

Содержание

Предисловие	10
Глава 1. Устраиваемся поудобнее	19
Установка.....	19
Установка Erlang.....	19
Установка Elixir.....	20
Запуск.....	21
Первые шаги.....	22
Навигация по тексту и истории команд.....	23
Навигация по файлам.....	23
Сделаем что-нибудь.....	24
Вызов функций.....	25
Числа в Elixir.....	26
Работа с переменными в оболочке.....	28
Глава 2. Функции и модули	31
Игры с fn.....	31
И &.....	33
Определение модулей.....	34
От модулей к свободным функциям.....	38
Деление кода на модули.....	38
Комбинирование функций с помощью оператора конвейера.....	40
Импортирование функций.....	41
Значения по умолчанию для аргументов.....	42
Документирование кода.....	43
Документирование функций.....	44
Документирование модулей.....	46
Глава 3. Атомы, кортежи и сопоставление с образцом	48
Атомы.....	48
Сопоставление с образцом и атомы.....	49
Логические атомы.....	50

Ограничители.....	52
Обозначайте подчеркиванием все, что не важно	55
Структуры данных: кортежи.....	57
Сопоставление с образцом и кортежи.....	58
Обработка кортежей.....	59
Глава 4. Логика и рекурсия	62
Логика внутри функций.....	62
Конструкция case.....	62
Конструкция cond.....	65
if или else	66
Присваивание значений переменным в конструкциях case и if.....	68
Самый желательный побочный эффект: IO.puts.....	69
Простая рекурсия.....	71
Обратный отсчет	71
Прямой отсчет.....	73
Рекурсия с возвратом значения.....	74
Глава 5. Взаимодействие с человеком	79
Строки.....	79
Многострочные строки	82
Юникод.....	82
Списки символов	83
Строковые метки.....	84
Запрос информации у пользователя.....	85
Ввод символов	85
Чтение строк текста	86
Глава 6. Списки	91
Основы списков.....	91
Деление списков на головы и хвосты	93
Обработка содержимого списков	94
Создание списка из головы и хвоста.....	96
Смешивание списков и кортежей	99
Создание списка списков	99
Глава 7. Пары имя/значение	103
Списки ключей	103
Списки кортежей с несколькими ключами	105

Словари	106
От списков к отображениям.....	107
Создание отображений	108
Изменение отображений.....	108
Чтение отображений.....	109
От отображений к структурам	109
Объявление структур.....	110
Создание и чтение экземпляров структур	110
Использование структур в сопоставлениях с образцом	111
Использование структур в функциях.....	112
Добавление поведения в структуры.....	114
Расширение существующих протоколов.....	116

Глава 8. Функции высшего порядка

и генераторы списков	118
Простые функции высшего порядка.....	118
Создание новых списков с помощью функций высшего порядка.....	121
Получение информации о списке	121
Обработка элементов списка с помощью функций.....	122
Фильтрация значений в списках.....	123
За пределами возможностей генераторов списков	124
Проверка списков	124
Разбиение списков	125
Свертка списков	126

Глава 9. Процессы

Интерактивная оболочка – это процесс.....	129
Порождение процессов из модулей.....	132
Легковесные процессы.....	135
Регистрация процесса.....	136
Когда процесс останавливается	137
Взаимодействие между процессами.....	138
Наблюдение за процессами	141
Наблюдение за движением сообщений между процессами.....	143
Разрыв и установка связей между процессами	145

Глава 10. Исключения, ошибки и отладка.....

Виды ошибок	152
Восстановление работоспособности после ошибок времени выполнения	153

Журналирование результатов выполнения и ошибок	156
Трассировка сообщений.....	157
Трассировка вызовов функций	159

Глава 11. Статический анализ, спецификации типов и тестирование.....

Статический анализ	161
Спецификации типов	164
Модульные тесты.....	167
Настройка тестов	170
Встраивание тестов в документацию	172

Глава 12. Хранение структурированных данных.....

Записи: структурирование данных до появления структур	173
Определение записей	174
Создание и чтение записей.....	175
Использование записей в функциях.....	177
Сохранение данных в долговременном хранилище Erlang.....	179
Создание и заполнение таблицы	181
Простые запросы.....	187
Изменение значений.....	187
Таблицы ETS и процессы.....	188
Следующие шаги.....	190
Хранение записей в Mnesia	191
Настройка базы данных Mnesia.....	191
Создание таблиц.....	192
Чтение данных	196

Глава 13. Основы OTP.....

Создание служб с помощью GenServer	199
Простой супервизор.....	204
Упаковка приложения с помощью Mix.....	209

Глава 14. Расширение языка Elixir с помощью макросов.....

