кода	26
Благодарности	27
Ждем ваших отзывов!	30
Глава 1. Конечные поля	31
Высшая математика	31
Определение конечного поля	32
Определение конечных множеств	33
Построение конечного поля на языке Python	34
Упражнение 1	36
Арифметика по модулю	36
Арифметика по модулю на языке Python	38
Сложение и вычитание конечных полей	39
Упражнение 2	40
Программная реализация операций сложения и вычитания на языке Python	40
Упражнение 3	41
Операции умножения и возведения в степень в конечном поле	41

Упражнение 4	43
Упражнение 5	43
Программная реализация операции умножения на языке Python	43
Упражнение 6	44
Программная реализация операции возведения в степень на языке Python	44
Упражнение 7	45
Операция деления в конечном поле	45
Упражнение 8	48
Упражнение 9	48
Переопределение операции возведения в степень	48
Заключение	49
Глава 2. Эллиптические кривые	51
Определение эллиптических кривых	51
Программная реализация эллиптических кривых на языке Python	57
Упражнение 1	58
Упражнение 2	58
Сложение точек	58
Математические основы сложения точек	62
Программная реализация операции сложения точек	64
Упражнение 3	66
Сложение точек, когда $x_1 \neq x_2$	66
Упражнение 4	68
Программная реализация операции сложения точек, когда $x_1 \neq x_2$	68
Упражнение 5	68
Сложение точек, когда $P_1 = P_2$	68
Упражнение 6	70
Программная реализация операции сложения точек, когда $P_1 = P_2$	70
Упражнение 7	71
Программная реализация операции сложения точек в еще одном исключительном случае	71
Заключение	72
Глава 3. Криптография по эллиптическим кривым	73
Эллиптические кривые над вещественными числами	73
Эллиптические кривые над конечными полями	75
Упражнение 1	76

Программная реализация эллиптических кривых над конечными полями	76
Сложение точек над конечными полями	78
Программная реализация операции сложения точек над конечными полями	79
Упражнение 2	80
Упражнение 3	80
Скалярное умножение для эллиптических кривых	80
Упражнение 4	82
Еще раз о скалярном умножении	83
Математические группы	84
Тождественность	85
Замкнутость	85
Обратимость	86
Коммутативность	87
Ассоциативность	87
Упражнение 5	87
Программная реализация скалярного умножения	88
Определение кривой для биткойна	91
Работа с кривой secp256k1	93
Криптография с открытым ключом	95
Подписание и верификация	95
Надписание цели	96
Подробнее о верификации	99
Верификация подписи	101
Упражнение 6	102
Программная реализация верификации подписей	102
Подробнее о подписании	103
Создание подписи	103
Упражнение 7	105
Программная реализация подписания сообщений	105
Заключение	108
Глава 4. Сериализация	109
Несжатый формат SEC	109
Упражнение 1	111
Сжатый формат SEC	111
Упражнение 2	116

Подписи в формате DER	116
Упражнение 3	118
Кодировка Base58	118
Передача открытого ключа	119
Упражнение 4	121
Формат адреса	121
Упражнение 5	122
Формат WIF	123
Упражнение 6	124
Еще раз о прямом и обратном порядке следования байтов	124
Упражнение 7	125
Упражнение 8	125
Упражнение 9	125
Заключение	125
Глава 5. Транзакции	127
Составляющие транзакции	127
Версия	130
Упражнение 1	131
Вводы	131
Синтаксический анализ сценариев	137
Упражнение 2	137
Выводы	137
Упражнение 3	139
Время блокировки	140
Упражнение 4	141
Упражнение 5	141
Программная реализация транзакций	142
Плата за транзакцию	143
Расчет платы за транзакцию	145
Упражнение 6	145
Заключение	145
Глава 6. Язык Script	147
Внутренний механизм Script	147
Принцип действия языка Script	149
Примеры операций	150
Программная реализация операций по их кодам	151

Упражнение 1	152
Синтаксический анализ полей сценариев	152
Программная реализация синтаксического анализатора и сериализатора сценариев	153
Объединение полей сценариев	156
Программная реализация объединенного набора команд	156
Стандартные сценарии	157
p2pk	157
Программная реализация вычисления сценариев	161
Внутреннее представление элементов в стеке	163
Упражнение 2	165
Затруднения, связанные с p2pk	165
Разрешение затруднений средствами p2pkh	166
p2pkh	167
Построение произвольных сценариев	171
Упражнение 3	174
Польза сценариев	175
Упражнение 4	175
Пиньята для алгоритма SHA-1	175
Заключение	176
Глава 7. Создание и проверка достоверности транзакций	177
Проверка достоверности транзакций	177
Проверка расходования вводов транзакции	178
Проверка суммы вводов относительно суммы выводов транзакции	178
Проверка подписи	180
Упражнение 1	184
Упражнение 2	185
Верификация всей транзакции	185
Создание транзакции	185
Построение транзакции	186
Составление транзакции	189
Подписание транзакции	191
Упражнение 3	192
Создание собственных транзакций в сети testnet	192
Упражнение 4	193
Упражнение 5	193
Заключение	193

Глава 8. Оплата по хешу сценария	195
Простая мультиподпись	196
Программная реализация операции OP_CHECKMULTISIG	200
Упражнение 1	200
Недостатки простой мультиподписи	201
Оплата по хешу сценария	201
Программная реализация p2sh	209
Более сложные сценарии	210
Адреса	210
Упражнение 2	210
Упражнение 3	211
Верификация подписей в p2sh	211
Упражнение 4	214
Упражнение 5	214
Заключение	214
Глава 9. Блоки	215
Монетизирующие транзакции	216
Упражнение 1	217
Сценарий ScriptSig	217
Протокол VIP0034	218
Упражнение 2	219
Заголовки блоков	219
Упражнение 3	220
Упражнение 4	220
Упражнение 5	220
Версия	220
Упражнение 6	222
Упражнение 7	222
Упражнение 8	222
Предыдущий блок	223
Корень дерева Меркла	223
Отметка времени	223
Биты	224
Одноразовый номер	224
Подтверждение работы	224
Каким образом добытчик криптовалюты генерирует хеш-коды	226
Цель	226

Упражнение 9	228
Сложность	228
Упражнение 10	229
Проверка достаточности подтверждения работы	229
Упражнение 11	229
Корректировка сложности	229
Упражнение 12	231
Упражнение 13	232
Заключение	232
Глава 10. Организация сети	233
Сетевые сообщения	233
Упражнение 1	235
Упражнение 2	235
Упражнение 3	235
Синтаксический анализ полезной информации	235
Упражнение 4	237
Подтверждение подключения к сети	237
Подключение к сети	238
Упражнение 5	241
Получение заголовков блоков	241
Упражнение 6	243
Присылаемые в ответ заголовки	243
Заключение	246
Глава 11. Упрощенная проверка оплаты	247
Предпосылки	247
Дерево Меркла	248
Родительский узел дерева Меркла	249
Упражнение 1	250
Родительский уровень дерева Меркла	251
Упражнение 2	252
Корень дерева Меркла	252
Упражнение 3	253
Корень дерева Меркла в блоках	253
Упражнение 4	254
Применение дерева Меркла	254
Древовидный блок Меркла	256

Структура дерева Меркла	258
Упражнение 5	259
Программная реализация дерева Меркла	259
Команда merkleblock	265
Упражнение 6	267
Применение битов признаков и хешей	267
Упражнение 7	272
Заключение	272
Глава 12. Фильтры Блума	273
Что такое фильтр Блума	273
Упражнение 1	276
Продвижение на шаг дальше	276
Фильтры Блума по протоколу VIP0037	277
Упражнение 2	279
Упражнение 3	280
Загрузка фильтра Блума	280
Упражнение 4	280
Получение древовидных блоков Меркла	280
Упражнение 5	281
Получение представляющей интерес транзакции	282
Упражнение 6	283
Заключение	284
Глава 13. Протокол Segwit	285
Оплата по хешу открытого ключа с отдельным заверением	285
Податливость транзакции	286
Устранение податливости транзакций	287
Транзакции по сценарию p2wpkh	287
Сценарий p2sh-p2wpkh	291
Программная реализация сценариев p2wpkh и p2sh-p2wpkh	296
Оплата по хешу сценария с отдельным заверением (p2wsh)	301
Сценарий p2sh-p2wsh	306
Программная реализация сценариев p2wsh и p2sh-p2wsh	311
Прочие усовершенствования	313
Заключение	314

