Содержание

От издательства 11

Предисловие 12

Глава 1. Почему вы должны учиться писать программы 14
1.1. Креативность и мотивация 15
1.2. Аппаратная архитектура компьютера 16
1.3. Изучение программирования 18
1.4. Слова и предложения 18
1.5. Диалог с Python 20
1.6. Терминология: интерпретатор и компилятор 22
1.7. Написание программы 24
1.8. Что такое программа 24
1.9. Структурные элементы программы 26
1.10. Что могло бы пойти не так 27
1.11. Отладка 29
1.12. Процесс обучения 30
1.13. Словарь терминов 31
1.14. Упражнения 32

Глава 2. Переменные, выражения и инструкции 34
2.1. Значения и типы 34
2.2. Переменные 35
2.3. Имена переменных и ключевые слова 36
2.4. Инструкции 37
2.5. Операторы и операнды 37
2.6. Выражения 38
2.7. Порядок выполнения операций 39
2.8. Оператор деления по модулю 39
2.9. Операции со строками 40
2.10. Запрос ввода от пользователя 40
2.11. Комментарии 41
2.12. Выбор легко запоминаемых имен переменных 42
2.13. Отладка 44
2.14. Словарь терминов 45
2.15. Упражнения 46

Глава 3. Условное выполнение 48
3.1. Логические выражения 48
3.2. Логические операторы 49

3.3. Условное выполнение 49
3.4. Альтернативная последовательность выполнения 51
3.5. Цепочечные условные инструкции 51
3.6. Вложенные условные инструкции 52
3.7. Перехват исключений с использованием ключевых слов `try` и `except` 53
3.8. Вычисление логических выражений по сокращенной схеме 55
3.9. Отладка 57
3.10. Словарь терминов 57
3.11. Упражнения 58

Глава 4. Функции 60

4.1. Вызовы функций 60
4.2. Встроенные функции 60
4.3. Функции преобразования типов 61
4.4. Математические функции 62
4.5. Случайные числа 63
4.6. Добавление новых функций 64
4.7. Определение и использование 66
4.8. Поток выполнения 66
4.9. Параметры и аргументы 67
4.10. Продуктивные и пустые функции 68
4.11. Зачем нужны функции 69
4.12. Отладка 70
4.13. Словарь терминов 70
4.14. Упражнения 72

Глава 5. Итерации 74

5.1. Обновление переменных 74
5.2. Инструкция `while` 74
5.3. Бесконечные циклы 75
5.4. Завершение отдельных итераций с помощью инструкции `continue` 77
5.5. Определение циклов с использованием инструкции `for` 78
5.6. Шаблоны цикла 79
5.6.1. Циклы подсчета и суммирования 79
5.6.2. Циклы вычисления максимума и минимума 80
5.7. Отладка 81
5.8. Словарь терминов 82
5.9. Упражнения 83

Глава 6. Строки 84

6.1. Строка – это последовательность 84
6.2. Получение длины строки с помощью функции `len` 85
6.3. Проход по строке с использованием цикла 85
6.4. Вырезки строк 86
6.5. Строки неизменяемы 87

6.6. Работа в цикле и подсчет ... 87
6.7. Оператор `in` ... 88
6.8. Сравнение строк ... 88
6.9. Методы строк ... 89
6.10. Синтаксический разбор (парсинг) строк ... 91
6.11. Оператор формата ... 92
6.12. Отладка ... 93
6.13. Словарь терминов ... 94
6.14. Упражнения ... 95

Глава 7. Файлы ... 96

7.1. Длительное хранение данных ... 96
7.2. Открытие файлов ... 97
7.3. Текстовые файлы и строки в них ... 98
7.4. Чтение файлов ... 99
7.5. Поиск в файле ... 100
7.6. Предоставление пользователю выбора имени файла ... 103
7.7. Использование `try`, `except` и `open` ... 104
7.8. Запись в файлы ... 105
7.9. Отладка ... 106
7.10. Словарь терминов ... 107
7.11. Упражнения ... 107

Глава 8. Списки ... 109

8.1. Список – это последовательность ... 109
8.2. Списки – изменяемые объекты ... 109
8.3. Проход по списку ... 110
8.4. Операции со списками ... 111
8.5. Вырезка из списка ... 111
8.6. Методы списков ... 112
8.7. Удаление элементов ... 113
8.8. Списки и функции ... 113
8.9. Списки и строки ... 115
8.10. Синтаксический анализ (парсинг) строк ... 116
8.11. Объекты и значения ... 116
8.12. Псевдонимы ... 117
8.13. Списки как аргументы ... 118
8.14. Отладка ... 119
8.15. Словарь терминов ... 124
8.16. Упражнения ... 124

Глава 9. Словари ... 127

9.1. Словарь как множество счетчиков ... 129
9.2. Словари и файлы ... 130
9.3. Циклы и словари ... 132

9.4. Расширенный синтаксический анализ текста ... 133
9.5. Отладка ... 135
9.6. Словарь терминов ... 135
9.7. Упражнения ... 136

