



ОГЛАВЛЕНИЕ

Введение	12
Аудитория	14
О третьем издании	15
Другие ресурсы	16
Некоторые типографские соглашения	17
Запуск примеров	17
Благодарности	18
ЧАСТЬ I	
Язык	19
Глава 1. Начинаем	20
1.1. Блоки	20
1.2. Некоторые лексические соглашения	22
1.3. Глобальные переменные	24
1.4. Отдельный интерпретатор	24
Упражнения	26
Глава 2. Типы и значения	27
2.1. Nil	28
2.2. Boolean (логические значения)	28
2.3. Числа	28
2.4. Строки	30
Литералы	30
Длинные строки	32
Приведения типов	33
2.5. Таблицы	34
2.6. Функции	38
2.7. userdata и нити	38
Упражнения	39
Глава 3. Выражения	40
3.1. Арифметические операторы	40
3.2. Операторы сравнения	41
3.3. Логические операторы	42

3.4. Конкатенация.....	43
3.5. Оператор длины.....	43
3.6. Приоритеты операторов	45
3.7. Конструкторы таблиц	46
Упражнения	48
Глава 4. Операторы.....	49
4.1. Операторы присваивания	49
4.2. Локальные переменные и блоки.....	50
4.3. Управляющие конструкции	52
if then else	53
while	53
repeat	54
Числовой оператор for.....	54
Оператор for общего вида	55
4.4. break, return и goto	57
Упражнения	60
Глава 5. Функции	62
5.1. Множественные результаты.....	64
5.2. Функции с переменным числом аргументов.....	68
5.3. Именованные аргументы	70
Упражнения	72
Глава 6. Еще о функциях	73
6.1. Замыкания	75
6.2. Неглобальные функции	79
6.3. Оптимизация хвостовых вызовов	82
Упражнения	83
Глава 7. Итераторы и обобщенный for	85
7.1. Итераторы и замыкания.....	85
7.2. Семантика обобщенного for.....	87
7.3. Итераторы без состояния	89
7.4. Итераторы со сложным состоянием.....	91
7.5. Подлинные итераторы (true iterarators).....	93
Упражнения	94
Глава 8. Компиляция, выполнение и ошибки	96
8.1. Компиляция	96
8.2. Предкомпилированный код.....	100
8.3. Код на C	102
8.4. Ошибки	103
8.5. Обработка ошибок и исключений.....	105

8.6. Сообщения об ошибках и стек вызовов	106
Упражнения	108
Глава 9. Сопрограммы	110
9.1. Основы сопрограмм	110
9.2. Каналы и фильтры	113
9.3. Сопрограммы как итераторы	117
9.4. Невытесняющая многонитевость	119
Упражнения	124
Глава 10. Законченные примеры	125
10.1. Задача о восьми королевах	125
10.2. Самые часто встречающиеся слова	127
10.3. Цепь Маркова	129
Упражнения	131
ЧАСТЬ II	
Таблицы и объекты	133
Глава 11. Структуры данных	134
11.1. Массивы	134
11.2. Матрицы и многомерные массивы	135
11.3. Связанные списки	137
11.4. Очереди и двойные очереди	138
11.5. Множества и наборы	139
11.6. Строчные буферы	141
11.7. Графы	142
Упражнения	144
Глава 12. Файлы данных и персистентность	146
12.1. Файлы с данными	146
12.2. Сериализация	148
Сохранение таблиц без циклов	151
Сохранение таблиц с циклами	152
Упражнения	155
Глава 13. Метатаблицы и метаметоды	156
13.1. Арифметические метаметоды	157
13.2. Метаметоды сравнения	160
13.3. Библиотечные метаметоды	161
13.4. Метаметоды для доступа к таблице	162
Метаметод <code>__index</code>	163
Метаметод <code>__newindex</code>	164
Таблицы со значениями по умолчанию	165