Глава 14. Дополнительные вопросы и следующие шаги	315
Темы для дальнейшего изучения	315
Криптовалютные кошельки	315
Платежные каналы и протокол Lightning Network	317
Участие в разработке биткойна	317
Другие предлагаемые проекты	318
Криптовалютный кошелек в сети testnet	318
Обозреватель блоков	318
Интернет-магазин	318
Служебная библиотека	319
Трудоустройство	319
Заключение	320
 Приложение. Ответы к упражнениям	 321
Глава 1. Конечные поля	321
Глава 2. Эллиптические кривые	325
Глава 3. Криптография по эллиптическим кривым	327
Глава 4. Сериализация	331
Глава 5. Транзакции	335
Глава 6. Язык Script	339
Глава 7. Создание и проверка достоверности транзакций	342
Глава 8. Оплата по хешу сценария	346
Глава 9. Блоки	349
Глава 10. Организация сети	353
Глава 11. Упрощенная проверка оплаты	356
Глава 12. Фильтры Блума	358
 Предметный указатель	 365

Об авторе

Джимми Сонг является разработчиком с более чем 20-летним стажем, начавший свою карьеру с чтения второго издания книги *Programming Perl*, вышедшей в том же самом издательстве, в котором вышла и данная книга. Он принимал участие во многих новых предприятиях и полностью занялся биткойном в 2014 году. С тех пор он внес свой посильный вклад в самые разные проекты биткойна с открытым кодом, включая Armory, Bitcoin Core, btcd и ruscoin.

Если вы когда-нибудь повстречаете Джимми и захотите вызвать в нем желание порассуждать, спросите его о биткойне, устойчивой валюте, худых последствиях от декретной валюты, посте, плотоядии, силовой атлетике, воспитании детей и ковбойских шляпах.

Об изображении на обложке

На обложке настоящего издания этой книги изображен медоед (*Mellivora capensis*), иначе называемый лысым барсуком. Несмотря на свое название, это животное внешне похоже скорее на ласку или хорька, чем на барсука. Оно обитает повсюду в Африке, Индии и Юго-Западной Азии. Медоед — хищное животное, проявляющее свой невероятно свирепый нрав, когда ему приходится защищаться.

Свое привычное название медоед получил из-за повадки совершать набеги на ульи и поедать любимый им мед и личинки пчел. У него толстая шкура, сводящая на нет последствия пчелиных укусов. Но рацион медоедов весьма разнообразен и включает в себя змей (в том числе ядовитых), грызунов, насекомых, лягушек, птиц, яйца, плоды, корни и луковицы растений. Было замечено, что медоед прогоняет молодых львов от своей добычи и относится к немногим видам животных, наблюдения за которым велись с использованием современных инструментов.

Медоед — крепко сложенное животное с длинным телом, широкой спиной и небольшой плоской головой. У него короткие ноги, оканчивающиеся острыми когтями, благодаря которым он исключительно хорошо роет землю. Медоед способен не только докопаться до своей добычи под землей, но и вырыть нору для себя (длиной в среднем 1–3 м). У основания его хвоста находится железа, наполненная зловонной секретией, которой он пользуется для пометки своей территории и предупреждения других животных. Шкура

на шее медоеда не натянута, что позволяет ему вертеть головой и кусаться, когда его хватают.

В 2011 году бесстрашное поведение медоеда стало сюжетом комического вирусного видео, снятого *National Geographic*.

Многие виды животных, изображенных на обложках книг издательства O'Reilly, находятся под угрозой исчезновения, хотя все они важны для нашего мира. Подробнее о том, как помочь спасению этих животных, можно узнать по адресу animals.oreilly.com. На первой странице обложки этой книги приведена иллюстрация с изображением медоеда, выполненная Карен Монгомери на основании черно-белой гравюры, взятой из энциклопедического издания *Natural History of Animals* (Естественная история животных).

Предисловие

Как замечательно быть писателем научно-фантастических повестей и романов! В них можно построить общество, в котором благосостояние больше не является миражом, возникающим из пустых обещаний правительств и манипуляций центральных банков, обмениваться ценностями можно безо всяких подозрений и оплаты услуг посредника, код может быть законом, а коллективные решения могут приниматься без искажений, вносимых централизацией... Для этого писателю достаточно открыть текстовый редактор и приступить к воплощению замысла своего научно-фантастического детища.

Но для написания захватывающих романов нужно не только воображение, но и знание окружающего мира. “Построение мира” — это не столько литературное правдоподобие или напичканный техническими терминами текст, сколько проникновение глубоко внутрь, чтобы добраться до сути и понять, каким образом действует этот мир, задаваясь вопросами вроде “Что если...” И чем лучше писатель понимает механизмы и коды, образующие описываемый им мир, тем более интересными становятся вопросы, которые в связи с ними возникают.

Изменить реальный мир намного труднее, чем написать научно-фантастический роман, но и для этого требуются знания. Помимо мудрости, идеализма, настойчивости, дисциплины и убежденности перед лицом сомнений, тому, кто претендует на преобразование окружающего мира, необходимо ясно понимать, какие средства и возможности ему доступны и каковы их ограничения.

В настоящее время мир биткойна и блокчейна все еще остается по большей части фантастическим. Доморожденные знатоки, вселяющие надежды, не скупящиеся на похвалы и совсем не понимающие реального положения вещей, слышны намного громче и оказывают большее влияние, чем те, кто выполняет всю тяжелую работу, чтобы изменить окружающий мир. Привлекая на помощь техническую терминологию и актуальные тематические метки в социальных сетях, политически мотивированные

разглагольствования, основанные на страхе и схемах быстрого обогащения, призывают жаждать знаний.

Но, читая техническую документацию и обзорные статьи, понять блокчейн можно не больше, чем научиться открывать свое дело, посещая школу предпринимательства или наблюдая презентации, составленные в PowerPoint.

Для этого придется программировать.

Чтобы разобраться в какой-нибудь технологии, лучше всего создать с ее помощью нечто полезное. До тех пор, пока вы самостоятельно не запрограммируете основополагающие конструктивные блоки приложения на основе блокчейна, вы не уловите разницы между пустой рекламной шумихой и осуществимыми возможностями.

Эта книга служит наиболее эффективным и исчерпывающим источником информации для изучения биткойна и блокчейна через программирование. Опираясь на свой опыт и знания, ее автор, Джимми Сонг, проложил такой путь к изучению, который проведет читателя от математических основ биткойна до новейших расширений и разветвлений. И на этом пути читателю предоставляется возможность поупражняться на примерах, которые были проверены на реальных учащихся и не только научат принципам действия биткойна, но и позволят интуитивно понять изящество и прелесть данной технологии.

Но путь этот нелегкий. Даже имея такого наставника и приверженца биткойна, как Джимми Сонг, читателю не стоит уповать на то, что, пролистав эту книгу скуки ради в промежутках между просмотром развлекательных сериалов от компании Netflix, он сможет легко разобраться в данной технологии. Для этого придется немало потрудиться. Кратчайшего пути к ее освоению не существует, как, впрочем, и возможности кратко изложить ее суть. Но именно это обстоятельство как нельзя лучше отвечает основополагающему принципу биткойна: иметь личную заинтересованность и демонстрировать доказательство выполнения работы. И лишь тогда можно доверять своим знаниям.

Удачного программирования!

— Кен Лю¹

¹ Кен Лю (Ken Liu) — автор эпической фантастической серии книг *The Dandelion Dynasty* (Династия одуванчиков) в стиле “силкпанк” (шелковая чепуха), в которой техника является волшебством, а также сборника повестей *The Paper Menagerie and Other Stories* (Бумажный зверинец и другие истории). Его научно-фантастический рассказ *Byzantine Empathy* (Византийская эмпатия) о блокчейне был первоначально опубликован в издательстве MIT Press.

Введение

Эта книга призвана обучить технологии биткойна на базовом уровне. В ней не рассматривается монетарная, экономическая или социальная динамика развития биткойна, но поясняется внутренний механизм его действия, позволяющий лучше понять возможности данной технологии. В настоящее время наблюдается тенденция широко расхваливать биткойн и блокчейн, не понимая сути дела. И цель этой книги — противостоять подобной тенденции.

Существует немало литературы, в которой излагается история развития, техническое описание биткойна и экономические особенности его применения. А эта книга призвана помочь вам разобраться в биткойне, программируя все компоненты для его библиотеки. Назначение такой библиотеки — помочь вам в изучении биткойна, а не быть исчерпывающей или эффективной.