Глава 10. Кортежи ... 138

10.1. Кортежи неизменяемы ... 138
10.2. Сравнение кортежей ... 139
10.3. Присваивание кортежам ... 141
10.4. Словари и кортежи ... 142
10.5. Множественное присваивание с помощью словарей ... 143
10.6. Наиболее часто встречающиеся слова ... 144
10.7. Использование кортежей как ключей в словарях ... 145
10.8. Последовательности: строки, списки и кортежи – ну и ну! ... 146
10.9. Отладка ... 146
10.10. Словарь терминов ... 147
10.11. Упражнения ... 147

Глава 11. Регулярные выражения ... 149

11.1. Символы определения совпадений в регулярных выражениях ... 150
11.2. Извлечение данных с использованием регулярных выражений ... 152
11.3. Объединение поиска и извлечения ... 154
11.4. Специальный символ экранирования (escape) ... 158
11.5. Итоговый обзор специальных символов ... 159
11.6. Дополнительный раздел для пользователей систем Unix/Linux ... 160
11.7. Отладка ... 161
11.8. Словарь терминов ... 161
11.9. Упражнения ... 162

Глава 12. Сетевые программы ... 163

12.1. Протокол HTTP – Hypertext Transfer Protocol ... 163
12.2. Самый простой в мире веб-браузер ... 164
12.3. Извлечение изображения с использованием протокола HTTP ... 166
12.4. Извлечение веб-страниц с помощью библиотеки `urllib` ... 169
12.5. Чтение двоичных файлов с использованием библиотеки `urllib` ... 170
12.6. Синтаксический анализ формата HTML и веб-скрейпинг ... 171
12.7. Синтаксический анализ формата HTML с использованием регулярных выражений ... 171
12.8. Синтаксический анализ формата HTML с использованием BeautifulSoup ... 173
12.9. Дополнительный раздел для пользователей систем Unix/Linux ... 176
12.10. Словарь терминов ... 177
12.11. Упражнения ... 177

Глава 13. Использование веб-сервисов ... 179

13.1. XML – eXtensible Markup Language ... 179
13.2. Синтаксический анализ XML ... 180

13.3. Проход в цикле по узлам ... 181
13.4. JSON – JavaScript Object Notation ... 182
13.5. Синтаксический анализ формата JSON ... 183
13.6. Программные интерфейсы приложений ... 185
13.7. Безопасность и использование API ... 186
13.8. Словарь терминов ... 187
13.9. Приложение 1: веб-сервис геокодирования Google ... 187
13.10. Приложение 2: Twitter ... 191

Глава 14. Объектно-ориентированное программирование ... 196

14.1. Управление более крупными программами ... 196
14.2. Приступим ... 197
14.3. Использование объектов ... 197
14.4. Начнем с программ ... 198
14.5. Разделение задачи на подзадачи ... 200
14.6. Наш первый объект Python ... 201
14.7. Классы как типы ... 204
14.8. Жизненный цикл объекта ... 204
14.9. Несколько экземпляров ... 206
14.10. Наследование ... 207
14.11. Резюме ... 208
14.12. Словарь терминов ... 209

Глава 15. Использование баз данных и SQL ... 210

15.1. Что такое база данных ... 210
15.2. Концепции базы данных ... 211
15.3. Браузер базы данных для SQLite ... 211
15.4. Создание таблицы базы данных ... 212
15.5. Обзор языка структурированных запросов SQL ... 215
15.6. Реализация глобального поиска в Twitter с использованием базы данных ... 216
15.7. Основы моделирования данных ... 222
15.8. Программирование с использованием нескольких таблиц ... 225
15.8.1. Ограничивающие условия в таблицах базы данных ... 227
15.8.2. Извлечение и/или вставка записи ... 228
15.8.3. Сохранение отношения следования за другом ... 229
15.9. Три типа ключей ... 230
15.10. Использование `JOIN` для извлечения данных ... 231
15.11. Резюме ... 233
15.12. Отладка ... 234
15.13. Словарь терминов ... 234

Глава 16. Визуализация данных ... 236

16.1. Создание карты OpenStreetMap по данным геокодирования ... 236
16.2. Визуализация сетей и сетевых соединений ... 239
16.3. Визуализация данных электронной почты ... 242

Приложение А. Участники проекта 248
А.1. Список участников проекта «Python for Everybody» («Python для всех») 248
А.2. Список участников проекта «Python for Informatics» 248
А.3. Предисловие к книге «Think Python» 249
А.3.1. Странная история книги «Think Python» 249
А.3.2. Благодарности за работу над «Think Python» 250
А.4. Список участников проекта «Think Python» 251

Приложение В. Подробная информация о защите авторского права 253

Предметный указатель 255

От издательства

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв на нашем сайте www.dmkpress.com, зайдя на страницу книги и оставив комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com; при этом укажите название книги в теме письма.

Если вы являетесь экспертом в какой-либо области и заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу dmkpress@gmail.com.

Скачивание исходного кода примеров

Скачать файлы с дополнительной информацией для книг издательства «ДМК Пресс» можно на сайте www.dmkpress.com на странице с описанием соответствующей книги.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы обеспечить высокое качество наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг, мы будем очень благодарны, если вы сообщите о ней главному редактору по адресу dmkpress@gmail.com. Сделав это, вы избавите других читателей от недопонимания и поможете нам улучшить последующие издания этой книги.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательство «ДМК Пресс» очень серьезно относится к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконной публикацией какой-либо из наших книг, пожалуйста, пришлите нам ссылку на интернет-ресурс, чтобы мы могли применить санкции.