Отслеживание доступа к таблице	166
Таблицы, доступные только для чтения.....	168
Упражнения	169
Глава 14. Окружение	170
14.1. Глобальные переменные с динамическими именами.....	170
14.2. Описания глобальных переменных.....	172
14.3. Неглобальные окружения	174
14.4. Использование <code>_ENV</code>	176
14.5. <code>_ENV</code> и <code>load</code>	179
Упражнения	181
Глава 15. Модули и пакеты	182
15.1. Функция <code>require</code>	184
Переименовывание модуля.....	185
Поиск по пути	186
Искатели файлов.....	187
15.2. Стандартный подход для написания модулей на Lua	188
15.3. Использование окружений	190
15.4. Подмодули и пакеты.....	192
Упражнения	193
Глава 16. Объектно-ориентированное программирование	195
16.1. Классы	197
16.2. Наследование	199
16.3. Множественное наследование	201
16.4. Скрытие	203
16.5. Подход с единственным методом	205
Упражнения	206
Глава 17. Слабые таблицы и финализаторы	208
17.1. Слабые таблицы.....	208
17.2. Функции с кэшированием	210
17.3. Атрибуты объекта.....	212
17.4. Опять таблицы со значениями по умолчанию	213
17.5. Эфемерные таблицы	215
17.6. Финализаторы	216
Упражнения.....	220
ЧАСТЬ III	
Стандартные библиотеки	221
Глава 18. Математическая библиотека.....	222

Упражнения	223
Глава 19. Библиотека для побитовых операций ...	224
Упражнения	227
Глава 20. Библиотека для работы с таблицами	228
20.1. Функции insert и remove	228
20.2. Сортировка	229
20.3. Конкатенация	230
Упражнения	231
Глава 21. Библиотека для работы со строками.....	232
21.1. Основные функции для работы со строками	232
21.2. Функции для работы с шаблонами	235
Функция string.find	235
Функция string.match	236
Функция string.gsub	236
Функция string.gsub	237
21.3. Шаблоны	238
21.4. Захваты	242
21.5. Замены	245
Кодировка URL	246
Замена табов	248
21.6. Хитрые приемы	249
21.7. Юникод	252
Упражнения	255
Глава 22. Библиотека ввода/вывода	256
22.1. Простая модель ввода/вывода	256
22.2. Полная модель ввода/вывода	260
Небольшой прием для увеличения быстродействия	261
Бинарные файлы	262
22.3. Другие операции над файлами	264
Упражнения	266
Глава 23. Библиотека функций операционной системы.....	267
23.1. Дата и время	267
23.2. Другие вызовы системы	270
Упражнения	271
Глава 24. Отладочная библиотека.....	272
24.1. Возможности по доступу (интроспекции)	272
Доступ к локальным переменным	275
Доступ к нелокальным переменным	276

Доступ к другим сопрограммам	277
24.2. Ловушки (hooks)	278
24.3. Профилирование	279
Упражнения	282

ЧАСТЬ IV

С API..... 283

Глава 25. Обзор С API 284

25.1. Первый пример	286
25.2. Стек	288
Помещение элементов на стек	290
Обращение к элементам	291
Другие операции со стеком	294
25.3. Обработка ошибок в С API	295
Обработка ошибок в коде приложения	296
Обработка ошибок в коде библиотек	296
Упражнения	297

Глава 26. Расширение вашего приложения 298

26.1. Основы	298
26.2. Работа с таблицами	300
26.3. Вызовы функций на Lua	305
26.4. Обобщенный вызов функции	307
Упражнения	309

Глава 27. Вызываем С из Lua 310

27.1. Функции на С	310
27.2. Продолжения	313
27.3. Модули на С	316
Упражнения	318

Глава 28. Приемы написания функций на С 320

28.1. Работа с массивами	320
28.2. Работа со строками	322
28.3. Сохранение состояния в функциях на С	326
Реестр	326
Значения, связанные с функцией	329
Значения, связанные с функцией, используемые несколькими функциями	332
Упражнения	333