Для кого написана эта книга

Эта книга адресована тем программистам, которые стремятся изучить принцип действия биткойна, программируя его самостоятельно. Читателям представится возможность изучить биткойн, программируя “голое железо” с помощью библиотеки биткойна, которую нужно будет создать заново. Поэтому данную книгу не стоит рассматривать как справочное пособие, к которому можно обратиться за описанием конкретного функционального средства.

Материал этой книги основывается главным образом на материале двухдневного семинара (<https://programmingbitcoin.com/schedule/>), на котором разработчики могут научиться всему, что следует знать о биткойне. Этот материал существенно усовершенствовался по мере того, как данный курс обучения был проведен 20 раз более чем для 400 обучающихся на момент написания данной книги.

Завершив чтение этой книги, вы будете уметь не только создавать транзакции, но и получать все необходимые данные от партнеров и отправлять транзакции по сети. Здесь есть все, что для этого требуется, включая

математический аппарат, синтаксический анализ, связность узлов сети и проверку достоверности блоков.

Что нужно знать

Предпосылкой для чтения этой книги служит умение программировать, в частности, на языке Python. Сама рассматриваемая здесь библиотека написана на языке Python версии 3, а многие предлагаемые упражнения могут быть выполнены в такой интерактивной среде, как, например, командная оболочка Jupyter Notebook. Знать язык Python желательно на среднем уровне, хотя и начальных знаний может быть достаточно для усвоения многих понятий.

Для усвоения материала книги, в особенности глав 1 и 2, необходимо немного знать математику. В этих главах представлены математические понятия, которые, вероятно, малознакомы тем, кто не является знатоком математики. Впрочем, знаний математики на уровне алгебры должно быть достаточно, чтобы усвоить новые понятия и выполнить упражнения по программированию, предлагаемые в этих главах.

При чтении этой книги пригодится знание общих основ вычислительной техники (например, функций хеширования), хотя для выполнения предлагаемых здесь упражнений наличие таких знаний не строго обязательно.

Как организована эта книга

Эта книга разделена на 14 глав. Материал каждой последующей главы построен на материале предыдущей и позволяет создать библиотеку для биткойна от начала и до конца.

Грубо говоря, в главах 1–4 описывается весь требующийся математический аппарат. В главах 5–8 рассматриваются транзакции, являющиеся основными единицами биткойна, а в главах 9–12 — блоки и организация сети. И наконец, в главах 13 и 14 обсуждается ряд дополнительных вопросов, фактически не требующих написания кода.

А точнее говоря, необходимый математический аппарат излагается в главах 1, “Конечные поля”, и 2, “Эллиптические кривые”. В частности, для понимания основ криптографии на эллиптических кривых требуется знать, что такое конечные поля и эллиптические кривые, поясняемые в главе 3, “Криптография по эллиптическим кривым”. И как только в конце главы 3 будет введено понятие криптографии с открытым ключом, в главе 4, “Сериализация”,

будут представлены механизмы синтаксического анализа и сериализации, с помощью которых сохраняются и передаются криптографические примитивы.

В главе 5, “Транзакции”, описывается структура транзакции, а в главе 6, “Язык Script” — язык используемый для написания умных контрактов, положенный в основу биткойна. В главе 7, “Создание и проверка достоверности транзакций”, основывающейся на материале предыдущих глав, поясняется, как проверять на достоверность и создавать транзакции, применяя криптографию на эллиптических кривых, описанную в главах 1–4. А в главе 8, “Оплата по хешу сценария”, разъясняется принцип действия оплаты по хешу сценария (p2sh) как способа повышения эффективности умных контрактов.

В главе 9, “Блоки”, описываются блоки, представляющие собой группы упорядоченных транзакций, а в главе 10, “Организация сети” — передача данных по сети в биткойне. В главах 11, “Упрощенная проверка оплаты”, и 12, “Фильтры Блума”, поясняется, как “тонкий” клиент (т.е. прикладная программа без доступа ко всему блокчейну) может запросить и организовать взаимобмен данными с теми узлами, в которых хранится весь блокчейн.

В главе 13, “Протокол Segwit”, описывается обратно совместимое обновление протокола, называемое Segwit и внедренное в 2017 году, а в главе 14, “Дополнительные вопросы и следующие шаги”, даются рекомендации по дальнейшему изучению биткойна. И хотя читать эти главы совсем необязательно, они включены в книгу для того, чтобы дать некоторое представление о дальнейших путях изучения данной технологии.

Подготовка

Чтобы извлечь наибольшую пользу из этой книги, вам придется создать среду, в которой вы сможете выполнять примеры кода и упражнения. Ниже перечислены действия, которые следует предпринять, чтобы хорошо подготовиться к проработке материала этой книги.

1. Установите на своем компьютере интерпретатор языка Python не ниже версии 3.5, загрузив его дистрибутив по указанным ниже адресам.
Для системы Windows — по адресу <https://www.python.org/ftp/python/3.6.2/python-3.6.2-amd64.exe>.
Для macOS — по адресу <https://www.python.org/ftp/python/3.6.2/python-3.6.2-macosx10.6.pkg>.

Для систем Linux — обратитесь к документации на отдельные дистрибутивы (многие дистрибутивы Linux вроде Ubuntu поставляются вместе с предварительно установленной версией Python 3.5+).

2. Установите сценарий `pip`, загрузив его по адресу <https://bootstrap.pypa.io/get-pip.py>.

3. Запустите этот сценарий в версии Python 3 следующей командой:

```
$ python3 get-pip.py
```

4. Установите распределенную систему контроля версий Git.

Инструкции по ее загрузке и установке находятся по адресу <https://git-scm.com/downloads>.

5. Загрузите исходный код, прилагаемый к этой книге, выполнив следующие команды:

```
$ git clone https://github.com/jimmysong/programmingbitcoin
$ cd programmingbitcoin
```

6. Установите виртуальную среду `virtualenv` следующей командой:

```
$ pip install virtualenv --user
```

7. Установите требования.

В Linux/macOS — следующими командами:

```
$ virtualenv -p python3 .venv
$ . .venv/bin/activate
(.venv) $ pip install -r requirements.txt
```

В Windows — следующими командами:

```
C:\programmingbitcoin> virtualenv -p
C:\PathToYourPythonInstallation\Python.exe .venv
C:\programmingbitcoin> .venv\Scripts\activate.bat
C:\programmingbitcoin> pip install -r requirements.txt
```

8. Запустите командную оболочку Jupyter Notebook:

```
(.venv) $ jupyter notebook
[I 11:13:23.061 NotebookApp] Serving notebooks
from local directory:2
/home/jimmy/programmingbitcoin
[I 11:13:23.061 NotebookApp]
The Jupyter Notebook is running at:3
```

² Обслуживание блокнотов из следующего локального каталога:

³ Jupyter Notebook выполняется по адресу:

```
[I 11:13:23.061 NotebookApp] http://localhost:8888/?token=
f849627e4d9d07d2158e3fcde93590eff4a9a7a01f65a8e7
[I 11:13:23.061 NotebookApp]
Use Control-C to stop this server and
shut down all kernels (twice to skip confirmation).4
[C 11:13:23.065 NotebookApp]
```

Copy/paste this URL into your browser when you connect for the first time, to login with a token:⁵

```
http://localhost:8888/?token=
f849627e4d9d07d2158e3fcde93590eff4a9a7a01f65a8e7
```

В итоге браузер должен открыться автоматически, как показано на рис. 1.