Ссылку на подозрительные материалы можно прислать по адресу электронной почты dmkpress@gmail.com.

Мы высоко ценим любую помощь по защите наших авторов, благодаря которой мы можем предоставлять вам качественные материалы.

Предисловие

Другая редакция книги под свободной лицензией

Для преподавателей и научных сотрудников вузов, которые постоянно твердят «публикуйся или исчезни», вполне естественным является желание создавать с нуля собственные оригинальные творения. Эта книга – эксперимент, но не начатый с нуля, а представляющий собой «другую редакцию» книги «Think Python: How to Think Like a Computer Scientist», написанной Алленом Б. Дауни (Allen B. Downey), Джеффом Элкнером (Jeff Elkner) и другими авторами.

В декабре 2009 г. я готовился к преподаванию курса SI502 – Networked Programming в Мичиганском университете в пятом семестре подряд и решил, что наступило время написать учебник по языку Python, в котором главное внимание было бы сосредоточено на обработке данных, а не на понимании алгоритмов и абстракций. Моя цель в курсе SI502 – научить людей навыкам обработки данных с использованием Python, которые оставались бы полезными в течение всей жизни. Лишь некоторые из моих студентов планировали стать профессиональными программистами. Но другие предполагали работать библиотекарями, менеджерами, юристами, биологами, экономистами и т. д., которые способны умело использовать (информационные) технологии в выбранной ими области.

Мне никогда не удавалось найти совершенную книгу, ориентированную на обработку данных на языке Python, для своего курса, поэтому я просто решил написать такую книгу. К счастью, за три недели до того, как я начал работу над своей новой книгой с нуля, на собрании преподавателей факультета доктор Атул Пракаш (Atul Prakash) показал мне книгу «Think Python», которую он использовал при чтении своего курса по языку Python в этом семестре. Оказалось, что это отлично написанный текстовый материал по информационным технологиям, уделяющий главное внимание коротким четким описаниям и легкий для изучения.

Общая структура этой книги была изменена для перехода к решению задач анализа данных как можно быстрее, и в нее был включен ряд работающих примеров и упражнений по анализу данных для самых начинающих.

Главы 2–10 похожи на книгу «Think Python», но и в них были внесены существенные изменения. Примеры и упражнения на вычисления были заменены на упражнения, ориентированные на обработку данных. Темы представлены в порядке, необходимом для последовательного формирования решений по анализу данных с постепенно увеличивающейся сложностью. Некоторые темы, такие как применение `try` и `except`, вынесены в более ранний материал и представлены как часть главы об условных конструкциях. Функциям уделено весьма немного внимания до тех пор, пока они не становятся необходимыми для снижения сложности программы, т. е. они не описываются как ранний пример абстракции. Почти все функции, определяемые пользователем, удалены из исходных кодов примеров и упражнений,

встречающихся до главы 4. Слово «рекурсия» вообще не появляется в этой книге (разумеется, за исключением этой строки).

В главах 1 и 11–16 весь материал принципиально новый, сфокусированный на использовании в реальной жизни методов анализа данных, приведены соответствующие примеры на Python, включая регулярные выражения для поиска и синтаксического разбора (парсинга), автоматизации задач на вашем компьютере, извлечения данных из сетевой среды, скрейпинг веб-страниц для получения данных, объектно-ориентированного программирования, использования веб-сервисов, синтаксического разбора (парсинга) данных в форматах XML и JSON, создания и использования баз данных с применением языка Structured Query Language (SQL) и визуализации данных.

Самая главная цель этих изменений – смещение фокуса с компьютерных технологий на информатику и включение в первый курс по компьютерным технологиям только тех тем, которые могут быть полезными, даже если студент не намерен становиться профессиональным программистом.

Студенты, которые найдут эту книгу интересной и захотят узнать больше, должны обратиться к книге «Think Python» Аллена Б. Дауни. Поскольку между этими двумя книгами очень много общего, студенты быстро овладеют навыками и умениями в дополнительных областях технического программирования и алгоритмического мышления, которые подробно рассматриваются в книге «Think Python». А с учетом того, что книги написаны в похожем стиле, изучение «Think Python» потребует лишь минимальных усилий.

Как обладатель прав на «Think Python» Аллен разрешил мне изменить лицензию, касающуюся материала из его книги, который включен в мою книгу, с GNU Free Documentation License на более новую лицензию Creative Commons Attribution – Share Alike. Это является следствием всеобщего перехода на лицензии, защищающие свободно распространяемую (открытую) документацию, – от лицензии GFDL к лицензии CC-BY-SA (например, «Википедия»). Использование лицензии CC-BY-SA сохраняет строгие традиции книжного «копилефта», но при этом лицензия становится еще более понятной для новых авторов при повторном использовании этого материала, если они находят его подходящим для своих работ.

Я считаю, что эта книга послужит примером, объясняющим, почему свободно распространяемые материалы так важны для будущего образования, и хочу поблагодарить Аллена Б. Дауни и издательство Cambridge University Press за их дальновидное решение сделать эту книгу доступной под открытым авторским правом. Надеюсь, что они довольны результатами моих усилий, а также надеюсь, что вам, моим читателям, понравится наш совместный труд.