Функции и макросы.....	213
Простой макрос	214
Создание новой логики.....	216
Программное создание функций.....	217
Когда (не) следует использовать макросы.....	219

Глава 15. Phoenix	221
Установка базовых компонентов фреймворка.....	221
Структура простого Phoenix-приложения.....	224
Представление страницы	224
Маршрутизация	225
Простой контроллер.....	227
Простое представление.....	228
Вычисления.....	230
Продвижение Elixir	237
Приложение А. Каталог элементов языка Elixir	239
Команды интерактивной оболочки	239
Зарезервированные слова	240
Операторы	241
Ограничители.....	243
Часто используемые функции.....	244
Приложение В. Создание документации с помощью ExDoc	247
Использование ExDoc вместе с Mix.....	247
Предметный указатель	251
Об авторах	259
Колофон	260

Предисловие

Язык Elixir предлагает разработчикам мощь функционального и конкурентного языка программирования Erlang, но с более дружелюбным синтаксисом, обширным набором библиотек и средствами метапрограммирования. Программный код на языке Elixir компилируется в байт-код Erlang, что позволяет объединять его с программным кодом, написанным на Erlang, и использовать инструменты Erlang. Однако, несмотря на общий фундамент, Elixir создает ощущение совершенно другого языка, более похожего, пожалуй, на Ruby, чем на Prolog, предшественника Erlang.

Книга «Введение в Elixir» познакомит вас с этим мощным языком.



Это издание книги «Введение в Elixir» охватывает версию 1.4 языка. Мы будем обновлять и дополнять его по мере развития Elixir. Если вы найдете ошибки или опечатки, сообщите нам, воспользовавшись системой учета ошибок.

Кому адресована эта книга

Эта книга адресована в основном тем, кто имеет опыт программирования на других языках, но ищет чего-то нового. Может быть, вас интересует весьма практичная и распределенная модель программирования, обладающая преимуществами масштабирования и высокой надежности. Может быть, вы хотите увидеть и понять, что такое «функциональное программирование». Или просто желаете познакомиться с чем-то новым без конкретной цели.

Мы полагаем, что функциональный стиль программирования стал намного более доступным гораздо раньше, чем вы научились программировать с использованием других парадигм. Однако знакомство с Elixir – а иногда даже просто его установка – требует немалых навыков. Поэтому, если вы вообще не имеете опыта программирования, добро пожаловать на борт, но имейте в виду, что на пути вам встретится немало сложностей.

Для кого эта книга не предназначена

Эта книга не предназначена тем, кто торопится достичь поставленной цели.

Если вы уже знакомы с языком Elixir, эта книга, скорее всего, вам не нужна, только если вы не ищете неторопливого, постепенного введения.

Если вы уже знакомы с языком Erlang, эта книга поможет вам увидеть отличия, но, скорее всего, ключевые понятия, обсуждаемые здесь, вам уже знакомы.

Если вы уже знакомы с функциональными языками программирования, это введение покажется вам слишком медленным. Если вы заскучаете, читая эту книгу, смело откладывайте ее в сторону и обратитесь к какой-нибудь другой книге или к электронной документации, рассказывающей о языке в более быстром темпе.

Что даст вам эта книга

С помощью этой книги вы научитесь писать простые программы на Elixir. Узнаете, почему Elixir упрощает создание надежных и легко масштабируемых программ. Здесь вы познакомитесь с терминологией, благодаря чему сможете читать другие ресурсы с информацией о языке Elixir, которые требуют определенного опыта для их понимания.

В теоретическом отношении вы познакомитесь с функциональным программированием. Узнаете, как проектировать программы, основанные на передаче сообщений и рекурсии, и как создавать программы для обработки потоков данных.

Самое важное: эта книга откроет ворота в мир разработки конкурентных программ. Даже притом, что это введение познакомит вас лишь с самыми основами открытой телекоммуникационной платформы (Open Telecom Platform, OTP), знание этих основ поможет вам открыть для себя новый, удивительный мир. После овладения синтаксисом и приемами структурирования программ на языке Elixir вы сможете приступить к созданию надежных и масштабируемых приложений, и это потребует от вас намного меньше усилий, чем при использовании других подходов!

Как читать эту книгу

Эта книга рассказывает историю появления языка Elixir. Наибольшую пользу вы получите, если будете читать все по порядку, хотя бы первый раз, однако вы можете читать выборочно и возвращаться к непочитанным главам, если почувствуете, что это необходимо.

Сначала вы установите Elixir и опробуете его интерактивную оболочку IEx. В этой оболочке вы будете проводить много времени, поэтому постарайтесь овладеть ею в полной мере, чтобы чувствовать себя в ней уютно. Затем вы научитесь загружать программный код в оболочку, чтобы получить возможность писать программы за ее пределами, и узнаете, как вызывать и смешивать этот код.

Вы детально изучите числа, потому что это самые простые представители базовых структур в Elixir. Затем вы познакомитесь с атомами, конструкциями сопоставления с образцом и ограничителями – наиболее распространенными элементами программ. После этого вы перейдете к строкам, спискам и рекурсии, лежащей в основе Elixir. Преодолев несколько тысяч рекурсий вниз и назад, вы перейдете к процессам, ключевым элементам Elixir, которые используют модель обмена сообщениями для поддержки конкуренции и отказоустойчивости.