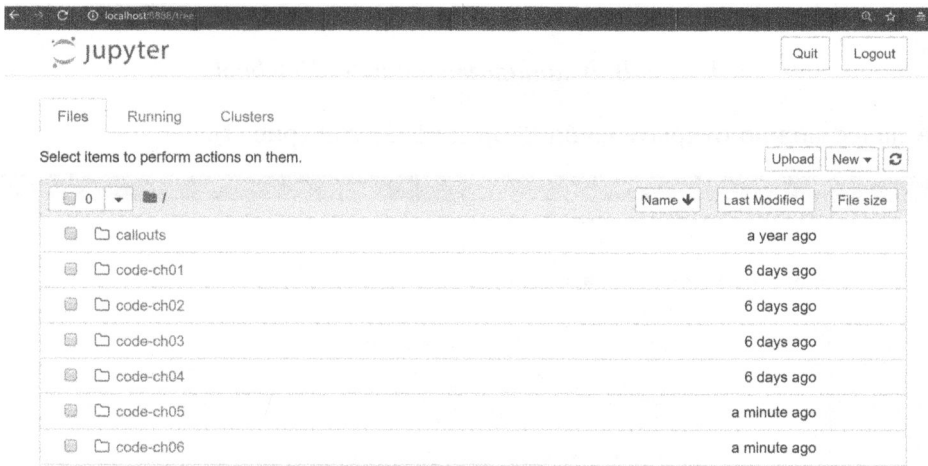


Рис. 1. Командная оболочка Jupyter Notebook

Отсюда можно перейти к каталогам, распределенным по отдельным главам книги. Так, для выполнения упражнений из главы 1 перейдите к каталогу code-ch01 (рис. 2).

⁴ Чтобы остановить сервер и все ядра, нажмите комбинацию клавиш <Ctrl+C> (дважды, чтобы пропустить подтверждение).
⁵ Скопируйте и вставьте этот URL в окне своего браузера при первом подключении, чтобы зарегистрироваться с указанным в нем токеном:

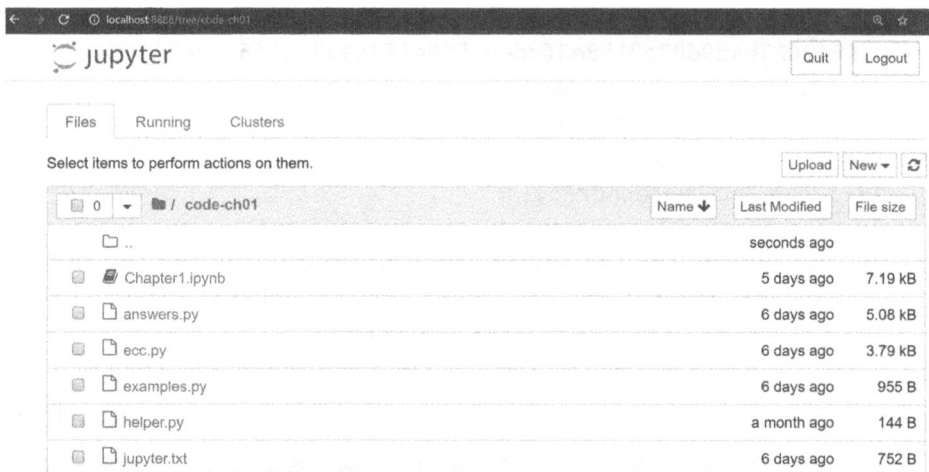


Рис. 2. Вид каталогов в Jupyter Notebook

А далее можно открыть файл Chapter1.ipynb (рис. 3).

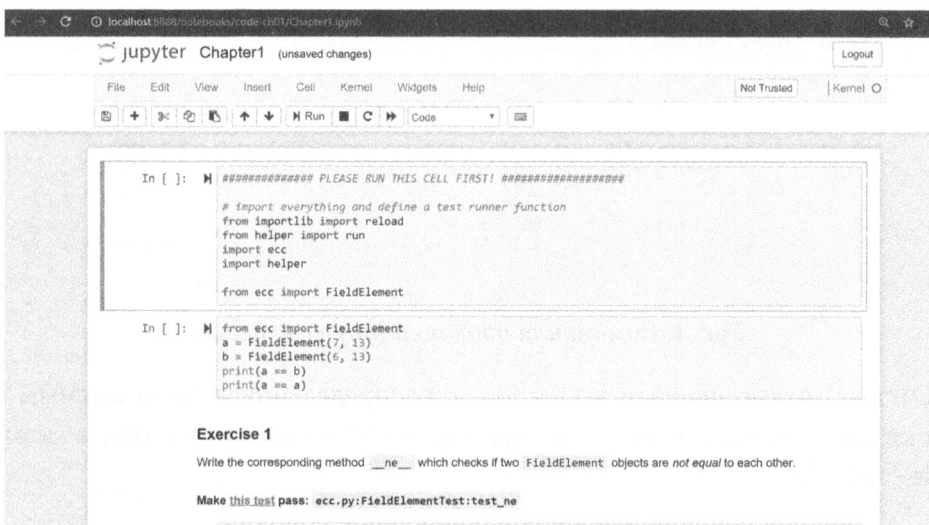


Рис. 3. Вид блокнота в Jupyter Notebook

Вам, возможно, придется освоиться с таким интерфейсом, если вы пользуетесь им впервые. Но истинный смысл командной оболочки Jupyter Notebook состоит в том, что в ней можно выполнять код Python из браузера, что упрощает экспериментирование с ним. Каждую “ячейку” кода можно выполнять по отдельности, чтобы наблюдать результаты, как будто это интерактивная оболочка Python.

Большая часть упражнений посвящена понятиям из области программирования, представленным в этой книге. И хотя модульные тесты уже написаны для вас, вам все же придется написать код Python для прохождения тестов. Проверить правильность написанного кода можно и непосредственно в Jupyter Notebook. Для этого необходимо отредактировать соответствующий файл, щелкнув на ссылке `this test` (данный тест; см. рис. 3). По этой ссылке произойдет переход на вкладку браузера, аналогичную приведенной на рис. 4. Отредактируйте здесь файл и сохраните его, чтобы тест был выполнен.

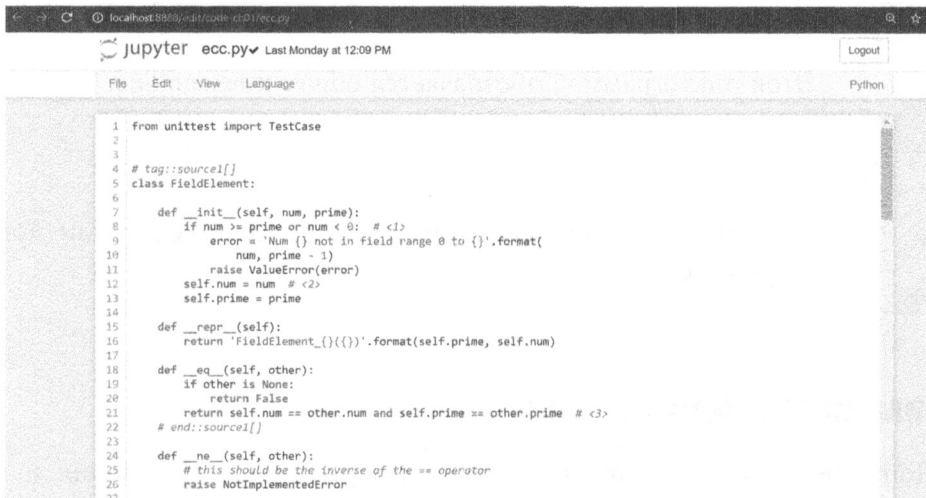


Рис. 4. Редактирование исходного файла `ecc.py`

Ответы к упражнениям

Все ответы к упражнениям, предлагаемым в этой книге, приведены в приложении. Они доступны также в файлах `code-ch<xx>/answers.py`, где `<xx>` — номер соответствующей главы.

Соглашения, принятые в этой книге

В этой книге приняты следующие условные обозначения.

Курсивом выделяются новые термины.

Моноширинным шрифтом выделяются адреса электронной почты, имена и расширения файлов, элементы кода (переменные, имена функций, операторы, переменные окружения, ключевые слова и типы данных), а также исходный код в листингах.

Полужирным моноширинным шрифтом выделяются команды или другой текст, который должен быть введен пользователем.

Наклонным моноширинным шрифтом выделяется текст, который должен быть заменен предоставляемыми пользователем значениями или же теми значениями, которые определяются по контексту.



Этой пиктограммой обозначается совет или рекомендация.



Этой пиктограммой обозначается общее примечание.



Этой пиктограммой обозначается предупреждение или предостережение.

Примеры исходного кода

Дополнительный материал книги (примеры кода, упражнения и пр.) доступен для загрузки по адресу <https://github.com/jimmysong/programmingbitcoin>. Эта книга служит учебным пособием, помогающим читателю решать стоящие перед ним задачи разработки прикладных программ. В общем, приведенные здесь примеры кода можно использовать в своих программах и документации. Для этого не нужно спрашивать разрешение у автора или издателя. Так, для применения в прикладной программе нескольких фрагментов кода из примеров в этой книге специальное разрешение не требуется. Но для продажи или распространения в иных целях на внешних носителях фрагментов кода из примеров обязательно нужно получить разрешение от издательства O'Reilly. Для цитирования текста и примеров кода из этой книги в ответах на вопросы специальное разрешение не требуется. Но для внедрения значительной части примеров кода в документацию на собственную продукцию обязательно необходимо разрешение от издательства O'Reilly.