Я хотел бы поблагодарить Аллена Б. Дауни (Allen B. Downey) и Лорен Каулз (Lauren Cowles) за их помощь, терпение и руководство в совместной работе и решении вопросов об авторских правах, касающихся этой книги.

Чарльз Северанс (Charles Severance)
www.dr-chuck.com
Анн-Арбор (шт. Мичиган), США
9 сентября 2013 г.

Чарльз Северанс является практикующим адъюнкт-профессором в Информационной школе Мичиганского университета.

Глава 1

Почему вы должны учиться писать программы

Написание программ (или программирование) – это весьма креативная и плодотворная деятельность. Вы можете писать программы по многим причинам, посвятив всю свою жизнь решению трудных задач анализа данных, или получать удовольствие, помогая кому-нибудь другому решить трудную задачу. Эта книга предполагает, что каждый должен знать, как написать программу, а как только вы узнаете, как написать программу, то определите, что именно вы хотите сделать с помощью новоприобретенных умений и навыков.

В повседневной жизни мы окружены компьютерами со всех сторон – от ноутбуков до мобильных телефонов. Мы можем считать эти компьютеры своими «личными помощниками», которые могут позаботиться о многих вещах вместо нас. В сущности, аппаратура в современных компьютерах создана так, чтобы непрерывно спрашивать нас: «Что я должен делать дальше?»

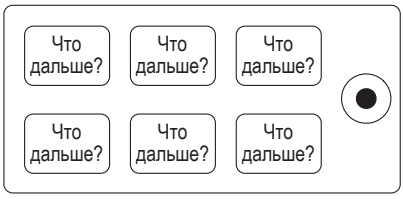

Рис. 1.1 ❖ Личный цифровой помощник

Программисты добавляют операционную систему и набор приложений к аппаратуре, и в итоге мы получаем личный цифровой помощник, который весьма полезен и способен помочь нам выполнить множество разнообразных дел.

Наши компьютеры быстры, обладают огромным объемом памяти и могли бы оказаться чрезвычайно полезными, если бы мы только знали язык, на котором могли бы объяснить компьютеру, что именно он «должен делать дальше». Если бы мы знали такой язык, то могли бы приказать компьютеру выполнять для нас многократно повторяющиеся (рутинные) задачи. Любопытно, что компьютеры лучше всего выполняют те задачи, которые нам, людям, кажутся скучными и утомительно-нудными.

Например, посмотрите на первые три абзаца этой главы и скажите, какое слово чаще всего используется в них и сколько раз это слово там встречается. Несмотря на то что вы способны читать и понимать слова за несколько секунд, их подсчет становится почти мучительной проблемой, потому что это не тот тип задач, для решения которых предназначен человеческий мозг. Для компьютера совсем наоборот – чтение и понимание текста с листа бумаги являются трудной задачей, но подсчет слов и вывод сообщения для нас – сколько раз самое часто встречающееся слово было использовано в заданном тексте – это очень простая задача:

```
python words.py
Enter file:words.txt
to 16
```

Наш «личный помощник по анализу информации» быстро сообщает, что слово «to» использовалось шестнадцать раз в первых трех абзацах этой главы.

Тот простой факт, что компьютеры лучше справляются с теми делами, в которых люди не преуспевают, является причиной, почему вам необходимо приобрести навыки общения на «языке компьютера». Как только вы изучите этот новый язык, то сразу сможете передать рутинные задачи своему партнеру (компьютеру), освобождая больше времени для тех дел, для которых вы лучше приспособлены. В такое партнерство вы привносите креативность, интуицию и изобретательность.

1.1. Креативность и мотивация

Хотя эта книга и не предназначена для профессиональных программистов, профессиональное программирование может быть чрезвычайно плодотворным делом как в финансовом, так и в личном плане. Создание полезных, изящных и интеллектуальных программ для использования другими людьми – это весьма креативная деятельность. Ваш компьютер или личный цифровой помощник (PDA – Personal Digital Assistant) обычно содержит множество разнообразных программ многих различных групп программистов, конкурирующих в борьбе за ваше внимание и заинтересованность. Они пытаются наилучшим образом удовлетворить ваши потребности и предоставить наибольшее удобство как пользователю в этом процессе. В некоторых ситуациях, когда вы выбираете часть программного обеспечения, программисты напрямую вознаграждаются благодаря вашему выбору.

Если считать программы креативным результатом работы группы программистов, то, возможно, на рис. 1.2 представлена более рациональная версия нашего личного цифрового помощника.

В данный момент наша главная мотивация – не зарабатывание денег и не удовлетворение потребностей конечных пользователей, а повышение эффективности при обработке данных и информации, которая встречается в повседневной жизни. В самом начале вы будете и программистом, и конечным пользователем своих программ. Повышая свое мастерство программис-

та, вы почувствуете, что программирование становится более креативным занятием, и тогда ваши мысли обратятся в сторону разработки программ для других людей.