После освоения основ вы займетесь более детальным изучением приемов отладки и хранения данных, а затем познакомитесь с комплектом инструментов, которые пригодятся вам в разработке программ на языке Elixir: открытой телекоммуникационной платформой (Open Telecom Platform, OTP) языка Erlang, которая охватывает не только телефоны.

Наконец, вы освоите макросы, средства, придающие языку Elixir огромную гибкость и позволяющие расширять его.

Некоторые предпочитают изучать языки программирования, знакомясь с его словарем, списками операторов, управляющих структур и типов данных. Эти списки также приводятся в книге, но они находятся не в основном тексте книги, а в приложении А.

Главное, что должна дать вам эта книга, – научить вас программированию на языке Elixir. Если вы этого не получили, дайте нам знать!

Другие ресурсы

Эта книга, возможно, не лучший для вас способ изучить язык Elixir. Все зависит от того, чему вы хотите научиться и как вы привыкли

осваивать что-то новое. Если вы ищете скоростное введение в язык, обратите внимание на книгу «Programming Elixir» (Pragmatic Publishers), в которой вы быстро познакомитесь с отличительными чертами Elixir. Похожее быстрое и более глубокое введение в Elixir вы также найдете в книгах «Elixir in Action» (Manning) и «The Little Elixir & OTP Guidebook» (Manning). Книга «Metaprogramming Elixir» (Pragmatic Publishers) исследует ключевые возможности языка, а книга «Programming Phoenix» (Pragmatic Publishers) подробно описывает мощный фреймворк на основе Elixir.

Если вам понравятся неспешность и основательность этой книги и у вас появится желание продолжить обучение в том же духе, обратите внимание на книгу «Études for Elixir» (O'Reilly). В ней вы найдете описания коротких программ, которые вы сможете сами написать на Elixir, более сложные, чем примеры в данной книге. Кроме того, ее структура очень похожа на структуру данной книги.

Существует также множество книг, но посвященных не Elixir, а языку Erlang. Однако мы надеемся, что в ближайшее время появится множество новых книг об Elixir. На сайте проекта Elixir вы найдете много руководств, документацию и ссылки на другие ресурсы.

Если в изучении Elixir вами движет желание вырваться из наезженной колеи, обязательно прочитайте книгу Брюса Тейта (Bruce Tate) «Seven Languages in Seven Weeks» (Pragmatic Publishers)¹, в которой дается обзор языков Ruby, Io, Prolog, Scala, Erlang, Clojure и Haskell. Описание Erlang в ней занимает всего 30 с небольшим страниц, но вполне возможно, что это именно то, что вам нужно.

Книги о языке Erlang тоже могут помочь понять, что делает Elixir таким замечательным.

Простое введение в Erlang, во многом похожее на эту книгу, вы найдете в книге «Introducing Erlang» (O'Reilly), которая познакомит вас с Erlang и функциональным программированием.

Попробуйте прочитать электронную версию книги (которую можно также приобрести в бумажном варианте, выпущенном издательством No Starch Books) с остроумными и забавными иллюстрациями «Learn You Some Erlang for Great Good»²!

Также можно порекомендовать классические книги об Erlang с похожими названиями: «Programming Erlang» (Pragmatic Publishers)

¹ *Teim Б.* Семь языков за семь недель. М.: ДМК Пресс, 2014. ISBN: 978-5-94074-539-6. – *Прим. перев.*

² learnyouosomeerlang.com.

и «Erlang Programming»¹ (O'Reilly). Они очень похожи и во многом перекликаются друг с другом, и обе могут послужить прекрасными руководствами, если эта книга покажется вам слишком медлительной или если вам требуется руководство, которое можно использовать как справочник. «Erlang Programming» имеет более практическую направленность, тогда как «Programming Erlang» описывает множество деталей, касающихся настройки окружения для программирования на Erlang.

Более продвинутое руководство «Erlang and OTP in Action» (Mapping) начинается с краткого 72-страничного введения в Erlang и затем обстоятельно описывает применение открытой телекоммуникационной платформы, фреймворка на языке Erlang для создания легко обслуживаемых и расширяемых конкурентных приложений. «Designing for Scalability with Erlang/OTP» (O'Reilly) исследует применение OTP и Erlang для решения задач, которые выглядят чрезвычайно сложными в других окружениях.

Посетите также веб-сайт проекта Erlang², где вы найдете последние обновления, документацию, примеры и многое другое.

Elixir изменит вас

Прежде чем вы продолжите, мы хотим предупредить вас, что использование языка Elixir может кардинально изменить ваши взгляды на программирование. Обычное для этого языка сочетание функционального кода, обработки, ориентированной на процессы, и распределенных вычислений в первый момент может показаться чем-то чужеродным. Однако после освоения Elixir может изменить ваши подходы к решению задач (причем они могут существенно отличаться от подходов, принятых в языке Erlang) и даже усложнить возврат к другим языкам, окружениям и культурам программирования.

Типографские соглашения

В этой книге приняты следующие типографские соглашения:

Курсив

Используется для обозначения новых терминов, адресов электронной почты, имен файлов и расширений имен файлов.