Ссылки на эту книгу как на первоисточник желательны, но не обязательны. В ссылке обычно указываются название книги, автор, издатель и ISBN:

Если читатель считает, что применение им примеров кода из этой книги выходит за рамки правомерного использования или упомянутых выше разрешений, он может связаться с издательством O'Reilly по адресу permissions@oreilly.com.

Благодарности

Тех, кого мне хотелось бы поблагодарить, — целый легион. Мне как автору этой книги пришлось черпать знания и опыт из многих источников, и поэтому трудно воздать должное всем, кто послужил такими источниками. Этим я хочу сказать, что если я забуду кого-нибудь упомянуть, то приношу им искреннее извинение.

Прежде всего, мне хотелось бы поблагодарить Господа Иисуса Христа, направившего меня на этот путь. Если бы не моя вера, я бы не взялся из этических убеждений вести блог о важности устойчивой валюты вообще и биткой-на в частности, что в конечном итоге привело к написанию этой книги.

Мои родители Кэти и Кьунг-Суп набрались мужества эмигрировать в Америку, когда мне было 8 лет от роду, что в конечном счете привело к тем возможностям, которые у меня появились. Мой отец подарил мне первые мои компьютеры (Commodore 16, ПК Hyundai на обыкновенном процессоре 8086 и еще один ПК на процессоре 486 с тактовой частотой 33 МГц от производителя, название которого я позабыл). А когда я учился в шестом и седьмом классах, моя мать наняла для меня частную преподавательницу по программированию, имя которой я уже не помню. Даже не знаю, как и где она ее нашла, когда я стал проявлять склонность к программированию. Эта позабытая мной преподавательница поддержала мое стремление к программированию, и, я надеюсь, моя признательность найдет свой путь к ней.

На своем жизненном пути я встретил немало учителей, и некоторых из них я в свое время просто ненавидел. В частности, мистер Марэн, миссис Эдельман и миссис Нельсон учили меня математике и вычислительной технике еще в школе. И хотя я совсем не являюсь сторонником современной образовательной системы, то, чему меня научили в школе, поспособствовало зарождению во мне любви к математике и программированию.

Мой одноклассник Эрик Сильберштейн дал мне после колледжа первую работу программистом в компании Idiom Technologies. Я собирался было

пойти по пути консультанта, когда судьбоносный телефонный звонок в 1998 году повел меня по пути программиста, которого я, по существу, с тех пор и не покидал.

На своей первой работе я познакомился с Кеном Лю, и просто удивительно, как мы занялись написанием книг, каждый на своем уровне. Он не только дал мне дельный совет по поводу издания книги, но и написал чудесное предисловие к ней. Кроме того, он отзывчивый и изумительный товарищ, и знакомство с ним для меня — большая честь.

Что же касается биткойна, то анонимный разработчик Сатоши Накамото изобрел то, что прежде казалось невозможным: децентрализованный цифровой дефицит. Изобретение биткойна имеет принципиальное значение, которое мир еще не до конца осознал. Я познакомился с биткойном на веб-сайте Slashdot (<https://slashdot.org/>) в 2011 году, а первую свою работу в качестве разработчика биткойна я получил от Алекса Мизрахи в 2013 году. И тогда я даже не представлял, что я должен был делать, хотя и немало узнал о биткойне под его руководством.

В 2013 году я встретил немало биткойнеров в группе Austin Bitcoin Meetup, а на следующей конференции Texas Bitcoin Conference в 2014 году — огромное количество людей, которых я знаю и поныне. И хотя я больше не принимаю участия не только в этой группе и конференции, я весьма признателен тем, кого я там встретил, в частности — Майклу Гольдштейну, Дениэлу Кравишу и Наполеону Коулу.

Алан Рейнер нанял меня на работу над проектом Armory еще в 2014 году, и я благодарен ему за возможность участвовать в таком значительном проекте.

Затем я работал в компании Paxos/itBit, где исполнительным директором был Чэд Каскарилла, а вице-президентом по проектированию — Радж Наир, которые расширили мой кругозор, пока я там работал. В частности, Радж обязал меня делать записи в блоге от компании Paxos/itBit, что, как ни удивительно, мне понравилось. И впоследствии я стал писать блог-посты на своем веб-сайте, что в конечном счете привело к организации семинаров и написанию этой книги.

Особого упоминания заслуживают трое моих коллег, с которыми я познакомился в компании Paxos/itBit. В частности, Ричард Кисс, создатель приложения rusoin, побудил меня к написанию этой книги, предложив писать ее вместе. И хотя на каком-то этапе мне пришлось писать ее самому, я все же благодарен ему за эту идею. Аарон Кезуэлл, будучи замечательным разработчиком, помог мне в организации семинаров и рецензировании этой книги.

Он не только прекрасный программист и математик, но, как я слышал, прилично владеет приемами каратэ. Майкл Флэксмен рецензировал все, что я написал о биткойне, включая мой блог, многие из моих библиотек в информационном хранилище GitHub и эту книгу. Он также помог мне в организации семинаров; и он — просто хороший собеседник. Майкл из тех, кто делает других лучше, и я благодарен ему за дружбу, которая для меня много значит.

В 2017 году приложение Vortex, Томас Хант и Тоун Вэйз привели меня на канал World Crypto Network, где началась моя деятельность на канале YouTube. В частности, Тоун вдохновлял меня на тяжкие труды и преданность делу.

Джон Ньюбери оказал мне помощь, когда я впервые принял участие в проекте Bitcoin Core; и он — просто хороший человек. Джон стал заметным участником данного проекта за относительно короткое время, что свидетельствует о его даровании и преданности делу. Благодарю также других участников проекта Bitcoin Core, в том числе Марко Фальке, Владимира ван дер Лаана, Алекса Моркоса, Пьетера Вуилле, Мэтта Коралло, Сухаза Дафтуара и Грега Максуэлла, которые просматривали написанный мною код и записи в блоге.

Особой благодарности заслуживает Дэвид Хардинг за научное рецензирование этой книги. Он трижды просмотрел всю ее рукопись, сделав немало полезных комментариев. Ему бы лучше написать когда-нибудь книгу о биткойне, поскольку он обладает энциклопедическими знаниями практически обо всем, что произошло в истории биткойна.

Джим Кельвин помог мне связаться с издательством O'Reilly, а Майк Лукидес как приемный редактор дал зеленый свет данному проекту. Андреас Антонополус дал мне ряд ценных указаний и рекомендовал меня издательству O'Reilly. Мишель Кронин помогала мне придерживаться намеченного графика, когда я целый год был полностью поглощен написанием этой книги. Кристен Браун как производственный редактор сделала немало для того, чтобы эта книга вовремя увидела свет, а Джеймс Фрелай выполнил ее литературное редактирование. Я большой поклонник издателя Тима О'Райли, который сослужил немалую службу инженерному сообществу, выпуская такие замечательные книги.

В процессе написания этой книги мне помогала целая компания биткойнеров из группы Austin Bitcoin Meetup, в том числе Брайан Бишоп, Уилл Коул, Наполеон Коул, Типтон Коул, Тур Демеестер, Джонни Дилли, Майкл Флэксмен, Паркер Льюис, Джастин Мун, Алан Пискителло и Эндрю Поельстра. Следует также отметить канал Slack (от компании TAAS), на котором Сайфедин Аммуз заявил, что он пишет книгу, успех которой вдохновил и меня.

Помимо биткойнеров из GitHub, бывшие слушатели моего курса по программированию биткойна также приняли участие в рецензировании этой книги. В частности, Джефф Флауэрс, Брайан Лиотти, Кэйзи Боумен, Джонсон Лау, Альберт Чен, Джейсон Лез, Томас Браунбергер, Эдуардо Кобейн и Спенсер Хенсон выявили ряд недостатков в рукописи. А мой преданный помощник Катрина Хавьер помогала мне в составлении многих диаграмм к этой книге.

Мои подписчики на канале YouTube, последователи в Twitter и читатели в Medium помогли мне найти себя и стать на путь предпринимательства.