Рис. 1.2 ❖ Программисты разговаривают с вами

1.2. Аппаратная архитектура компьютера

Прежде чем начать изучение языка, на котором передаются инструкции компьютерам для разработки программного обеспечения, необходимо немного познакомиться с тем, как устроены компьютеры. Если бы вы разобрали компьютер или мобильный телефон и заглянули поглубже внутрь, то увидели бы его составные части, показанные на рис. 1.3.

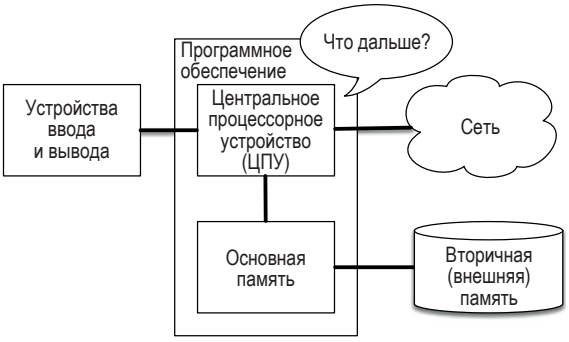

Рис. 1.3 ❖ Аппаратная архитектура

Ниже приведены определения высокого уровня этих составных частей:

- центральное процессорное устройство (ЦПУ), или просто центральный процессор (ЦП) (Central Processing Unit – CPU) – это та часть компьютера, которая одержима вопросом «что дальше?». Если ваш компьютер определен как имеющий тактовую частоту 3.0 ГГц, это значит, что ЦПУ будет спрашивать «что дальше?» три миллиарда раз в секунду. Вы должны научиться говорить быстро, чтобы успевать за ЦПУ;
- основная память (main memory) используется для хранения информации, которая необходима ЦПУ в срочном порядке. Основная память почти так же быстра, как ЦПУ. Но информация, хранящаяся в основной памяти, бесследно исчезает при выключении компьютера;

- вторичная (внешняя) память (secondary memory) также используется для хранения информации, но она намного медленнее основной памяти. Преимущество вторичной памяти состоит в том, что она может хранить информацию, даже когда компьютер выключен. Примерами вторичной (внешней) памяти являются дисковые накопители и устройства флеш-памяти (обычно в составе USB-накопителей и мобильных музыкальных плееров);
- устройства ввода и вывода – это просто экран дисплея, клавиатура, мышь, микрофон, динамик, тачпад и т. п. Все это – способы взаимодействия с компьютером;
- в наше время большинство компьютеров также имеют устройство сетевого соединения (network connection) для получения информации по сети. Можно считать сеть очень медленным «устройством» для хранения и извлечения данных, которое, возможно, не всегда «готово к работе». Так что в некотором смысле сеть – это более медленная и временами ненадежная форма вторичной (внешней) памяти.

Большинство подробностей о том, как работают эти компоненты, лучше оставить производителям компьютеров, но все же полезно знать, что означают некоторые термины, чтобы мы могли говорить о некоторых таких составных частях компьютера при написании программ.

Ваша работа как программиста состоит в использовании и регулировании каждого из этих ресурсов для решения поставленной перед вами задачи и анализа данных, получаемых из этого решения. Как программист вы в основном будете «разговаривать» с ЦПУ и сообщать ему, что делать дальше. Иногда вы будете приказывать ЦПУ использовать основную память, вторичную (внешнюю) память, сеть или устройства ввода/вывода.

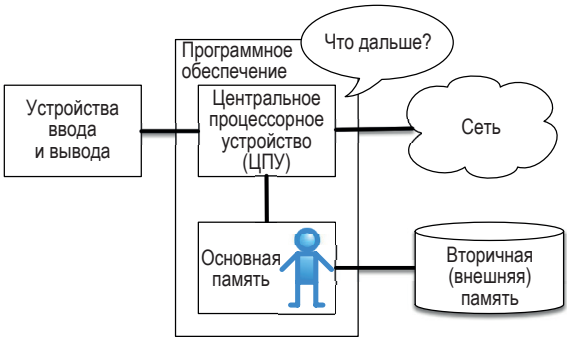

Рис. 1.4 ❖ Где находитесь вы

Вы должны быть тем человеком, который отвечает на вопрос ЦПУ «что дальше?». Но уменьшение размеров вашего тела до 5 мм с последующим перемещением внутрь компьютера, чтобы вы смогли подавать три миллиарда команд в секунду, может оказаться чрезвычайно неудобным. Так что вместо этого вы должны написать свои инструкции заранее. Мы называем эти сохраненные инструкции программой, а запись инструкций и обеспечение их корректности – программированием.

1.3. Изучение программирования

В остальной части книги мы попытаемся превратить вас в такого человека, который хорошо владеет искусством программирования. После прочтения книги вы станете программистом, возможно, непрофессиональным, но, по крайней мере, вы получите навыки для определения задачи анализа данных/информации и разработки программы для решения этой задачи.

В той или иной степени вам необходимы два навыка (умения), чтобы стать программистом:

- во-первых, вы должны знать язык программирования (Python) – требуются знания словаря и грамматики. Вы обязаны правильно произносить (т. е. использовать) слова этого нового языка и знать, как составлять правильно сформулированные «предложения» на этом новом языке;
- во-вторых, необходимо умение «рассказывать истории». Записывая рассказ, вы объединяете слова и предложения, чтобы донести главную мысль до читателя. Существует умение и искусство составления рассказа, а умение написать рассказ улучшается постоянной практикой написания текстов и получением некоторой обратной связи. В программировании наша программа – это «рассказ», а задача, которую вы пытаетесь решить, – это «основная мысль».