¹ *Чезарини Ф.* Программирование в Erlang. М.: ДМК Пресс, 2015. ISBN: 978-5-97060-160-0. – *Прим. перев.*

² <http://www.erlang.org/>.

Моноширинный шрифт

Применяется для оформления листингов программ и программных элементов внутри обычного текста, таких как имена переменных и функций, инструкций и ключевых слов.

Моноширинный жирный

Обозначает команды или другой текст, который должен вводиться пользователем.

Моноширинный курсив

Обозначает текст, который должен замещаться фактическими значениями, вводимыми пользователем или определяемыми из контекста.



Так обозначаются советы, предложения и примечания общего характера.



Так обозначаются предупреждения и предостережения.



Так выделяются советы и предложения.

Использование программного кода примеров

В этой книге приводится множество примеров, цель которых – познакомить вас с основными идеями. Примеры сделаны максимально короткими, чтобы вам проще было видеть отличия между ними. Вы можете свободно использовать примеры программного кода из этой книги, как посчитаете нужным, но вы не сможете взять код из этой книги и составить из него законченное приложение (если только не испытываете особую нежность к вычислениям скоростей свободно падающих объектов). Тем не менее примеры помогут вам выяснить шаги, которые необходимо проделать для создания приложения. Код примеров можно загрузить из репозитория GitHub¹. (Он также доступен по ссылке **Examples** в каталоге издательства².)

Данная книга призвана оказать вам помощь в решении ваших задач. Вам не нужно обращаться в издательство за разрешением, если

¹ <https://github.com/simonstl/introducing-elixir>.

² <http://shop.oreilly.com/product/0636920050612.do>.

вы не собираетесь воспроизводить существенные части программного кода. Например, если вы разрабатываете программу и используете в ней несколько отрывков программного кода из книги, вам не нужно обращаться за разрешением. Однако в случае продажи или распространения компакт-дисков с примерами из этой книги вам необходимо получить разрешение от издательства O'Reilly. Если вы отвечаете на вопросы, цитируя данную книгу или примеры из нее, получение разрешения не требуется. Но при включении существенных объемов программного кода примеров из этой книги в вашу документацию вам необходимо будет получить разрешение издательства.

Мы приветствуем, но не требуем добавлять ссылку на первоисточник при цитировании. Под ссылкой на первоисточник мы подразумеваем указание авторов, издательства и ISBN. Например: «Introducing Elixir, Second Edition, by Simon St. Laurent and J. David Eisenberg (O'Reilly). Copyright 2017 Simon St. Laurent and J. David Eisenberg, 978-1-491-95677-9».

За получением разрешения на использование значительных объемов программного кода примеров из этой книги обращайтесь по адресу permissions@oreilly.com.

Дополнительная помощь

Мы очень надеемся, что вам понравится эта книга и вы сможете самостоятельно освоить ее, мы также надеемся, что вы сможете оказать помощь другим читателям, изучающим Elixir. Вы можете оказать такую помощь множеством способов:

- если вы обнаружите техническую неточность, невнятное описание или что-то, что можно улучшить, сообщите об этом через систему регистрации ошибок (<http://bit.ly/elixir-2e-errata>);
- если вам понравилась (или не понравилась) книга, пожалуйста, оставьте свой отзыв. Наиболее предпочтительным местом для этого являются сайт Amazon.com и страница книги на сайте издательства O'Reilly. Подробное описание, что понравилось и что не понравилось, пригодится другим читателям и нам;
- если вы считаете, что можете рассказать о языке Elixir намного больше, поделитесь своими знаниями, например в веб или в своей книге, на учебных курсах или каким-то другим способом, наиболее удобным для вас.

Мы будем исправлять опечатки и ошибки в книге и постараемся своевременно откликаться на проблемы, поднимаемые в отзывах.

Даже после завершения книги мы, скорее всего, будем продолжать добавлять в нее абзацы и разделы. Если вы приобрели электронную версию книги, вы будете получать эти дополнения бесплатно, по крайней мере до того момента, пока не выйдет новое издание. Мы не обещаем скорого выхода нового издания, если только в мире Elixir не произойдут существенные перемены.

Надеемся, что эта книга увлечет вас настолько, что вам захочется поделиться своими знаниями.

Используйте ее во благо

У каждого из нас есть свое понимание, что такое «во благо». Тем не менее, пожалуйста, постарайтесь использовать мощь Elixir для создания проектов, улучшающих наш мир или, по крайней мере, не ухудшающих его.

Как связаться с нами

С вопросами и предложениями, касающимися этой книги, обращайтесь в издательство:

O'Reilly Media, Inc.

1005 Gravenstein Highway North Sebastopol, CA 95472

800-998-9938 (США или Канада)

707-829-0515 (международный и местный)

707-829-0104 (факс)

Список опечаток, файлы с примерами и другую дополнительную информацию вы найдете на странице книги http://bit.ly/introducing_elixir_2e.

Свои пожелания и вопросы технического характера отправляйте по адресу bookquestions@oreilly.com.