И наконец, моя жена Джули и мои дети оказывали мне всяческую поддержку в течение двух последних лет. Если бы не они, то сомневаюсь, что у меня были бы стимулы так много работать.

Ждем ваших отзывов!

Вы, читатель этой книги, и есть главный ее критик. Мы ценим ваше мнение и хотим знать, что было сделано нами правильно, что можно было сделать лучше и что еще вы хотели бы увидеть изданным нами. Нам интересны любые ваши замечания в наш адрес.

Мы ждем ваших комментариев и надеемся на них. Вы можете прислать нам электронное письмо либо просто посетить наш веб-сайт и оставить свои замечания там. Одним словом, любым удобным для вас способом дайте нам знать, нравится ли вам эта книга, а также выскажите свое мнение о том, как сделать наши книги более интересными для вас.

Отправляя письмо или сообщение, не забудьте указать название книги и ее авторов, а также свой обратный адрес. Мы внимательно ознакомимся с вашим мнением и обязательно учтем его при отборе и подготовке к изданию новых книг.

Наши электронные адреса:

E-mail: info.dialektika@gmail.com

WWW: <http://www.dialektika.com>

Конечные поля

Самое сложное в программировании биткойна — выяснить, с чего следует начать. Ведь существует столько взаимозависимых компонентов, что изучение одного из них может привести к необходимости изучить другой, а тот, в свою очередь, — изучить еще что-нибудь, прежде чем разобраться в первоначальном компоненте.

Назначение этой главы — облегчить вам начало. Как ни странно, мы начнем с основ математики, которые потребуются, чтобы уяснить особенности криптографии на эллиптических кривых. А криптография на эллиптических кривых, в свою очередь, даст представление об алгоритмах подписания и верификации. Ведь именно на этих алгоритмах основывается принцип действия транзакций, которые являются атомарными единицами перевода стоимости в биткойне. Изучив сначала конечные поля и эллиптические кривые, вы сможете твердо усвоить понятия, необходимые для логичного продвижения вперед.

Но имейте в виду, что материал этой и двух последующих глав сродни вегетарианской диете, особенно если вы давно не занимались формальной математикой. Тем не менее прочтите их, поскольку представленные в них понятия и примеры кода будут использоваться в остальной части книги.

Высшая математика

Изучение математических структур может оказаться несколько путающим занятием, но можно надеяться, что в этой главе миф о трудностях освоения высшей математики будет развеян. В частности, для изучения конечных полей достаточно математических знаний на уровне алгебры.

Конечные поля можно рассматривать как математическую дисциплину, вполне приемлемую для изучения вместо тригонометрии, если бы в образовательной системе не было принято решение, что изучать тригонометрию

важнее. Этим я хочу сказать, что изучить конечные поля нетрудно и для этого потребуется не больше предварительной подготовки, чем знание алгебры.

В этой главе вам нужно будет разобраться в криптографии на эллиптических кривых. А криптография на эллиптических кривых требуется для того, чтобы уяснить алгоритмы подписания и верификации, составляющие саму суть биткойна. Как упоминалось выше, материал этой и двух последующих глав может показаться, на первый взгляд, никак не связанным с рассматриваемым здесь предметом; тем не менее настоятельно рекомендуется их прочитать. Основы, изложенные в этих главах, упростят понимание не только биткойна, но и подписей Шнорра, конфиденциальных транзакций и прочих передовых технологий биткойна.

Определение конечного поля

В математике *конечное поле* (finite field) определяется как конечное множество чисел, над которыми можно выполнить две математические операции, + (сложения) и $\times$ (умножения), удовлетворяющие перечисленным ниже условиям.

1. Если a и b принадлежат множеству, то результаты операций $a + b$ и $a \times b$ также принадлежит этому множеству. Такое свойство можно назвать *замкнутым* (closed).
2. Нуль существует и обладает свойством $a + 0 = a$. Такое свойство можно назвать *аддитивным тождеством* (additive identity).
3. Единица существует и обладает свойством $a \times 1 = a$. Такое свойство можно назвать *мультипликативным тождеством* (multiplicative identity).
4. Если a принадлежит множеству, то ему же принадлежит и $-a$, определяемое как значение, удовлетворяющее выражению $a + (-a) = 0$. Такое свойство можно назвать *аддитивной инверсией* (additive inverse).
5. Если a не равно 0 и принадлежит множеству, то ему же принадлежит и a^{-1} , определяемое как значение, удовлетворяющее выражению $a \times a^{-1} = 1$. Такое свойство можно назвать *мультипликативной инверсией* (multiplicative inverse).

Рассмотрим каждое из перечисленных выше условий более подробно.

Мы работаем с конечным множеством чисел. Поскольку оно конечное, то можно задать некоторое число p , определяющее размер множества, называемое также *порядком* (order) множества.

Так, условие №1 говорит о том, что множество замкнуто относительно сложения и умножения. Это означает, что операции сложения и умножения определяются таким образом, чтобы их результаты принадлежали данному множеству. Например, множество $\{0,1,2\}$ не является замкнутым относительно сложения, поскольку $1 + 2 = 3$ и $2 + 2 = 4$, но ни число 3, ни число 4 не принадлежит данному множеству. Безусловно, операцию сложения можно определить несколько иначе, чтобы сделать данное множество замкнутым, но с помощью “обычной” операции сложения такое множество нельзя сделать замкнутым. А с другой стороны, множество $\{-1,0,1\}$ является замкнутым относительно умножения. Если перемножить два любых числа из этого множества, а всего таких комбинаций девять, то их произведение всегда будет принадлежать множеству.

В математике имеется и другая возможность определить операцию умножения особым образом, чтобы сделать подобные множества замкнутыми. О том, как именно определить операции сложения и умножения, речь пойдет далее в этой главе, а до тех пор следует сказать, что операции сложения и вычитания можно определить *иначе*, чем вам, вероятно, известно.

В частности, условия №2 и 3 определяют аддитивное и мультипликативное тождества. Это означает, что числа 0 и 1 принадлежат множеству.

Условие №4 определяет аддитивную инверсию. Это означает, что если a принадлежит множеству, то ему же принадлежит и $-a$. Используя аддитивную инверсию, можно определить операцию вычитания.

Далее, условие №5 определяет, что аналогичным свойством обладает и операция умножения. Если a принадлежит множеству, то ему же принадлежит и a^{-1} . Это означает, что $a \times a^{-1} = 1$. Используя мультипликативную инверсию, можно определить операцию деления. Следует заметить, что для конечного поля это сделать будет непросто.

Определение конечных множеств

Если порядок (или размер) множества равен p , то элементы множества можно обозначить как 0, 1, 2, ... $p-1$. Именно эти числа, а совсем не обязательно традиционные числа 0, 1, 2, 3 и так далее, называются *элементами* множества. И хотя они во многом ведут себя, как традиционные числа, они

различаются порядком выполнения операций сложения, вычитания, умножения и пр.

В математическом обозначении множество конечных полей выглядит следующим образом:

$$F_p = \{0, 1, 2, \dots, p-1\}.$$

Каковы же элементы множества конечных полей? F_p — это конкретное конечное поле, называемое “полем p ”, “полем 29” или полем любого размера, которое в математике называется *порядком*. А числа в фигурных скобках $\{\}$ обозначают элементы, находящиеся в полях. Эти элементы именуются как 0, 1, 2 и так далее поскольку так удобнее для наших целей.

Так, конечное поле порядка 11 выглядит следующим образом:

$$F_{11} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}.$$

А конечное поле порядка 17 — таким образом:

$$F_{17} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16\}.$$

И наконец, конечное поле порядка 983 выглядит так, как показано ниже.

$$F_{983} = \{0, 1, 2, \dots, 982\}$$

Следует, однако, иметь в виду, что порядок конечного поля всегда на 1 больше значения его последнего элемента. Вы, вероятно, обратили внимание на то, что порядок конечного поля всякий раз оказывается равным простому числу. По разным причинам, которые станут понятны в дальнейшем, оказывается, что порядок конечных полей *должен* быть равен степени простого числа и что наибольший для нас интерес представляют именно те конечные поля, порядок которых равен простому числу.