После того как вы освоите один из языков программирования, например Python, вы обнаружите, что гораздо проще изучить второй язык, такой как JavaScript или C++. Новый язык программирования имеет совершенно другой словарь и грамматику, но умение решать задачи останется одним и тем же при использовании всех языков программирования.

Вы изучите «словарь» и «предложения» языка Python достаточно быстро. Больше времени займет приобретение способности написать логически связную программу для решения качественно новой задачи. Мы изучаем программирование во многом так же, как учимся писать. Начинаем с чтения и объяснения программ, затем пишем простые программы, после этого пишем программы, сложность которых постепенно увеличивается со временем. В некоторый момент вы чувствуете, что «нашли свою музу», знаете, как создать собственные шаблоны, и более естественным для вас становится понимание, как сформулировать задачу и написать программу для ее решения. Когда вы достигаете этого момента, программирование становится весьма приятным и креативным процессом.

Мы начинаем со словаря и структуры программы на языке Python. Будьте терпеливы, так как простые примеры напомнят вам то время, когда вы начинали учиться читать.

1.4. Слова и предложения

В отличие от человеческих языков, словарь Python действительно невелик. Этот словарь мы называем «зарезервированными словами» (reserved words). Эти слова обладают особым значением в языке Python. Когда Python видит

эти слова в программе, для него они имеют один и только один смысл. Позже, когда вы начнете писать программы, то будете создавать собственные слова, имеющие особый смысл лично для вас, – они называются переменными (variables). Вы получите большую свободу выбора имен для своих переменных, но в качестве таких имен не сможете использовать ни одно из зарезервированных слов языка Python.

Когда вы дрессируете собаку, то используете специальные слова, например «сидеть», «тубо» и «апорт». Когда вы говорите с собакой, но не используете какое-либо из этих зарезервированных слов, она просто вопросительно смотрит на ваше лицо до тех пор, пока вы не произнесете зарезервированное слово. Например, если вы говорите: «Хотел бы я, чтобы больше людей могли гулять для улучшения своего здоровья», – то большинство собак слышат: «бла, бла, бла, гулять, бла, бла, бла», потому что «гулять» – это зарезервированное слово на собачьем языке. Возможно, многие предполагают, что в языке общения между людьми и кошками нет таких зарезервированных слов[1].

Зарезервированные слова в языке, на котором люди разговаривают с Python, перечислены в табл. 1.1.

Таблица 1.1. Зарезервированные слова языка Python

`and`	`del`	`global`	`not`	`with`
`as`	`elif`	`if`	`or`	`yield`
`assert`	`else`	`import`	`pass`	
`break`	`except`	`in`	`raise`	
`class`	`finally`	`is`	`return`	
`continue`	`for`	`lambda`	`try`	
`def`	`from`	`nonlocal`	`while`	

Вот и все слова, но, в отличие от собаки, Python уже полностью выдрессирован. Если вы говорите «try», то Python будет пытаться (выполнить определенное действие) каждый раз, когда вы произносите это слово без ошибки.

Мы выучим все эти зарезервированные слова и научимся правильно их использовать в нужное время, но сейчас необходимо сосредоточиться на аналоге «разговора» с Python (в человеческо-собачьем языке). В диалоге с Python хорошо то, что мы можем даже сообщить, что именно нужно сказать, передавая ему сообщение в (одиночных) кавычках:

```
print('Hello world!')
```

Мы только что написали свое первое синтаксически правильное предложение на языке Python. Это предложение начинается с функции `print`, за которой следует выбранная нами строка текста, заключенная в одиночные кавычки. Строки в инструкциях вывода всегда заключаются в кавычки. Одиночные и двойные кавычки выполняют одну и ту же функцию. Большинство людей используют одиночные кавычки, за исключением тех случаев, когда одиночная кавычка (которая также является символом апострофа) содержится в строке.

---

[1] http://xkcd.com/231/.

1.5. Диалог с Python

Теперь, когда нам известно слово и простое предложение в Python, необходимо узнать, как начать диалог с Python, чтобы проверить наши новые познания в языке.

Прежде чем начать диалог с Python, сначала необходимо установить соответствующее программное обеспечение на ваш компьютер и научиться запускать Python. Но это требует описания слишком многих подробностей в данной главе, поэтому я предлагаю обратиться на сайт www.py4e.com, где приведены подробные инструкции и деморолики по установке и запуску Python в системах Macintosh и Windows. В некоторый момент вы окажетесь в окне терминала или командной строки, где нужно будет ввести команду `python`, чтобы запустить интерпретатор языка Python, который выполняется в интерактивном режиме и выглядит приблизительно так:

```
Python 3.5.1 (v3.5.1:37a07cee5969, Dec  6 2015, 01:54:25)
[MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Промпт (приглашение) >>> – это способ интерпретатора Python спросить вас «Что прикажете делать дальше?». Python готов начать диалог с вами. Все, что вам нужно знать, – как разговаривать на языке Python.

Например, предположим, что вы пока еще не знали даже самых простых слов и предложений языка Python. Вероятно, вы воспользуетесь стандартной фразой, которую применяют астронавты при посадке на далекой планете и попытке поговорить с ее обитателями:

```
>>> I come in peace, please take me to your leader
#   Я пришел с миром, пожалуйста, отведите меня к вашему предводителю.
File "<stdin>", line 1
I come in peace, please take me to your leader
     ^
SyntaxError: invalid syntax
>>>
```

Получилось не очень удачно. Если вы быстро не придумаете что-то еще, то обитатели этой планеты, вероятнее всего, проткнут вас копьями, насадят на вертел, поджарят и съедят на обед.

К счастью, в свои путешествия вы прихватили с собой экземпляр этой книги, открыли ее как раз на этой странице и попытались снова:

```
>>> print('Hello world!')
Hello world!
```

Это выглядит намного лучше, и вы пробуете продолжить общение:

```
>>> print('You must be the legendary god that comes from the sky')
You must be the legendary god that comes from the sky
# Должно быть, вы бог из легенд, пришедший с неба
```

```
>>> print('We have been waiting for you for a long time')
We have been waiting for you for a long time
# Мы очень долго ждали вас
>>> print('Our legend says you will be very tasty with mustard')
Our legend says you will be very tasty with mustard
# Наши легенды гласят, что вы будете очень вкусны с горчицей
>>> print 'We will have a feast tonight unless you say
# Мы будем пировать нынешней ночью, если вы не скажете
File "<stdin>", line 1
print 'We will have a feast tonight unless you say
                                                  ^
SyntaxError: Missing parentheses in call to 'print'
>>>
```

Диалог продолжался неплохо некоторое время, но затем вы сделали крошечную ошибку в использовании языка Python, и Python снова направил на вас копья.

В этот момент вы должны также понять: несмотря на то что Python чрезвычайно сложен и мощен, а также весьма требователен к синтаксису, используемому для общения с ним, тем не менее Python не обладает разумом. В действительности вы только что вели диалог с самим собой, но использовали корректный синтаксис.

В определенном смысле, когда вы используете программу, написанную кем-то другим, в диалоге между вами и этими другими программистами Python действует как посредник. Python предоставляет создателям программ способ, позволяющий определить, как предположительно происходит такой диалог. И уже в нескольких следующих главах вы станете одним из этих программистов, использующих Python для беседы с пользователями своей программы.

Прежде чем мы прекратим наш первый диалог с интерпретатором Python, вы должны узнать, как правильно попрощаться при контакте с обитателями планеты Python:

```
>>> good-bye
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
NameError: name 'good' is not defined
>>> if you don't mind, I need to leave
File "<stdin>", line 1
if you don't mind, I need to leave
         ^
SyntaxError: invalid syntax
>>> quit()
```

Обратите внимание: ошибки различны в первых двух неправильных попытках. Вторая ошибка отличается от первой, потому что `if` – зарезервированное слово, Python заметил его и посчитал, что мы пытаемся что-то сказать, но синтаксис этого предложения оказался некорректным.

Правильный способ попрощаться с Python – ввод `quit()` после интерактивного промпта >>>. Вероятно, вам потребовалось бы некоторое время, чтобы найти этот способ, так что наличие книги под рукой оказывается полезным.

1.6. Терминология: интерпретатор и компилятор

Python – язык высокого уровня, предназначенный для относительно простого чтения и записи команд человеком и чтения и обработки команд компьютерами. К другим языкам высокого уровня относятся Java, C++, PHP, Ruby, Basic, Perl, JavaScript и многие другие. В действительности аппаратура внутри центрального процессорного устройства (ЦПУ) не понимает ни один из этих языков.

ЦПУ понимает только язык, который мы называем машинным языком. Машинный язык очень прост и, откровенно говоря, весьма утомителен для записи, потому что представлен исключительно нулями и единицами:

```
00101000111010010010101000000111 1
11100110000011101010010101101101
...
```

Машинный язык на первый взгляд кажется чрезвычайно простым, потому что использует только нули и единицы, но его синтаксис даже сложнее и намного замысловатее, чем синтаксис Python. Поэтому лишь немногие программисты пишут на машинном языке. Вместо этого мы создаем разнообразные трансляторы, позволяющие программистам писать программы на языках высокого уровня, таких как Python или JavaScript, а эти трансляторы преобразуют программы в машинный язык для их реального выполнения ЦПУ.

Поскольку машинный язык напрямую связан с аппаратурой компьютера, он не является переносимым между различными типами аппаратного обеспечения. Программы, написанные на языках высокого уровня, можно переносить на различные компьютеры, используя соответствующий интерпретатор на новом компьютере или перекомпилируя исходный код для создания версии программы на машинном языке для нового компьютера.

Трансляторы языков программирования делятся на две общие категории: (1) интерпретаторы и (2) компиляторы.

Интерпретатор (interpreter) считывает исходный код программы в том виде, как он написан программистом, выполняет синтаксический анализ (парсинг) исходного кода и интерпретирует (выполняет) инструкции сразу же, «на лету». Python – это интерпретатор, и когда мы запускаем его в интерактивном режиме, то можем вводить строку на его языке, и Python немедленно обрабатывает и выполняет ее, после чего снова готов к приему другой строки, которую мы можем ввести.

Некоторые строки сообщают Python, что мы хотим запомнить некоторое значение на будущее. Необходимо выбрать имя для запоминания такого значения, и мы можем использовать это символьное имя для извлечения этого значения в дальнейшем. Мы применяем термин переменная (variable) для обозначения меток, используемых для обращения к таким сохраненным данным.

```
>>> x = 6
>>> print(x)
6
>>> y = x * 7
>>> print(y)
42
>>>
```

В этом примере мы попросили Python запомнить числовое значение шесть и использовали метку `x`, чтобы можно было получить это значение позже. Мы проверили, действительно ли Python запомнил значение, воспользовавшись функцией `print`. Затем мы попросили Python извлечь значение `x` и умножить его на семь и запомнить новое вычисленное значение в `y`. После этого попросили Python вывести текущее значение, хранящееся в `y`.

Даже если мы вводим все эти команды в одной строке одновременно, Python определяет их как упорядоченную последовательность инструкций, при этом более поздние инструкции могут извлекать данные, созданные в более ранних инструкциях. Мы пишем наш первый простой абзац с четырьмя предложениями в логическом и разумном порядке.

Для интерпретатора вполне естественной является способность вести интерактивный диалог, как показано выше. Компилятор (compiler) требует обработки всей программы в целом в файле, затем он запускает процесс преобразования исходного кода на языке высокого уровня в машинный язык, после чего помещает полученный в результате машинный язык (машинный код) в файл для последующего выполнения.

Если вы пользуетесь системой Windows, то чаще всего эти выполняемые программы на машинном языке имеют суффикс (расширение) «.exe» или «.dll», обозначающие соответственно «executable» (выполняемый файл) и «dynamic link library» (динамически связываемая библиотека). В системах Linux и Macintosh нет специальных суффиксов (расширений), однозначно определяющих файл как выполняемый.

Если открыть выполняемый файл в текстовом редакторе, то его содержимое выглядит абсолютно бессмысленно и нечитабельно:

```
^?ELF^A^A^A^@^@^@^@^@^@^@^@^B^@^C^@^A^@^@^@\xa0\x82
^D^H4^@^@^@\x90^]^@^@^@^@^@^@4^@ ^@^G^@(^@$^@!^@^F^@
^@^@4^@^@^@4\x80^D^H4\x80^D^H\xe0^@^@^@\xe0^@^@^@^E
^@^@^@^D^@^@^@^C^@^@^@^T^A^@^@^T\x81^D^H^T\x81^D^H^S
^@^@^@^S^@^@^@^D^@^@^@^A^@^@^@^A\^D^HQVhT\x83^D^H\xe8
....
```

Очень не просто читать или писать на машинном языке, но, к счастью, существуют интерпретаторы и компиляторы, позволяющие нам писать на языках высокого уровня, таких как Python или C.

В этот момент обсуждения компиляторов и интерпретаторов у вас должен возникнуть небольшой вопрос о самом интерпретаторе Python. На каком языке он написан? На компилируемом? Когда мы вводим команду `python`, что происходит на самом деле?

Интерпретатор Python написан на языке высокого уровня C. Исходный код интерпретатора Python можно посмотреть на сайте www.python.org и даже

самостоятельно поработать с этим исходным кодом. Так что Python сам по себе является программой, скомпилированной в машинный код. Когда вы устанавливаете Python на свой компьютер (или его устанавливает поставщик компьютера), то копируете экземпляр программы Python в машинном коде в свою систему. В Windows выполняемый машинный код интерпретатора Python, вероятнее всего, находится в файле с именем:

```
C:\Python35\python.exe
```

Это даже больше, чем вам в действительности необходимо знать, чтобы стать программистом на Python, но иногда стоит ответить на эти небольшие назойливые вопросы в самом начале.

1.7. Написание программы

Ввод команд в интерпретаторе Python – это отличный способ поэкспериментировать с функциональными возможностями этого языка, но такой подход не рекомендуется для решения более сложных задач.

Если нужно написать программу, то мы используем текстовый редактор, чтобы записать инструкции Python в файл, называемый скриптом (script). По принятому соглашению скрипты Python имеют имена с расширением `.py`.

Для выполнения скрипта вы должны сообщить интерпретатору Python имя соответствующего файла. В окне команд нужно ввести команду `python hello.py`, как показано ниже:

```
$ cat hello.py
print('Hello world!')
$ python hello.py
Hello world!
```

Символ «$» – это промпт (приглашение) операционной системы, а команда `cat hello.py` показывает нам, что файл *hello.py* содержит однострочную программу на языке Python для вывода строки.

Мы вызываем интерпретатор Python и приказываем прочитать этот исходный код из файла *hello.py* вместо вывода приглашения для ввода строк кода Python в интерактивном режиме.

Вы наверняка заметите, что нет необходимости вводить `quit()` в конце программы на языке Python, записанной в файле. Когда Python читает исходный код из файла, то знает, что нужно остановиться при достижении конца этого файла.

1.8. Что такое программа

Самое обобщенное определение программы – это последовательность инструкций на языке Python, предназначенная для выполнения каких-либо действий. Даже наш простой скрипт *hello.py* является программой. Эта одно-