Дополнительную информацию о книгах, обсуждения, конференции и новости вы найдете на веб-сайте издательства: <http://www.oreilly.com>.

Ищите нас на Facebook: <http://facebook.com/oreilly>.

Следуйте за нами в Твиттере: <http://twitter.com/oreillymedia>.

Смотрите нас на YouTube: <http://www.youtube.com/oreillymedia>.

Благодарности

Вокруг Elixir сплотилось удивительно дружелюбное сообщество, открытое для вопросов и предложений по широкому кругу проблем. Мы были счастливы получать ответы на свои вопросы и наслаждались общением с членами сообщества, искренне старавшимися объяснить сложные проблемы на примерах кода.

Хосе Валима (José Valim) оказывал нам неоценимую помощь по всем аспектам в книге. Наш товарищ и конкурент Дейв Томас (Dave Thomas) подтвердил, что да, Elixir жив и мир ждет его. Со стороны Erlang Франческо Цезарини (Francesco Cesarini) приветствовал наше желание рассказать об этом новом, братском языке. Обозреватели: Андреа Леопарди (Andrea Leopardi), Мэтт Миллз (Matt Mills), Байбек Пандей (Bibek Pandey), Алексей Шолик (Alexei Sholik), Дэвид Лоренцетти (David Lorenzetti), Бенгт Клеберг (Bengt Kleberg), Мистраль Контрастин (Mistral Contrastin), Оджи де Блайк мл. (Augie De Blicek Jr.), Эри ван Вингерден (Arie van Wingerden), Элиас Каррильо (Elias Carrillo) и Николас (Nicholas) – помогли нам в поиске ошибок.

Меган Бланшетт (Meghan Blanchette) помогала в работе над первым изданием. Морин Спенсер (Maureen Spencer) и Хезер Шерер (Heather Scherer) оказали ценную помощь с подготовкой второго издания.

Кроме того, участие Дж. Дэвида Эйзенберга (J. David Eisenberg) в проекте не раз спасало Симона Сенлорена (Simon St. Laurent)!

Спасибо также Симону, который дал Дэвиду первый и приятный опыт работы над книгой.

Глава 1

Устраиваемся поудобнее

Изучение Elixir удобнее всего начинать в интерактивной оболочке Interactive Elixir (IEx). Этот интерфейс командной строки – лучшее средство, чтобы выяснить, что возможно и что невозможно в Elixir. Он поможет избавиться от головной боли в будущем, поэтому устраивайтесь поудобнее!

Установка

Elixir действует поверх Erlang, поэтому для начала необходимо установить Erlang в системе, а затем Elixir.

Установка Erlang

Установка Erlang в Windows выполняется очень просто (<http://www.erlang.org/downloads>). Загрузите двоичный файл для Windows, запустите установку – и все! Если вы – храбрый новичок, только начинающий осваивать программирование, для вас такой вариант станет лучшим выбором.

Пользователи Linux или macOS могут загрузить исходный код и скомпилировать его. В macOS придется распаковать архив с исходным кодом, затем перейти в каталог, созданный в процессе распаковки, и выполнить последовательность команд: `./configure`, `make` и `sudo make install`. Однако эта простая последовательность действий увенчается успехом, только если в системе установлены все необходимые инструменты и библиотеки, в противном случае есть риск получить множество малопонятных ошибок. В частности,

в новейших версиях XCode компания Apple использует компилятор LLVM вместо GCC, соответственно, велика вероятность отсутствия компилятора GCC в новейших версиях системы macOS, необходимого для сборки Erlang.

Вы можете спокойно игнорировать ошибки, вызванные отсутствием пакета FOP, который Erlang использует для создания документации в формате PDF – вы сможете загрузить ее отдельно. Кроме того, в новейших версиях macOS в конце может появиться ошибка, сообщающая, что wxWidgets не работает в 64-битных версиях системы. Ее тоже можно безопасно игнорировать, пока.

Если по каким-то причинам компиляция из исходного кода невозможна или нежелательна, на сайте Erlang Solutions (<https://www.erlang-solutions.com/resources/download.html>) можно найти большое количество установочных пакетов для наиболее распространенных версий дистрибутивов. Кроме того, многие диспетчеры пакетов (Debian, Ubuntu, MacPorts, Homebrew и др.) включают Erlang. Это может быть не самая последняя версия, но наличие любой действующей версии Erlang намного лучше ее отсутствия. Обычно распространители дистрибутивов стремятся включать в свои репозитории самые свежие версии пакетов, поэтому если у вас не заладилось с компиляцией, обратите внимание на диспетчеры пакетов.



Во многих системах Erlang все чаще включается в установку по умолчанию, в том числе в Ubuntu, в основном благодаря широкому распространению CouchDB.

Установка Elixir

Завершив установку Erlang, загрузите скомпилированную версию Elixir (<https://github.com/elixir-lang/elixir/releases>) или исходный код (<https://github.com/elixir-lang/elixir/tags>) из репозитория GitHub. Некоторые диспетчеры пакетов уже включают Elixir, в том числе Homebrew. Примеры, представленные в этой книге, должны без проблем выполняться в Elixir 1.3.0.