Построение конечного поля на языке Python

Нам требуется каким-то образом представить каждый элемент конечного поля. Поэтому создадим в Python класс, представляющий единственный элемент конечного поля. Этому классу естественно присвоить имя `FieldElement`. Этот класс представляет элемент из конечного поля F_{prime} . Скелетное представление этого класса выглядит следующим образом:

```
class FieldElement:
```

```
    def __init__(self, num, prime):
        if num >= prime or num < 0: ❶
            error = 'Num {} not in field range 0 to {}'.format(
```

```

        num, prime - 1)
    raise ValueError(error)
self.num = num ❷
self.prime = prime

def __repr__(self):
    return 'FieldElement_{}({})'.format(
        self.prime, self.num)

def __eq__(self, other):
    if other is None:
        return False
    return self.num == other.num
        and self.prime == other.prime ❸

```

- ❶ Сначала в классе проверяется, находится ли значение свойства `num` в пределах от 0 до `prime-1` включительно. Если это не так, то элемент конечного поля типа `FieldElement` некорректен и передается исключение типа `ValueError`, которое должно быть непременно передано, если получено неподходящее значение.
- ❷ В остальной части метода `__init__()` создаваемому объекту данного класса присваиваются инициализирующие значения.
- ❸ В методе проверяется равенство двух объектов класса `FieldElement`. И эта проверка дает истинный результат лишь в том случае, если свойства `num` и `prime` равны.

То, что мы уже определили, позволяет сделать следующее:

```

>>> from ecc import FieldElement
>>> a = FieldElement(7, 13)
>>> b = FieldElement(6, 13)
>>> print(a == b)
False
>>> print(a == a)
True

```

В языке Python можно заменить операцию `==` методом `__eq__()`, что и было сделано в классе `FieldElement` и чем можно выгодно воспользоваться в дальнейшем, как демонстрируется в примерах кода, прилагаемых к этой книге. Установив командную оболочку Jupyter Notebook (см. раздел “Подготовка” во введении этой книги), можно перейти к файлу `code-ch01/Chapter1.ipynb` и выполнить находящийся в нем код, чтобы увидеть полученные результаты. А для выполнения следующего упражнения откройте

файл `ess.ru`, щелкнув на ссылке в блоке Exercise 1 (Упражнение 1). Если у вас возникнут затруднения при выполнении данного упражнения, помните, что ответ к каждому упражнению этой книги, можно найти в Приложении.

Упражнение 1

Напишите соответствующий метод `__ne__()`, в котором оба объекта типа `FieldElement` проверяются на *неравенство*.

Арифметика по модулю

Одним из средств, позволяющих сделать конечное поле замкнутым относительно сложения, вычитания, умножения и деления, является *арифметика по модулю* (modulo arithmetic).

Используя арифметику по модулю, можно определить операцию сложения в конечном множестве, что вам должно быть известно с тех пор, как вы научились делить в столбик. Вспомните, как это делается, глядя на пример задачи, приведенной на рис. 1.1.

$$\begin{array}{r} 2R1 \\ 3 \overline{)7} \end{array}$$

Рис. 1.1. Первый пример деления в столбик

Всякий раз, когда не происходит деления нацело, возникает “остаток”, т.е. величина, остающаяся от фактического деления одного числа на другое¹. Аналогичным образом определяется и модуль, а операция по модулю обозначается символом `%`:

$$7 \% 3 = 1$$

Еще один пример деления в столбик приведен на рис. 1.2.

$$\begin{array}{r} 3R6 \\ 7 \overline{)27} \end{array}$$

Рис. 1.2. Второй пример деления в столбик

¹ На рис. 1.1 `R1` обозначает остаток от деления 7 и 3, равный 1. — *Примеч. ред.*

Рассуждая формально, операция по модулю означает получение остатка от деления одного числа на другое. Ниже приведен еще один пример такой операции над большими числами.

$$1747 \% 241 = 60$$

Если угодно, арифметику по модулю можно рассматривать как арифметику “оборота стрелок часов”. Рассмотрим в качестве примера следующую задачу.

Сейчас 3 часа. Что будет показывать часовая стрелка через 47 часов?

Решение: 2 часа, поскольку $(3 + 47) \% 12 = 2$ (рис. 1.3).

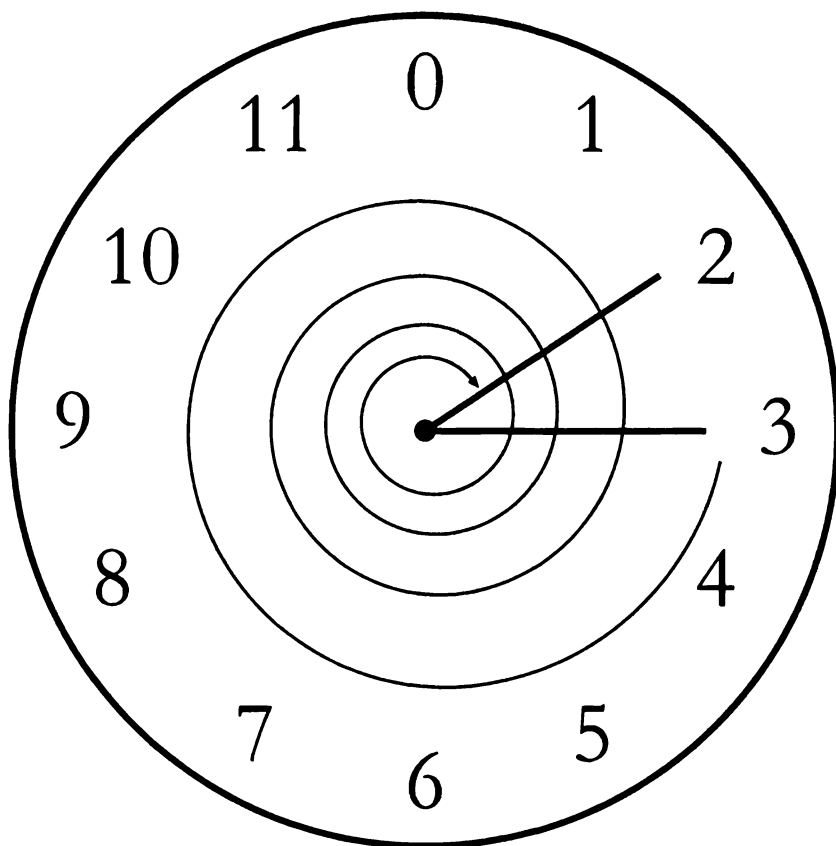


Рис. 1.3. Перемещение часовой стрелки на 47 часов вперед

Такой “оборот стрелок” можно рассматривать и в смысле перехода через 0 всякий раз, когда часовая стрелка перемещается на 12 часов вперед.

Операцию по модулю можно выполнять и над отрицательными числами. В качестве еще одного примера рассмотрим такую задачу.

Сейчас 3 часа. Что показывала часовая стрелка 16 часов назад?

Решение: 11 часов.

$$(3 - 16) \% 12 = 11$$

Перемещение минутной стрелки часов также является операцией по модулю. В качестве примера можно привести следующую задачу.

Сейчас 12 минут второго. Что будет показывать минутная стрелка через 843 минуты?

Решение: она будет показывать 15 минут второго.

$$(12 + 843) \% 60 = 15$$

Аналогично можно сформулировать такую задачу.

Сейчас 23 минуты второго. Что будет показывать минутная стрелка через 97 минут?

В данном случае минутная стрелка будет показывать 0 минут.

$$(23 + 97) \% 60 = 0$$

0 — это еще одно средство для обозначения отсутствия остатка.

Результат операции по модулю (%) для минут всегда находится в пределах от 0 до 59 включительно. И это свойство оказывается очень удобным, поскольку даже очень большие числа можно свести к относительно малым пределам с помощью операции по модулю, как показано ниже.

$$14738495684013 \% 60 = 33$$

Мы будем пользоваться далее операцией по модулю, определяя арифметику конечных полей. В большинстве операций над конечными полями операция по модулю применяется в некоторой форме.

Арифметика по модулю на языке Python

В языке Python для арифметики по модулю служит операция %. Ниже показано, каким образом она применяется непосредственно в коде.

```
>>> print(7 % 3)
1
```

Операцию по модулю можно выполнять и над отрицательными числами, как показано ниже.

```
>>> print(-27 % 13)
12
```

Сложение и вычитание конечных полей

Напомним, что операцию сложения в конечном поле необходимо определить таким образом, чтобы ее результат гарантированно остался в множестве. Таким образом, требуется обеспечить *замкнутость* операции сложения в конечном поле.