Глава 29. Задаваемые пользователем типы в С ... 334

29.1. Пользовательские данные (userdata)	335
29.2. Мета таблицы	337

29.3. Объектно-ориентированный доступ	340
29.4. Доступ как к обычному массиву.....	342
29.5. Легкие объекты типа userdata (light userdata).....	344
Упражнения	345
Глава 30. Управление ресурсами.....	346
30.1. Итератор по каталогу	346
30.2. Парсер XML.....	349
Упражнения	358
Глава 31. Нити и состояния.....	360
31.1. Многочисленные нити.....	360
31.2. Состояния Lua.....	365
Упражнения	373
Глава 32. Управление памятью.....	374
32.1. Функция для выделения памяти	374
32.2. Сборщик мусора	377
API сборщика мусора	378
Упражнения	380



ВВЕДЕНИЕ

Когда Вальдемар, Луис и я начали разработку Lua в 1993 году, мы с трудом могли себе представить, что Lua так распространится. Начавшись как домашний язык для двух специфичных проектов, сейчас Lua широко используется во всех областях, которые могут получить выигрыш от простого, расширяемого, переносимого и эффективного скриптового языка, таких как встроенные системы, мобильные устройства и, конечно, игры.

Мы разработали Lua с самого начала для интегрирования с программным обеспечением, написанным на C/C++ и других распространенных языках. Эта интеграция несет с собой много преимуществ. Lua – это крошечный и простой язык, частично из-за того, что он не пытается делать то, в чем уже хорош C, например быстрдействие, низкоуровневые операции и взаимодействие с программами третьих сторон. Для этих задач Lua полагается на C. Lua предлагает то, для чего C недостаточно хорош: достаточная удаленность от аппаратного обеспечения, динамические структуры, отсутствие избыточности и легкость тестирования и отладки. Для этих целей Lua располагает безопасным окружением, автоматическим управлением памятью и хорошими возможностями для работы со строками и другими типами данных с изменяемым размером.

Часть силы Lua идет от его библиотек. И это не случайно. В конце концов, одной из главных сил Lua является расширяемость. Многие особенности языка вносят в это свой вклад. Динамическая типизация предоставляет большую степень полиморфизма. Автоматическое управление памятью упрощает интерфейсы, поскольку нет необходимости решать, кто отвечает за выделение и освобождение памяти или как обрабатывать переполнения. Функции высших порядков и анонимные функции позволяют высокую степень параметризации, делая функции более универсальными.

В большей степени, чем расширяемым языком, Lua является «склеивающим» (*glue*) языком. Lua поддерживает компонентный подход к разработке программного обеспечения, когда мы создаем приложе-

ние, склеивая вместе существующие высокоуровневые компоненты. Эти компоненты написаны на компилируемом языке со статической типизацией, таком как C/C++; Lua является «клеем», который мы используем для компоновки и соединения этих компонентов. Обычно компоненты (или объекты) представляют более конкретные низкоуровневые сущности (такие как виджеты и структуры данных), которые почти не меняются во время разработки программы и которые занимают основную часть времени выполнения итоговой программы. Lua придает итоговую форму приложению, которая, скорее всего, сильно меняется во время жизни данного программного продукта. Однако, в отличие от других «склеивающих» технологий, Lua является полноценным языком программирования. Поэтому мы можем использовать Lua не только для «склеивания» компонентов, но и для адаптации и настройки этих компонентов, а также для создания полностью новых компонентов.

Конечно, Lua не единственный скриптовый язык. Существуют другие языки, которые вы можете использовать примерно для тех же целей. Тем не менее Lua предоставляет целый набор возможностей, которые делают его лучшим выбором для многих задач и дает ему свой уникальный профиль:

- *Расширяемость.* Расширяемость Lua настолько велика, что многие рассматривают Lua не как язык, а как набор для построения DSL (domain-specific language, язык, созданный для определенной области, применения). Мы разрабатывали Lua с самого начала, чтобы он был расширяемым как через код на Lua, так и через код на C. Как доказательство Lua реализует большую часть своей базовой функциональности через внешние библиотеки. Взаимодействие с C/C++ действительно просто, и Lua был успешно интегрирован со многими другими языками, такими как Fortran, Java, Smalltalk, Ada, C#, и даже со скриптовыми языками, такими как Perl и Python.
- *Простота.* Lua – это простой и маленький язык. Он основан на небольшом числе понятий. Эта простота облегчает изучение. Lua вносит свой вклад в то, что его размер очень мал. Полный дистрибутив (исходный код, руководство, бинарные файлы для некоторых платформ) спокойно размещается на одном флоппи-диске.
- *Эффективность.* Lua обладает весьма эффективной реализацией. Независимые тесты показывают, что Lua – один из самых быстрых языков среди скриптовых языков.

- *Портируемость.* Когда мы говорим о портируемости, мы говорим о запуске Lua на всех платформах, о которых вы только слышали: все версии Unix и Windows, PlayStation, Xbox, Mac OS X и iOS, Android, Kindle Fire, NOOK, Haiku, QUALCOMM Brew, большие серверы от IBM, RISC OS, Symbian OS, процессоры Rabbit, Raspberry Pi, Arduino и многое другое. Исходный код для каждой из этих платформ практически одинаков. Lua не использует условную компиляцию для адаптации своего кода под различные машины, вместо этого он придерживается стандартного ANSI (ISO) C. Таким образом, вам обычно не нужно адаптировать его под новую среду: если у вас есть компилятор с ANSI C, то вам просто нужно откомпилировать Lua.

Аудитория

Пользователи Lua обычно относятся к одной из трех широких групп: те, кто используют Lua, уже встроенный в приложение, те, кто используют Lua отдельно от какого-либо приложения (standalone), и те, кто используют Lua и C вместе.

Многие используют Lua, встроенный в какое-либо приложение, например в Adobe Lightroom, Nmap или World of Warcraft. Эти приложения используют Lua-C API для регистрации новых функций, создания новых типов и изменения поведения некоторых операций языка, конфигурируя Lua для своей области. Часто пользователи такого приложения даже не знают, что Lua – это независимый язык, адаптированный под данную область. Например, многие разработчики плагинов для Lightroom не знают о других использованиях этого языка; пользователи Nmap обычно рассматривают Lua как скриптовый язык Nmap; игроки в World of Warcraft могут рассматривать Lua как язык исключительно для данной игры.

Lua также полезен и как просто независимый язык, не только для обработки текста и одноразовых маленьких программ, но также и для различных проектов от среднего до большого размера. Для подобного использования основная функциональность Lua идет от ее библиотек. Стандартные библиотеки, например, предоставляют базовую функциональность по работе с шаблонами и другие функции для работы со строками. По мере того как Lua улучшает свою поддержку библиотек, появилось большое количество внешних пакетов. Lua Rocks, система для сборки и управления модулями для Lua, сейчас насчитывает более 150 пакетов.

Наконец, есть программисты, которые используют Lua как библиотеку для C. Такие люди больше пишут на C, чем на Lua, хотя им требуется хорошее понимание Lua для создания интерфейсов, которые являются простыми, легкими для использования и хорошо интегрированными с языком.

Эта книга может оказаться полезной всем этим людям. Первая часть покрывает сам язык, показывая, как можно использовать весь его потенциал. Мы фокусируемся на различных конструкциях языка и используем многочисленные примеры и упражнения, чтобы показать, как их использовать для практических задач. Некоторые главы этой части покрывают базовые понятия, такие как управляющие структуры, в то время как остальные главы покрывают более продвинутые темы, такие как итераторы и сопрограммы.

Вторая часть полностью посвящена таблицам, единственной структуре данных в Lua. Главы этой части обсуждают структуры данных, их сохранение (persistence), пакеты и объектно-ориентированное программирование. Именно там мы покажем всю силу языка.

Третья часть представляет стандартные библиотеки. Эта часть особенно полезна для тех, кто использует Lua как самостоятельный язык, хотя многие приложения включают частично или полностью стандартные библиотеки. В этой части каждой библиотеке посвящена отдельная глава: математической библиотеке, побитовой библиотеке, библиотеке по работе с таблицами, библиотеке по работе со строками, библиотеке ввода/вывода, библиотеке операционной системы и отладочной библиотеке.

Наконец, последняя часть книги покрывает API между Lua и C. Эта часть заметно отличается от всей остальной книги. В этой части мы будем программировать на C, а не на Lua. Для кого-то эта часть может оказаться неинтересной, а для кого-то – наоборот, самой полезной частью книги.

О третьем издании

Эта книга является обновленной и расширенной версией второго издания «Programming in Lua» (также известной как PiL 2). Хотя структура книги практически та же самая, это издание включает в себя полностью новый материал.

Во-первых, я обновил всю книгу на Lua 5.2. Глава об окружениях (environments) была практически полностью переписана. Я также переписал несколько примеров для того, чтобы показать преимущества

использования новых возможностей, предоставляемых Lua 5.2. Однако я четко обозначил отличия от Lua 5.1, поэтому вы можете использовать книгу для работы с этой версией языка.

Во-вторых, и более важно, я добавил упражнения во все главы книги. Сложность этих упражнений варьируется от простых вопросов до полноценных небольших проектов. Некоторые примеры иллюстрируют важные аспекты программирования на Lua и так же, как и примеры, расширят ваш набор полезных приемов.

Так же, как мы поступили с первым и вторым изданиями «Programming in Lua», мы сами опубликовали третье издание. Несмотря на маркетинговые ограничения, этот подход обладает рядом преимуществ: мы сохраняем полный контроль над содержимым книги; мы сохраняем все права для предложения книги в других формах; мы обладаем свободой для выбора, когда выпустить следующее издание; мы можем быть уверены, что книга всегда будет доступна.

Другие ресурсы

Краткое руководство необходимо всем, кто хочет освоить язык. Эта книга не заменяет краткое руководство. Напротив, они дополняют друг друга. Руководство только описывает Lua. Оно не показывает ни примеров, ни объяснений для конструкций языка. С другой стороны, оно полностью описывает язык: эта книга опускает некоторые, редко используемые, «темные углы» Lua. Более того, руководство описывает язык. Там, где эта книга расходится с руководством, доверяйте руководству. Чтобы получить руководство и дополнительную информацию по Lua, посетите веб-сайт <http://www.lua.org>.

Вы также можете найти полезную информацию на сайте пользователей Lua, поддерживаемом сообществом пользователей Lua, <http://lua-users.org>. Помимо других ресурсов, он предлагает также обучающий курс (tutorial), список сторонних пакетов и документации, архив официальной рассылки по Lua.

Эта книга описывает Lua 5.2, хотя большая часть содержимого также применима к Lua 5.1 и Lua 5.0. Некоторые отличия Lua 5.2 от предыдущих версий Lua 5 четко обозначены в книге. Если вы используете более свежую версию (выпущенную после этой книги), обратитесь к руководству по поводу отличий между версиями. Если вы используете версию старше, чем 5.2, то, может, пора подумать о переходе на более новую версию.

Некоторые типографские соглашения

В этой книге строки символов (“literal string”) заключены в двойные кавычки, а одиночные символы, например ‘a’, заключены в одиночные кавычки. Строки, которые являются шаблонами, также заключены в одиночные кавычки, например ‘[%w_]*’. В книге моноширинный шрифт используется для фрагментов кода и идентификаторов. Для **зарезервированных слов** используется жирный шрифт. Большие фрагменты кода показаны с применением следующего стиля:

```
-- program "Hello World"  
print("Hello World") --> Hello World
```

Обозначение --> показывает результат выполнения оператора или результат выражения:

```
print(10) --> 10  
13 + 3    --> 16
```

Поскольку двойной знак минус (--) начинает комментарий в Lua, ничего не будет если вы включите такой результат вывода (вместе с -->) в свою программу. Наконец, в книге используется обозначение <--> для обозначения того, что что-то эквивалентно чему-то другому:

```
this <--> that
```

Запуск примеров

Вам понадобится интерпретатор Lua для запуска примеров из этой книги. В идеале вам следует использовать Lua 5.2, однако большинство примеров без каких-либо изменений будет работать и на Lua 5.1.

Сайт Lua (<http://www.lua.org>) хранит весь исходный код для интерпретатора. Если у вас есть компилятор с C и знание того, как скомпилировать C код на вашем компьютере, то вам лучше попробовать поставить Lua из исходного кода; это действительно легко. Сайт *Lua Binaries* (поищите luabinaries) предлагает уже откомпилированные интерпретаторы для всех основных платформ. Если вы используете Linux или другую UNIX-подобную систему, вы можете проверить репозиторий вашего дистрибутива; многие дистрибутивы уже предлагают готовые пакеты с Lua. Для Windows хорошим выбором является *Lua for Windows* (поищите luaforwindows), являющийся

удобным набором для работы с Lua. Он включает в себя интерпретатор, интегрированный редактор и много библиотек.

Если вы используете Lua, встроенный в приложение, как WoW или Nmap, то вам может понадобиться руководство по данному приложению (или помощь «местного гуру»), для того чтобы разобраться, как запускать ваши программы. Тем не менее Lua остается все тем же языком; большинство примеров, которые мы рассмотрим в этой книге, применимы, несмотря на то, как вы используете Lua. Однако я рекомендую начать изучение Lua с интерпретатора для запуска ваших примеров.

Благодарности

Прошло уже почти десять лет с тех пор, как я опубликовал первое издание этой книги. Многие друзья и различные организации помогли мне в этом пути.

Как всегда, Луиг Хенрик де Фигуредо и Вальдемар Селес, соавторы Lua, предлагали все варианты помощи. Андре Карригал, Аско Кауппи, Бретт Капилик, Диего Нехаб, Эдвин Морагас, Фернандо Джефферсон, Гэвин Врес, Джон Д. Рамсделл и Норман Ремси предложили неоценимые замечания и полезные взгляды для различных изданий этой книги.

Луиза Новаэс, глава отдела искусства и дизайна в PUC-Rio, смогла найти время в своем занятом графике, чтобы создать идеальную обложку для данного издания.

Lightning Source, Inc. предложило надежный и эффективный вариант для печати и распространения данной книги. Без них самим издать эту книгу не получилось бы.

Центр латино-американских исследований в Стенфордском университете предоставил мне крайне необходимый перерыв от регулярной работы в очень стимулирующем окружении, во время которого я и сделал большую часть работы над третьим изданием.

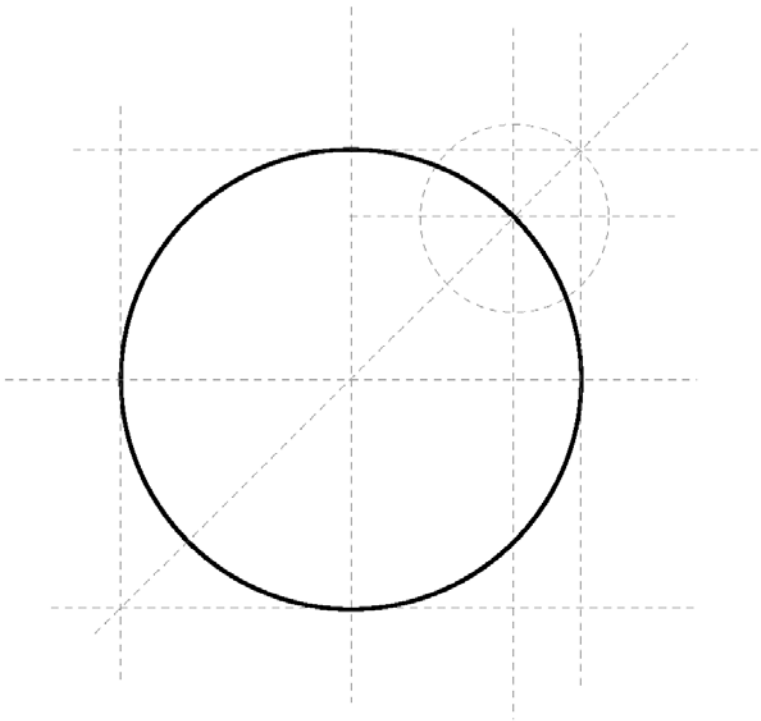
Я также хотел бы поблагодарить Pontifical Catholic University Рио де Жанейро (PUC-Rio) и Бразильский национальный исследовательский совет (CNPq) за их продолжающуюся поддержку моей работы.

Наконец, я должен выразить мою глубокую благодарность Ноэми Родригес за все виды помощи (технической, и не только) и за освещение моей жизни.



Часть I

Язык





ГЛАВА 1

Начинаем

Продолжая традицию, наша первая программа на Lua просто печатает "Hello World":

```
print("Hello World")
```

Если вы используете отдельный интерпретатор Lua, то все, что вам надо для запуска вашей первой программы, – это запустить интерпретатор – обычно он называется `lua` или `lua5.2` – с именем текстового файла, содержащего вашу программу. Если вы сохранили приведенную выше программу в файле `hello.lua`, то вам следует выполнить следующую команду:

```
% lua hello.lua
```

Как более сложный пример наша следующая программа определяет функцию для вычисления факториала заданного числа, спрашивает у пользователя число и печатает его факториал:

```
-- defines a factorial function
function fact (n)
  if n == 0 then
    return 1
  else
    return n * fact(n-1)
  end
end

print("enter a number:")
a = io.read("*n") -- reads a number
print(fact(a))
```

1.1. Блоки

Каждый кусок кода, который Lua выполняет, такой как файл или отдельная строка в интерактивном режиме, называется *блоком* (chunk). Блок – это просто последовательность команд (или операторов).

Lua не нужен разделитель между подряд идущими операторами, но вы можете использовать точку с запятой, если хотите. Я лично использую точку с запятой только для разделения операторов, записанных в одной строке. Разбиение на строки не играет никакой роли в синтаксисе Lua; так, следующие четыре блока допустимы и эквивалентны:

```
a = 1
b = a*2
a = 1;
b = a*2;
a = 1; b = a*2
a = 1 b = a*2 -- ugly, but valid
```

Блок может состоять всего из одного оператора, как в примере «Hello World», или состоять из набора операторов и определений функций (которые на самом деле являются просто присваиваниями, как мы увидим позже), как в примере с факториалом. Блок может быть так велик, как вы хотите. Поскольку Lua используется также как язык для описания данных, блоки в несколько мегабайт не являются редкостью. Интерпретатор Lua не имеет каких-либо проблем при работе с большими блоками.

Вместо того чтобы записывать ваши программы в файл, вы можете запустить интерпретатор в интерактивном режиме. Если вы запустите lua без аргументов, то вы увидите его приглашения для ввода:

```
% lua
Lua 5.2 Copyright (C) 1994-2012 Lua.org, PUC-Rio
>
```

Соответственно, каждая команда, которую вы вводите (как, например, `print "Hello World"`), выполняется немедленно, после того как вы ее введете. Для выхода из интерпретатора, просто наберите символ конца файла (`ctrl-D` в UNI, `ctrl-Z` в Windows) или позовите функцию `exit` из библиотеки операционной системы – вам нужно набрать `os.exit()`.

В интерактивном режиме Lua обычно интерпретирует каждую строку, которую вы вводите, как отдельный блок. Однако если он обнаруживает, что строка не является законченным блоком, то он ждет продолжения ввода до тех пор, пока не получится законченный блок. Таким образом вы можете вводить многострочные определения, такие как функция `factorial`, прямо в интерактивном режиме. Однако обычно более удобным является помещать подобные определения в файл и затем вызывать Lua для выполнения этого файла.

Вы можете использовать опцию `-i` для того, чтобы заставить Lua перейти в интерактивный режим после выполнения заданного блока:

```
% lua -i prog
```

Команда вроде этой выполнит блок в файле `prog` и затем перейдет в интерактивный режим. Это особенно полезно для отладки и ручного тестирования. В конце данной главы мы рассмотрим другие опции командной строки для интерпретатора Lua.

Другим способом запускать блоки является функция `dofile`, которая немедленно выполняет файл. Например, допустим, что у вас есть файл `lib1.lua` со следующим кодом:

```
function norm (x, y)
  return (x^2 + y^2)^0.5
end
```

```
function twice (x)
  return 2*x
end
```

Тогда в интерактивном режиме вы можете набрать

```
> dofile("lib1.lua") -- load your library
> n = norm(3.4, 1.0)
> print(twice(n)) --> 7.0880180586677
```

Функция `dofile` также полезна, когда вы тестируете фрагмент кода. Вы можете работать с двумя окнами: в одном находится текстовый редактор с вашей программой (например, в файле `prog.lua`), и в другом находится консоль с запущенным интерпретатором Lua в интерактивном режиме. После того как вы сохранили изменения в вашей программе, вы выполняете `dofile("prog.lua")` в консоли для загрузки нового кода; затем вы можете начать использование нового кода, вызывая функции и печатая результаты.

1.2. Некоторые лексические соглашения

Идентификаторы (или имена) в Lua являются строками из латинских букв, цифр и знака подчеркивания, не начинающимися с цифры; например:

```
i      j      i10      _ij
aSomewhatLongName      _INPUT
```

Вам лучше избегать идентификаторов, состоящих из подчеркивания, за которым следуют заглавные латинские буквы (например, `_VERSION`); они зарезервированы для специальных целей в Lua. Обычно я использую идентификатор `_` (одиночное подчеркивание) для пустых (dummy) переменных.

В старых версиях Lua понятие того, что является буквой, зависело от локали. Однако подобные буквы делают вашу программу неподходящей для запуска на системах, которые не поддерживают данную локаль. Поэтому Lua 5.2 рассматривает в качестве букв только буквы из следующих диапазонов: A-Z и a-z.

Следующие слова зарезервированы, вы не можете использовать их в качестве идентификаторов:

```
and      break      do      else      elseif
end      false      goto      for      function
if      in      local      nil      not
or      repeat      return      then      true
until      while
```

Lua учитывает регистр букв: **and** – это зарезервированное слово, однако `And` и `AND` – это два разных идентификатора.

Комментарий начинается с двух знаков минуса (`--`) и продолжается до конца строки. Lua также поддерживает блочный комментарий, который начинается с `--[[` и идет до следующего `]]`¹. Стандартный способ закомментировать фрагмент кода – это поместить его между `--[[` и `--]]`, как показано ниже:

```
--[[
print(10) -- no action (commented out)
--]]
```

Для того чтобы снова сделать этот код активным, просто добавьте один минус к первой строке:

```
---[[
print(10) --> 10
--]]
```

В первом примере `--[[` в первой строке начинает блочный комментарий, и двойной минус в последней строке также находится в этом комментарии. Во втором примере `---` [начинает обычный

¹ Блочные комментарии могут быть более сложными, как мы увидим в разделе 2.4.

Конец ознакомительного фрагмента.
Приобрести книгу можно
в интернет-магазине
«Электронный универс»
e-Univers.ru