Затем настройте переменную окружения PATH, добавив в нее путь к каталогу *elixir/bin*.

Инструкции по настройке Elixir можно найти в руководстве, по адресу: <http://elixir-lang.org/getting-started/introduction.html>¹.

¹ На русском языке: <http://elixir-lang.ru/install>. – *Прим. перев.*

Запуск

Перейдите в командную строку (например, откройте окно терминала) и введите команду `mix new first_app`. Она запустит инструмент Mix (<https://hexdocs.pm/mix/Mix.html>), который «реализует задачи создания, компиляции и тестирования проектов на языке Elixir, управления их зависимостями и многие другие». В данном случае введенная команда создаст новый, пустой проект в каталоге с именем *first_app*:

```
$ mix new first_app
* creating README.md
* creating .gitignore
* creating mix.exs
* creating config
* creating config/config.exs
* creating lib
* creating lib/first_app.ex
* creating test
* creating test/test_helper.exs
* creating test/first_app_test.exs
```

Your Mix project was created successfully.
You can use “mix” to compile it, test it, and more:

```
cd first_app
mix test
```

Run “mix help” for more commands.

```
$
```



Не забудьте добавить путь к каталогу с выполняемым файлом Elixir в переменную окружения `PATH`, чтобы инструмент Mix смог найти его.

Не будем тратить время на компиляцию и тестирование пустого проекта, а сразу перейдем в каталог *first_app* и запустим оболочку IElixir, выполнив следующие команды:

```
$ cd first_app
$ iex -S mix
```

Вы увидите на экране примерно следующие строки, возможно, с курсором в строке приглашения `iex(1)>`. Обратите внимание, что везде, где это необходимо, длинные строки переформатированы, чтобы уместить их по ширине страницы:

```
$ cd first_app
[david@localhost first_app]$ iex -S mix
Erlang/OTP 19 [erts-8.0] [source] [64-bit] [smp:4:4] [async-threads:10] [hipe]
```

```
[kernel-poll:false]
Compiling 1 file (.ex)
Generated first_app app
Interactive Elixir (1.3.1) - press Ctrl+C to exit (type h() ENTER for help)
iex(1)>
```

Вот мы и в Elixir! (Первая строка с упоминанием Erlang подчеркивает, что Elixir действует внутри Erlang. Но не беспокойтесь об этом!)

Первые шаги

Прежде чем окунуться в захватывающий процесс программирования на языке Elixir, всегда имеет смысл научиться выходить из оболочки. Нажатие комбинации **Ctrl+C** приведет к появлению меню. Если в этом меню ввести «a», IEx завершит работу, и вы увидите приглашение к вводу командной оболочки системы, в которой вы запускали IEx:

```
iex(1)>
BREAK: (a)bort (c)ontinue (p)roc info (i)nfo (l)oaded
       (v)ersion (k)ill (D)b-tables (d)istribution
a
$
```

В оболочке iex можно также попросить вывести справку (после повторного запуска), введя h() или просто h¹:

```
iex(1)> h()
# IEx.Helpers
```

IEx.Helpers

Добро пожаловать в интерактивную оболочку Interactive Elixir. Сейчас вы смотрите документацию для модуля IEx.Helpers, включающего множество вспомогательных функций, делающих работу с интерактивной оболочкой Elixir более удобной.

Это сообщение сгенерировано вызовом вспомогательной функции h(), которую обычно обозначают как h/θ (поскольку она принимает θ аргументов).

Функцию h можно использовать для вызова документации к любому модулю Elixir или функции:

¹ Здесь и далее в книге справочный текст приводится в переводе на русский язык, но имейте в виду, что в действительности вся документация в Elixir на английском языке. — *Прим. перев.*

```
| h Enum
| h Enum.map
| h Enum.reverse/1
```

Для исследования любого значения, созданного в интерактивной оболочке, можно также использовать функцию `i`:

```
| i "hello"
```

Существует большое количество других вспомогательных функций:

```
...
:ok
```

Что, собственно, произошло? Мы выполнили команду `!ex`, вызвали вспомогательную функцию `h()`, которая вывела перед вами некоторую справочную информацию. Она выводит на экран большой объем текста и завершает вывод, возвращая `:ok`.

Навигация по тексту и истории команд

Занявшись исследованием интерактивной оболочки, вы обнаружите, что своим действием она во многом напоминает другие командные оболочки или Emacs. Клавиши со стрелками влево и вправо перемещают курсор назад и вперед по редактируемой строке. Некоторые комбинации клавиш повторяют действие этих же комбинаций в текстовом редакторе Emacs. Комбинация **Ctrl+A** перенесет курсор в начало строки, а **Ctrl+E** – в конец. Если вы по ошибке ввели два символа не в той последовательности, нажмите **Ctrl+T**, чтобы поменять их местами.

Кроме того, после ввода закрывающей круглой, квадратной или фигурной скобки будет подсвечена парная ей открывающая скобка.

Клавиши со стрелками вверх и вниз выполняют навигацию по истории команд, упрощая их повторный запуск. На конкретную команду в истории можно также сослаться как `v(N)`, где `N` – номер строки.

Навигация по файлам

Оболочка IEx поддерживает некоторые простые операции с файловой системой, потому что вам может потребоваться переключаться между файлами, составляющими вашу программу. Команды имеют те же имена, что и команды Unix, но записываются как вызовы функций. Первоначально текущим рабочим каталогом оболочки IEx является каталог, в котором вы ее запустили. Узнать путь к этому каталогу можно с помощью `pwd()`:

```
iex(1)> pwd()  
/Users/elixir/code/first_app  
:ok
```

Для перехода в другой каталог используйте функцию `cd()`, но вы обязательно должны заключить ее аргумент в двойные кавычки:

```
iex(2)> cd ".."  
/Users/elixir/code  
:ok  
iex(3)> cd "first_app"  
/Users/elixir/code/first_app  
:ok
```

Получить список с содержимым каталога можно с помощью команды `ls()`, которая в случае вызова без аргумента вернет список содержимого текущего каталога, а с аргументом – список содержимого указанного каталога.

Сделаем что-нибудь

Проще всего начать игру с Elixir – использовать интерактивную оболочку как калькулятор. В отличие от обычной командной строки, она позволяет вводить арифметические выражения и получать результаты:

```
iex(1)> 2 + 2  
4  
iex(2)> 27 - 14  
13  
iex(3)> 35 * 42023943  
1470838005  
iex(4)> 4 * (3 + 5)  
32  
iex(5)> 200 / 15  
13.333333333333334
```

Первые три оператора – сложение (+), вычитание (-) и умножение (*) – действуют как обычно при работе с целыми или вещественными числами. Круглые скобки позволяют изменять порядок выполнения операторов, как показано в строке 4. (Порядок выполнения операций приводится в приложении А.) Четвертый оператор, /, возвращает вещественный результат (число с дробной частью), как показано в строке 5. Операторы необязательно окружать пробелами, но они делают код более читаемым.

Вызов функций

Функции – это блоки логики, которые принимают аргументы и возвращают значение. Проще всего начать исследование с математических функций. Например, если требуется получить целочисленный результат деления (целочисленных аргументов), используйте функцию `div()` вместо оператора `/` и функцию `rem()`, чтобы получить остаток, как показано в строках 6 и 7:

```
iex(6)> div(200,15)
13
iex(7)> rem(200, 15)
5
iex(8)> rem 200, 15
5
```



Строка 8 демонстрирует особенность синтаксиса Elixir: необязательность заключения аргументов в круглые скобки. Если вы считаете, что круглые скобки улучшают читаемость кода, используйте их. Если ввод круглых скобок кажется вам лишней работой, просто не вводите их. Elixir интерпретирует пробел, следующий за именем функции, как открывающую круглую скобку, автоматически закрывающуюся в конце строки. Когда подобный стиль приводит к неожиданным результатам, Elixir может попросить вас «не вставлять пробелы между именем функции и открывающей круглой скобкой».

Elixir допускает передачу целых чисел вместо вещественных, но вещественные числа не всегда можно использовать вместо целых. Если вам потребуется преобразовать вещественное число в целое, используйте встроенную функцию `round()`:

```
iex(9)> round 200/15
13
```

Функция `round()` отбрасывает дробную часть числа. Если дробная часть больше или равна `.5`, целая часть увеличивается на 1, округляя число вверх. Если вам требуется просто отбросить дробную часть, оставив целую, используйте функцию `trunc()`.

Сослаться на предыдущую команду можно по номеру строки, с помощью функции `v()`. Например:

```
iex(10)> 4*v(9)
52
```

Команда в строке 9 возвращает результат 13, соответственно, выражение `4*13` дает в результате 52.



Если вам интересно, для ссылки на предыдущие команды можно использовать отрицательные числа. `v(-1)` вернет предыдущий результат, `v(-2)` – результат, полученный перед предыдущим, и так далее.

Для выполнения более сложных вычислений Elixir позволяет использовать модуль `math` языка Erlang, содержащий классический набор функций, поддерживаемых научным калькулятором. Эти функции возвращают вещественные значения. Константа «пи» доступна как функция `:math.pi()`. Доступны также тригонометрические, логарифмические, экспоненциальные функции, функция квадратного корня и (исключая Windows) даже функция гауссовой ошибки. (Тригонометрические функции принимают аргументы в радианах, а не в градусах, поэтому при необходимости вам придется выполнять соответствующие преобразования.) При использовании этих функций вам придется также вводить префикс `:math.`, но это не слишком высокая плата за удобство.

Например, ниже показано, как получить синус от нуля радиан:

```
iex(11)> :math.sin(0)
0.0
```

Обратите внимание, что в результате получилось число `0.0`, а не просто `0`. Это указывает, что результат является вещественным числом. (И да, вы можете то же самое вычисление выполнить командой `:math.sin 0` без круглых скобок.)

Ниже показано, как получить косинус от π и 2π радиан:

```
iex(12)> :math.cos(:math.pi())
-1.0
iex(13)> :math.cos(2 * :math.pi())
1.0
```

Чтобы вычислить 2 в степени 16, используйте функцию:

```
iex(14)> :math.pow(2,16)
65536.0
```

Полный перечень функций, имеющих в модуле `math` и доступных в Elixir, вы найдете в приложении А.

Числа в Elixir

Elixir распознает два вида чисел: целые и вещественные (их также часто называют числами с плавающей точкой). В отличие от целых чисел, не имеющих дробной части, вещественные числа включают


```
iex(6)> :math.sin(0)
0.0
iex(7)> :math.sin(:math.pi())
1.2246467991473532e-16
```

Если бы в Elixir использовалось более точное представление числа π и вычисления с вещественными числами выполнялись бы с большей точностью, результат в строке 7 был бы ближе к нулю.

Для финансовых вычислений лучше всего использовать целые числа. Используйте наименьшую единицу измерения – например, копейки – и помните, что одна копейка – это 1/100 рубля. (Финансовые сделки могут выполняться в более мелких единицах, но и в этом случае желательно использовать целые числа и при необходимости применять известный множитель.) Однако для более сложных вычислений вполне можно использовать вещественные числа, просто помните, что результат получается не совсем точным.

Elixir поддерживает операции с целыми числами еще в нескольких системах счисления, кроме десятичной. Например, если понадобится, двоичное значение 1010111 можно записать как:

```
iex(8)> 0b01010111
87
```

Elixir вернет значение в виде десятичного числа. Аналогично можно указать шестнадцатеричное (в системе счисления с основанием 16) число, используя префикс `x` вместо `b`:

```
iex(9)> 0xcafe
51966
```

Чтобы ввести отрицательное число, просто добавьте знак минус (-) перед ним. Этот прием можно использовать с целыми числами, а также шестнадцатеричными, двоичными и вещественными числами:

```
iex(10)> -1234
-1234
iex(11)> -0xcafe
-51966
iex(12)> -2.045234324e6
-2045234.324
```

Работа с переменными в оболочке

Функция `v()` позволяет ссылаться на результаты предыдущих вычислений, но это не самый удобный способ хранения числовых ре-

зультатов, и к тому же функция $v()$ работает только в интерактивной оболочке – это не универсальный механизм. Намного разумнее хранить значения в памяти, давая им текстовые имена, то есть создавая переменные.

Имена переменных в Elixir начинаются с буквы в нижнем регистре или символа подчеркивания. Обычные переменные начинаются с буквы в нижнем регистре, а «необязательные» – с символа подчеркивания (см. раздел «Обозначайте подчеркиванием все, что не важно» в главе 3). Но пока давайте придерживаться обычных переменных. Синтаксис присваивания значений переменным должен быть вам знаком по школьному курсу алгебры или по другим языкам программирования, например:

```
iex(1)> n=1
1
```

Чтобы узнать значение переменной, просто введите ее имя:

```
iex(2)> n
1
```

Elixir, в отличие от многих других функциональных языков программирования (включая Erlang), позволяет присваивать переменным новые значения:

```
iex(3)> n=2
2
iex(4)> n=n+1
3
```

Elixir сначала выполняет выражение справа, затем оператор $=$ сопоставляет результат с левой стороной. Он присвоит переменной n новое значение, если попросить его об этом, и даже будет использовать прежнее значение n в выражении справа, чтобы получить новое значение. Инstrukция $n=n+1$ означает: «присвоить значение выражения $n+1$, которое равно 3, переменной n ».

В операции присваивания все вычисления должны производиться справа от знака «равно». Даже если известно, что m имеет значение 6, Elixir не сможет выполнить инструкцию $2*m = 3*4$:

```
iex(5)> 2*m=3*4
** (CompileError) iex:12: illegal pattern
```

Оболочка IEx будет помнить переменные, пока вы не попросите ее забыть их.

Кроме того, в одну строку можно включить несколько инструкций, разделив их точкой с запятой (;). Синтаксически она действует подобно окончанию строки:

```
iex(6)> distance = 20; gravity = 9.8
9.8
iex(7)> distance
20
iex(8)> gravity
9.8
```

IElixir сообщит только значение последней инструкции, но, как можно видеть в строках 7 и 8, все переменные получили соответствующие значения.

Если в оболочке накопилось много ненужного, вызовите `clear`. Эта функция просто очистит экран.

Прежде чем перейти к следующей главе, где рассказывается о модулях и функциях, поиграйте немного в IElixir. Полученный опыт, даже такой простой, поможет вам увереннее двигаться вперед. Попробуйте использовать переменные и посмотрите, что происходит с большими целыми числами. Elixir прекрасно справляется с поддержкой даже гигантских чисел. Попробуйте смешивать в вычислениях целые и вещественные числа и посмотрите, что из этого получится. Все это должно быть для вас не очень сложно.