Используя рассмотренную выше арифметику по модулю, можно сделать операцию сложения замкнутой. Допустим, что имеется конечное поле 19:

$$F_{19} = \{0, 1, 2, \dots, 18\}$$

Здесь $a, b \in F_{19}$. Обратите внимание на то, что $\in$ означает “является элементом”. В данном случае a и b являются элементами конечного поля F_{19} .

Замкнутость операции сложения обозначается следующим образом:

$$a +_f b \in F_{19}$$

Операция сложения в конечном поле обозначается как $+_f$ во избежание путаницы с обычной целочисленной операцией сложения, обозначаемой как $+$.

Если воспользоваться арифметикой по модулю, то можно гарантировать замкнутость операции сложения в конечном поле. В частности, операцию $a +_f b$ можно определить следующим образом:

$$a +_f b = (a + b) \% 19$$

Например:

$$7 +_f 8 = (7 + 8) \% 19 = 15$$

$$11 +_f 17 = (11 + 17) \% 19 = 9$$

И т.д.

В замкнутой операции сложения из множества берутся два числа, которые складываются, а конечный результат “оборачивается”, чтобы получить сумму. В данном случае мы создаем свою операцию сложения, хотя ее результат не вполне очевиден. Выражение $11 +_f 17 = 9$ выглядит неверно, поскольку мы просто не привыкли к операции сложения в конечном поле.

В общем случае операция сложения в конечном поле определяется следующим образом:

$$a +_f b = (a + b) \% p$$

где $a, b \in F_p$.

Аддитивная инверсия определяется так, как показано ниже, где из $a \in F_p$ следует, что $-a \in F_p$.

$$-_f a = (-a) \% p$$

И снова для ясности обозначение $-_f$ помогает отличать операций вычитания и инверсии в конечном поле от обычных целочисленных операций вычитания и инверсии.

Так, операция инверсии в конечном поле F_{19}

$$-_f 9 = (-9) \% 19 = 10$$

означает, что

$$9 +_f 10 = 0$$

и оказывается истинной.

Аналогично можно выполнить операцию вычитания в конечном поле:

$$a -_f b = (a - b) \% p$$

где $a, b \in F_p$.

Так, в конечном поле F_{19} можно выполнить следующие операции вычитания:

$$11 -_f 9 = (11 - 9) \% 19 = 2$$

$$6 -_f 13 = (6 - 13) \% 19 = 12$$

И т.д.

Упражнение 2

Решите приведенные ниже задачи в конечном поле F_{57} при условии, что все знаки $+$ здесь обозначают операцию сложения $+_f$, а все знаки $-$ — операцию вычитания $-_f$.

- $44 + 33$
- $9 - 29$
- $17 + 42 + 49$
- $52 - 30 - 38$

Программная реализация операций сложения и вычитания на языке Python

Теперь в классе `FieldElement` можно определить методы `__add__()` и `__sub__()` для реализации операций сложения и вычитания в конечном поле. Суть этих методов в том, что в них можно выполнять операции, аналогичные приведенным ниже.

```
>>> from ecc import FieldElement
>>> a = FieldElement(7, 13)
>>> b = FieldElement(12, 13)
>>> c = FieldElement(6, 13)
>>> print(a+b==c)
True
```

В методе `__add__()` данного класса можно определить, что именно обозначает операция сложения (+). Как это сделать на языке Python? Для этого воспользуемся приобретенными знаниями арифметики по модулю и создадим новый метод в классе `FieldElement`, как показано ниже.

```
def __add__(self, other):
    if self.prime != other.prime: ❶
        raise TypeError('Cannot add two numbers in different Fields')
    num = (self.num + other.num) % self.prime ❷
    return self.__class__(num, self.prime) ❸
```

- ❶ Мы должны убедиться, что элементы принадлежат к одному и тому же конечному полю, а иначе данное вычисление не имеет никакого смысла.
- ❷ Операция сложения в конечном поле определяется с помощью операции по модулю, как пояснялось ранее.
- ❸ Мы должны вернуть экземпляр класса, к которому удобно получать доступ с помощью метода `self.__class__()`. Для этого данному методу передаются два инициализирующих аргумента, `num` и `self.prime`, как и для упоминавшегося ранее метода `__init__()`.

Следует заметить, что вместо метода `self.__class__()` можно было бы просто воспользоваться классом `FieldElement`, но тогда сделать этот метод наследуемым было бы непросто. Подклассификацию класса `FieldElement` мы еще осуществим в дальнейшем, а здесь нам важно сделать метод наследуемым.

Упражнение 3

Напишите соответствующий метод `__sub__()`, в котором определяется операция вычитания двух объектов типа `FieldElement`.

Операции умножения и возведения в степень в конечном поле

Подобно тому, как была ранее определена новая операция сложения (+), в конечном поле, можно определить новую замкнутую операцию умножения

для конечных полей. А многократно умножая одно и то же число, можно также определить новую операцию возведения в степень. И в этом разделе поясняется, как определить подобные операции с помощью арифметики по модулю.

Операция умножения, по существу, означает многократное сложение чисел, как демонстрируется в следующих примерах:

$$5 \times 3 = 5 + 5 + 5 = 15$$

$$8 \times 17 = 8 + 8 + 8 + \dots \text{(всего 17 чисел 8)} \dots + 8 = 136$$

Аналогичным образом можно определить операцию умножения в конечном поле. Так, оперируя снова конечным полем F_{19} , операции умножения в нем можно обозначить следующим образом:

$$5 \times_f 3 = 5 +_f 5 +_f 5$$

$$8 \times_f 17 = 8 +_f 8 +_f 8 +_f \dots \text{(всего 17 чисел 8)} \dots +_f 8$$

Как вам должно быть уже известно, в правой части приведенных выше выражений получается число, принадлежащие конечному множеству F_{19} :

$$5 \times_f 3 = 5 +_f 5 +_f 5 = 15 \% 19 = 15$$

$$8 \times_f 17 = 8 +_f 8 +_f 8 + \dots \text{(всего 17 чисел 8)} \dots +_f 8 =$$

$$(8 \times 17) \% 19 = 136 \% 19 = 3$$

Обратите внимание на то, что результат второй операции умножения не вполне очевиден. Мы обычно не считаем, что $8 \times_f 17 = 3$, но это именно та необходимая часть операции умножения в конечном поле, без которой ее нельзя определить как замкнутую. Это означает, что результат операции умножения в конечном поле всегда принадлежит множеству $\{0, 1, \dots, p-1\}$.

А операцию возведения в степень можно просто определить как многократное умножение числа:

$$7^3 = 7 \times_f 7 \times_f 7 = 343$$

В конечном поле операцию возведения в степень можно выполнить, используя арифметику по модулю, как показано ниже на примере конечного поля F_{19} .

$$7^3 = 343 \% 19 = 1$$

$$9^{12} = 7$$

Операция возведения в степень также дает не вполне очевидные результаты, ведь обычно мы не считаем, что $7^3 = 1$ или $9^{12} = 7$. Но, опять же, конечные поля должны быть определены таким образом, чтобы выполнение

арифметических операций в них *всегда* приводило в конечном итоге к числу, находящемуся в данном поле.

Упражнение 4

Решите приведенные ниже уравнения в конечном поле F_{97} при условии, что операции умножения и возведения в степень выполняются в данном конечном поле.

- $95 \times 45 \times 31$
- $17 \times 13 \times 19 \times 44$
- $12^7 \times 77^{49}$

Упражнение 5

Какое из приведенных ниже множеств чисел находится в конечном поле F_{19} , если $k = 1, 3, 7, 13, 18$? Есть ли в этих множествах что-нибудь примечательное?

$$\{k \times 0, k \times 1, k \times 2, k \times 3, \dots, k \times 18\}$$



Почему поля оказываются простыми

В ответе к упражнению 5 объясняется, почему поля должны содержать количество элементов, составляющее степень *простого* числа. Неважно, какое значение k выбрано, если оно больше нуля, то умножение всего множества на k даст в конечном итоге то же самое множество, что и первоначальное.

Очевидно, что порядок простого числа приводит к равнозначности каждого элемента конечного поля. Если бы порядком множества было составное число, то умножение данного множества на один из делителей привело бы к меньшему множеству.

Программная реализация операции умножения на языке Python

Теперь, когда стало ясно, какой должна быть операция умножения в конечном поле, в классе `FieldElement` можно определить метод `__mul__()`, заменяющий обычную операцию умножения `*`. Такая операция должна выполняться следующим образом: