Моей супруге Флоре за 16 потрясающих лет совместной жизни и долгие годы впереди.

Моим сыновьям Джонатану и Эндрю, которым так не терпелось появиться на свет.

Люблю вас всех!

Содержание

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. Издательства «ДМК Пресс» и Springer очень серьезно относятся 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

Я свято верю, что программирование лучше всего изучать с упором на практику. Безусловно, читать книги по программированию и наблюдать за тем, как преподаватель строит алгоритмы у доски, бывает очень полезно, но не меньшую, если не большую пользу принесет самостоятельное решение задач и воплощение изученных концепций программирования на практике. С учетом вышесказанного я решил большую часть данной книги посвятить практическим упражнениям и их решениям, а основы языка Python дать очень кратко и поверхностно.

В книге содержится 186 задач различной степени сложности, охватывающих самые разные учебные дисциплины и сферы жизнедеятельности. Для их решения вам будет достаточно базовых знаний языка, полученных в любом курсе по Python. Каждое из приведенных упражнений призвано улучшить общее понимание концепции языка Python и научить вас справляться с типичными для программирования проблемами и задачами. Также я надеюсь, что подобранные к упражнениям тематики подогреют ваш интерес к их решению.

Во второй части книги приведены подробные решения с комментариями примерно к половине задач из книги. Большинство фрагментов кода включают в себя краткую аннотацию с описанием использованной техники программирования или специфического синтаксиса языка.

Надеюсь, у вас найдется время сравнить свои решения задач с моими, даже если в процессе вы не испытывали никаких трудностей. Это может помочь вам выявить недостатки в своих программах или узнать о новых для себя приемах в программировании, с помощью которых можно было бы решить задачу проще. Иногда вы будете видеть, что ваш вариант решения задачи оказался более быстрым и/или легким по сравнению с моим. Кроме того, если вы застрянете на каком-то из заданий, беглого взгляда в ответ может быть достаточно, чтобы самостоятельно, без посторонней помощи завершить его решение. При написании кода я старался использовать максимально академичный стиль программирования, включая многословные комментарии и хорошо подобранные имена переменных. Советую вам придерживаться таких же правил – пусть ваш код будет не только правильным, но и простым для понимания, а также легкодоступным для будущих исправлений и доработок.

Упражнения, для которых в книге предложено решение, будут помечены словом (Решено). Также для каждой задачи будет указано количество строк в решении. Разумеется, вам не нужно стремиться к тому, чтобы ваш код состоял ровно из такого же количества строк, но эта информация

может помочь вам в поиске решения и не даст окончательно заблудиться в дебрях алгоритмов.

Книгу, которую вы держите в руках, можно использовать по-разному. Вводный курс по Python, добавленный в данном издании, вполне подойдет для самостоятельного изучения основ этого языка программирования. Также книгу можно использовать как сборник упражнений при чтении более подробного учебника по Python, не снабженного достаточным количеством задач для самостоятельного разбора. Можно попробовать изучать язык исключительно по этой книге, но, вероятно, это будет не самый простой способ освоить Python, поскольку теоретическая база, изложенная в начале каждой главы, охватывает лишь основные особенности языка и не вдается в подробности и нестандартные ситуации. Используете вы другие книги для параллельного обучения Python или нет, тщательное изучение теории, выполнение всех без исключения упражнений и их сверка с решениями в конце данной книги, без сомнений, позволят вам выйти на новый уровень программирования.

Благодарности

Я бы хотел сердечно поблагодарить доктора Тома Дженкинса (Tom Jenkyns) за проверку материалов этой книги в процессе ее написания. Его многочисленные советы и комментарии позволили внести важные правки, что положительно сказалось на качестве книги.

Бен Стивенсон
Калгари, Канада
Март, 2019

Часть I

УПРАЖНЕНИЯ

Глава 1

Введение в программирование

Компьютеры помогают нам в выполнении самых разных задач, будь то чтение новостей, просмотр видеороликов, совершение покупок, заказ услуг, решение сложных математических уравнений, общение с семьей и друзьями и многое другое. Все эти задачи требуют от человека ввода исходной информации. Это может быть адрес видеосюжета в сети или название книги для поиска. В ответ компьютер генерирует ссылку на книгу, проигрывает музыку или показывает на экране текст либо изображение.

Но как компьютер узнает, какую именно вводную информацию затребовать? Откуда он знает, какие действия выполнять в ответ? Какой вывод он должен сгенерировать, и в каком виде его необходимо представить пользователю? Ответ на все эти вопросы прост: пользователь предоставляет компьютеру инструкции, согласно которым он действует.

Алгоритм представляет собой конечный набор эффективных шагов для решения той или иной задачи. Шаг можно считать эффективным, если он не содержит в себе двусмысленности и при этом может быть выполнен. Количество шагов должно быть конечным, и должна быть возможность их подсчитать. Рецепты блюд, инструкции для сборки мебели или игрушек и последовательность цифр для открытия кодового замка – все это примеры алгоритмов, с которыми мы сталкиваемся ежедневно в обычной жизни.

Алгоритмы могут быть гибкими и приспосабливаться к конкретным задачам, под которые они разработаны. Слова, числа, линии, стрелки, изображения и другие символы могут использоваться для обозначения шагов, которые должны быть выполнены. И хотя формы алгоритмов могут сильно отличаться, все они включают в себя шаги, которые нужно пройти один за другим для успешного завершения задачи.

Компьютерная программа представляет собой последовательность инструкций, контролирующую работу компьютера. В инструкциях четко прописано, когда необходимо выполнить чтение данных или вывод ре-

зультата и как манипулировать значениями для получения ожидаемого результата. Любой алгоритм должен быть преобразован в компьютерную программу, чтобы компьютер смог приступить к его выполнению. Именно процесс перевода алгоритма в программу и называется программированием, а человек, осуществляющий это преобразование, – программистом.

Компьютерные программы пишут на языках программирования, имеющих определенные правила синтаксиса, которые необходимо строго соблюдать. Если этого не делать, компьютер будет выдавать ошибки, вместо того чтобы следовать инструкциям. За долгое время было создано великое множество языков программирования, каждый из которых обладает своими достоинствами и недостатками. Одними из наиболее популярных и востребованных сегодня являются языки Java, C++, JavaScript, PHP, C# и Python. Несмотря на различия, все эти языки позволяют программисту контролировать поведение компьютера.

В данной книге мы говорим о языке программирования Python, поскольку он является одним из наиболее простых для освоения с нуля. Кроме того, этот язык может использоваться для решения самых разнообразных задач. В первой главе мы поговорим об операциях, отвечающих за ввод/вывод информации и произведение вычислений. В следующих главах затронем конструкции языка, применимые в других областях программирования.

1.1. Хранение и управление значениями

Переменной (variable) называется именованная область памяти, хранящая значение. В Python имя переменной должно начинаться с буквы или символа подчеркивания, после чего могут следовать любые сочетания букв, цифр и символов подчеркивания. При этом имена переменных являются регистрозависимыми, то есть имена `count`, `Count` и `COUNT` будут адресовать разные переменные. Создание переменной происходит в момент выполнения операции присваивания значения. При этом имя переменной, которой мы присваиваем значение, должно располагаться слева от знака присваивания (`=`), а само значение – справа. Например, в следующем выражении будет создана переменная `x`, которой будет присвоено значение `5`:

```
x = 5
```

Отметим, что справа от знака присваивания может находиться не только простое значение, но и выражение любой степени сложности – с использованием скобок, математических операторов, чисел и переменных, созданных на более ранних этапах. Самыми распространенными *мате-*

матическими операторами, представленными в языке Python, являются *сложение* (+), *вычитание* (-), *умножение* (*), *деление* (/) и *возведение в степень* (**). Также к простым операторам относятся *деление без остатка* (//) и вычисление *остатка от деления* (%). Первый возвращает целую часть от полученного в результате деления частного, а второй – остаток.

В следующем примере переменной y присваивается результат возведения переменной x в квадрат, к которому прибавлена единица:

```
y = 1 + x ** 2
```

В Python соблюдается строгий порядок выполнения математических операций. Поскольку переменной x ранее уже было присвоено значение 5, а оператор возведения в степень имеет более высокий приоритет по сравнению с оператором сложения, переменная y получит значение 26.

Одна и та же переменная может находиться как слева, так и справа от оператора присваивания, как показано ниже:

```
y = y - 6
```

Хотя изначально может показаться, что это выражение не имеет смысла, фактически в Python оно вычисляется так же точно, как и любое другое. Сначала будет вычислено значение выражения, стоящего справа от знака равенства, а затем оно будет присвоено переменной, указанной слева. В нашем конкретном случае на первом шаге значение переменной y, равное 26, будет уменьшено на 6, в результате чего будет получено значение 20. Оно и будет записано в ту же переменную y, заменив собой прежнее значение 26. После этого обращение к переменной y будет возвращать значение 20 – до тех пор, пока ей не будет присвоено новое значение.

1.2. Вызов функций

Есть масса привычных последовательностей действий, которые повторно используются программами. Это может быть чтение ввода с клавиатуры, сортировка списков или извлечение корня из числа. Python предлагает ряд *функций* для выполнения этих и многих других типовых операций. Программы, которые мы будем создавать, будут регулярно вызывать эти функции, так что нам не придется раз за разом решать похожие друг на друга задачи самостоятельно.

Вызвать функцию можно по имени с обязательными скобками после него. Многие функции требуют передачи значений в скобках, таких как список элементов для сортировки или число, из которого необходимо извлечь квадратный корень. Эти переданные значения называются *аргументами функции* и помещаются в круглые скобки, следующие после ее

имени при вызове. Если функция принимает несколько аргументов, они должны разделяться запятыми.

Многие функции вычисляют результат, который может быть сохранен в переменной посредством операции присваивания. В этом случае имя переменной указывается слева от оператора присваивания, а функция располагается справа. Например, в следующем выражении вызывается функция округления числа, результат которой присваивается переменной `r`:

```
r = round(q)
```

Значение переменной `q`, которая должна быть инициализирована раньше, передается функции `round` в качестве аргумента. При выполнении функция находит ближайшее к значению переменной `q` целое число и возвращает его. После этого возвращенное число присваивается переменной `r`.

1.2.1. Чтение ввода

Программы, написанные на Python, могут читать ввод с клавиатуры при помощи функции *input*. Вызов этой функции предписывает программе остановить выполнение и ждать завершения пользовательского ввода с клавиатуры. После нажатия пользователем клавиши `Enter` символы, введенные им ранее, возвращаются функцией `input`, и выполнение программы продолжается. Обычно возвращенное функцией `input` значение записывается в переменную при помощи оператора присваивания, чтобы впоследствии его можно было использовать. Например, следующее выражение считывает ввод пользователя с клавиатуры и записывает результат в переменную `a`.

```
a = input()
```

Функция `input` всегда возвращает значение *строкового типа* (string), что в компьютерной терминологии означает последовательность символов. Если вы запрашивали у пользователя его имя, название книги или улицы, будет логично сохранить введенное значение в переменной как строку. Если же речь идет о возрасте, температуре или сумме счета в ресторане, текстовое значение, введенное пользователем, лучше будет конвертировать в число. Программист, следуя логике, должен определить, будет это число целым или с плавающей запятой (то есть с десятичными знаками после запятой). Преобразование значения в целочисленный тип осуществляется путем вызова функции *int*, тогда как перевод в числовой тип с плавающей запятой можно выполнить при помощи функции *float*. Функции приведения типа принято использовать непосредственно в том же выражении, где запрашивается ввод с клавиатуры. Например, в сле-

дующем примере у пользователя запрашивается его имя, количество приобретаемых товаров и цена за штуку. Каждое введенное значение сохраняется в соответствующей переменной, при этом непосредственно перед сохранением оно преобразуется в нужный тип: имя остается строковым, количество товаров становится целочисленным, а цена – числовым типом с плавающей запятой.

```
name = input("Введите имя: ")
quantity = int(input("Сколько товаров вы желаете приобрести? "))
price = float(input("Какова цена за единицу товара? "))
```

Заметьте, что при каждом вызове функции `input` мы передавали ей аргумент в виде вопроса. Этот аргумент является необязательным и при наличии будет показываться на экране непосредственно перед вводом с клавиатуры. Тип этого аргумента должен быть строковым. Такие значения заключаются в кавычки, чтобы анализатор Python понимал, что необходимо воспринимать их как строковые, а не пытаться искать подходящие имена функций или переменных.

Математические операции допустимо производить как с целочисленными значениями, так и с числами с плавающей запятой. Например, в следующем выражении мы пытаемся вычислить общую сумму приобретаемых товаров, записывая результат в новую переменную:

```
total = quantity * price
```

Это выражение успешно выполнится только в том случае, если составляющие его переменные `quantity` и `price` были заранее преобразованы в числовой тип при помощи функций `int` и `float`. Попытка выполнить операцию умножения без предварительного преобразования переменных в числовой тип повлечет за собой ошибку программы.

1.2.2. Вывод результата

Вывести значение на экран можно при помощи функции *print*. Она может быть вызвана с одним аргументом, представляющим значение, которое необходимо вывести на экран. Например, в следующем фрагменте кода на экран будут последовательно выведены единица, слово `Hello` с восклицательным знаком и текущее содержимое переменной `x`. При этом переменная `x` может характеризоваться строковым типом данных, числовым или любым другим, о которых мы пока не упоминали. Каждое значение будет выведено на отдельной строке.

```
print(1)
print("Hello!")
print(x)
```

Допустимо передавать в функцию `print` сразу несколько аргументов для их последовательного вывода на экран. Все аргументы функции при этом должны быть разделены запятыми, как показано ниже.

```
print("Когда x равен", x, ", y будет равен", y)
```

Все переданные аргументы будут выведены на одной строке. При этом аргументы, заключенные в кавычки, будут автоматически восприниматься как строковые и выводиться как есть. Остальные аргументы в этом списке – это переменные. При выводе на экран переменной мы будем видеть ее текущее значение. Также стоит отметить, что все аргументы на экране будут отделяться друг от друга пробелами.

Помимо строковых констант и переменных, переданные в функцию `print` аргументы могут представлять собой сложные выражения, включающие в себя скобки, математические операторы и вызовы других функций. Рассмотрите следующий пример:

```
print("Если умножить", x, "на", y, "получится", x * y)
```

Выражение `x * y` будет автоматически вычислено, и на экране появится полученный результат умножения.

1.2.3. Импорт дополнительных функций

Некоторые функции, такие как `input` и `print`, используются в программах на Python достаточно часто, другие – реже. В результате было решено наиболее популярные функции сделать доступными всем по умолчанию, а остальные разбить на *модули*, которые разработчики смогут подключать к своим программам по мере необходимости. К примеру, специфические математические функции собраны в отдельном модуле с именем `math`. Импортировать их в свою программу можно, написав соответствующую инструкцию, показанную ниже:

```
import math
```

В модуле `math` содержится огромное количество математических функций, включая `sqrt`, `ceil`, `sin` и другие. Чтобы воспользоваться функцией из загруженного модуля, необходимо перед ней указать имя этого модуля с разделяющей их точкой. Например, в следующем выражении выполняется извлечение квадратного корня из переменной `y` (которая должна быть объявлена заранее) путем вызова функции `sqrt` из модуля `math`, и результат сохраняется в новой переменной `z`:

```
z = math.sqrt(y)
```

Также распространенными модулями в языке Python являются `random`, `time`, `sys` и многие другие. Больше информации о модулях и содержащихся в них функциях можно найти на просторах интернета.

1.3. Комментарии

Комментарии позволяют разработчикам оставлять заметки в программе о том, как, что и зачем они делали в данном месте кода. Эта информация бывает крайне полезной, если приходится возвращаться к своей программе через долгое время или разбираться в коде, написанном другим разработчиком. При выполнении программы компьютер игнорирует все комментарии в коде, они присутствуют там исключительно для человека.

В Python строки с *комментариями* должны начинаться с символа решетки (#). Комментарий распространяется от первого вхождения этого символа и до конца строки. Комментарий может занимать как всю строку целиком, так и ее часть – в этом случае символ # ставится после строки кода, и комментарий будет распространяться до конца строки.

Рабочие файлы Python обычно начинаются с блока комментариев, кратко описывающих назначение программы. Это поможет любому стороннему разработчику быстро понять, для чего предназначен данный файл, без необходимости разбираться в коде. По комментариям также легко можно определить, какие блоки в программе чем занимаются при достижении общего результата. При решении задач из этой книги я настоятельно рекомендовал бы вам активно использовать комментарии в своем коде.

1.4. Форматирование значений

Иногда бывает, что в результате выполнения математической операции получается число с большим количеством знаков после запятой. И хотя в некоторых программах критически важно выводить результирующие числа с полной точностью, зачастую бывает необходимо округлить их до определенного количества знаков после запятой. Бывает также, что совокупность целых чисел нужно вывести на экран в виде столбцов. Конструкции форматирования в Python позволяют решить эти и многие другие задачи.

Для этого разработчику достаточно указать Python необходимый *спецификатор формата* (format specifier) при выводе его на экран. Спецификатор формата представляет собой последовательность символов, определяющую нюансы форматирования значения. В числе прочих в нем

указывается символ, говорящий о том, какой тип данных при форматировании значения должен использоваться. Например, символ `f` используется для форматирования в виде числа с плавающей запятой, символы `d` и `i` говорят о том, что мы имеем дело с целыми числами, а `s` символизирует строки. Этим главным управляющим символам могут предшествовать знаки, обеспечивающие тонкую настройку формата отображения. В данном разделе мы рассмотрим только вариант тонкой настройки форматирования чисел с плавающей запятой таким образом, чтобы отображалось определенное количество знаков после запятой, а значения в целом занимали указанное количество символов при выводе, что позволит форматировать их в виде аккуратных столбцов. С помощью спецификатора форматирования можно выполнить и другие преобразования выводимых значений, но их описание выходит за рамки этой книги.

Итак, ограничить количество видимых знаков после запятой в числах с плавающей запятой можно, поставив точку и это значение непосредственно перед символом `f` в спецификаторе. Например, спецификатор `.2f` говорит о том, что при выводе значения на экран в нем должны остаться два знака после запятой, а спецификатор `.7f` предписывает наличие семи десятичных знаков. При ограничении количества знаков после запятой выполняется операция округления, тогда как лишние позиции дополняются нулями. Количество десятичных знаков не может быть указано при форматировании строковых или целочисленных значений.

В то же время значения всех трех перечисленных типов данных могут быть отформатированы таким образом, чтобы они занимали минимальное количество знакомест. Для этого достаточно указать минимальную ширину значения в символах, что удобно при выполнении форматирования данных в виде столбцов. Число, определяющее минимальную ширину значения при выводе, необходимо вставить в спецификаторе слева от символа `d`, `i`, `f` или `s` – перед точкой с ограничителем количества десятичных знаков, если таковые имеются. Например, спецификатор `8d` говорит о том, что значение должно быть отформатировано как целое число и при выводе должно занимать минимум восемь знакомест, тогда как `6.2f` предполагает в значении с плавающей запятой наличие двух знаков после запятой, а минимальное занимаемое место значением должно составлять шесть символов, включая дробную часть. При необходимости ведущие позиции при форматировании значений дополняются пробелами.

После формирования правильного спецификатора к нему слева остается добавить символ процента (`%`), что придаст ему законченный и корректный вид. Обычно спецификаторы форматирования используются внутри строк, причем они могут занимать как всю строку, так и ее часть. Вот несколько примеров использования спецификаторов в строках: `"%8d"`, `"Сумма долга: %.2f"` и `"Привет, %s! Добро пожаловать к нам"`.

Примечание. Язык программирования Python предлагает сразу несколько способов выполнения форматирования значений, включая использование форматирующего оператора (%), функцию и метод `format`, шаблонные строки и f-строки. На протяжении этой книги мы будем применять с данной целью форматирующий оператор, но вы вольны использовать любую из существующих техник.

После создания спецификатора его можно использовать для форматирования значений. Строка, содержащая спецификатор, должна располагаться слева от *форматирующего оператора* (%), а значение, которое требуется отформатировать, – справа. При выполнении оператора значение, указанное справа от него, вставляется в строку слева (на место спецификатора с использованием указанных правил форматирования), в результате чего формируется итоговая строка. При этом все оставшиеся части строки, за исключением спецификатора, остаются неизменными.

Допустимо осуществлять форматирование сразу нескольких значений в одной строке, разместив более одного спецификатора в тексте слева от форматирующего оператора. В этом случае значения для форматирования справа от форматирующего оператора необходимо перечислить через запятую и заключить их в круглые скобки.

Форматирование строк зачастую выполняется в рамках функции `print`. В следующем примере сначала на экран выводится значение переменной `x` с двумя знаками после запятой, а затем – целая фраза, форматирующая значения сразу из двух переменных.

```
print("%.2f" % x)
print("%s съел %d пирожков!" % (name, numCookies))
```

Некоторые другие варианты форматирования приведены в табл. 1.1. Переменные `x`, `y` и `z` предварительно были инициализированы значениями 12, –2,75 и «Andrew».

Таблица 1.1. Форматирование значений в Python

Фрагмент кода:
Результат:
Описание:
	`"%d" % x`
	`"12"`
	Значение из переменной `x` отображается в формате целого числа

Фрагмент кода:
Результат:
Описание:
	`"%f" % y`
	`"-2.75"`
	Значение из переменной `y` отображается в формате числа с плавающей запятой

Фрагмент кода:
Результат:
Описание:
	`"%d и %f" % (x, y)`
	`"12 и -2.75"`
	Значение из переменной `x` отображается в формате целого числа, а значение из переменной `y` отображается в формате числа с плавающей запятой. Остальные символы в строке остаются без изменений

Таблица 1.1 *(окончание)*

Фрагмент кода:
Результат:
Описание:
	`"%.4f" % x`
	`"12.0000"`
	Значение из переменной x отображается в формате числа с плавающей запятой с четырьмя десятичными знаками

Фрагмент кода:
Результат:
Описание:
	`"%.1f" % y`
	`"-2.8"`
	Значение из переменной y отображается в формате числа с плавающей запятой с одним десятичным знаком. При выводе значение будет округлено, поскольку изначально в переменной находится значение с большим количеством десятичных знаков

Фрагмент кода:
Результат:
Описание:
	`"%10s" % z`
	`"    Andrew"`
	Значение из переменной z отображается в формате строки, занимающей минимум десять знакомест. Поскольку в слове Andrew шесть букв, к результату добавится четыре ведущих пробела

Фрагмент кода:
Результат:
Описание:
	`"%4s" % z`
	`"Andrew"`
	Значение из переменной z отображается в формате строки, занимающей минимум четыре знакоместа. Поскольку в слове Andrew шесть букв, результат окажется равен исходному значению переменной z

Фрагмент кода:
Результат:
Описание:
	`"%8i%8i" % (x, y)`
	`"      12      -2"`
	Значения из переменных x и y отображаются в формате целого числа, занимающего минимум восемь знакомест. При этом были добавлены ведущие пробелы. При преобразовании значения переменной y в целочисленный тип его дробная часть была отброшена (а не округлена)

1.5. Работа со строками

Так же, как с числами, в Python можно манипулировать со строками при помощи специальных операторов и передавать их в функции в качестве аргументов. Самые распространенные операции, которые приходится выполнять со строковыми значениями, – это их конкатенация, определение длины строки и извлечение определенных символов и подстрок. Этим действиям и будет посвящен данный раздел. Информацию о других операциях со строками в Python можно без труда найти в интернете.

Строки в Python можно *конкатенировать* при помощи оператора сложения (+). В результате строка, стоящая справа от оператора конкатенации, будет добавлена в конец строки, стоящей слева, с образованием нового строкового значения. Например, в представленном ниже фрагменте кода сначала запрашивается информация об имени и фамилии у пользователя, после чего собирается новое строковое значение, состоящее из фамилии

и имени, разделенных запятой и пробелом. Полученный результат выводится на экран.

```
# Запрашиваем у пользователя имя и фамилию
first = input("Введите имя: ")
last = input("Введите фамилию: ")

# Конкатенируем строки
both = last + ", " + first

# Отображаем результат
print(both)
```

Количество символов в строке называется длиной строки. Это всегда положительное значение числового типа, и получить его можно при помощи функции *len*. На вход функция требует единственный строковый аргумент и возвращает его длину. Следующий пример демонстрирует применение функции `len` для определения длины имени человека:

```
# Спрашиваем имя пользователя
first = input("Введите имя: ")

# Вычисляем длину строки
num_chars = len(first)

# Отображаем результат
print("В вашем имени", num_chars, "символов")
```

Иногда необходимо получить доступ к *конкретным символам в строке*. Например, вам может понадобиться извлечь первые символы из имени, отчества и фамилии пользователя, чтобы собрать инициалы.

Каждый символ в строке имеет свой уникальный индекс. Первый символ обладает индексом 0, а последний – индексом, равным длине строки минус один. Получить доступ к символу в строке по индексу можно, поместив его значение в квадратные скобки непосредственно после строковой переменной. В следующей программе демонстрируется отображение на экране инициалов человека:

```
# Запрашиваем имя пользователя
first = input("Введите имя: ")
middle = input("Введите отчество: ")
last = input("Введите фамилию: ")
# Извлекаем первый символ из всех трех переменных и собираем их вместе
initials = first[0] + middle[0] + last[0]
# Выводим инициалы
print("Ваши инициалы:", initials)
```

Получить из строки несколько символов, стоящих подряд, можно, указав два индекса в квадратных скобках, разделенных двоеточием. Эта операция по-другому называется выполнением *среза* (slicing) строки. Исполь-

зование срезов позволяет эффективно извлекать подстроки из строковых переменных.

1.6. Упражнения

Задачи для самостоятельного выполнения, приведенные в данном разделе, помогут вам воплотить на практике знания, полученные в этой главе. И хотя это будут небольшие по размеру фрагменты кода, они должны стать важным шагом на пути к полноценным программам, которые вы будете писать в будущем.

Упражнение 1. Почтовый адрес

(*Решено. 9 строк*)

Напишите несколько строк кода, выводящих на экран ваше имя и почтовый адрес. Адрес напишите в формате, принятом в вашей стране. Никакого ввода от пользователя ваша первая программа принимать не будет, только вывод на экран и больше ничего.

Упражнение 2. Приветствие

(*9 строк*)

Напишите программу, запрашивающую у пользователя его имя. В ответ на ввод на экране должно появиться приветствие с обращением по имени, введенному с клавиатуры ранее.

Упражнение 3. Площадь комнаты

(*Решено. 13 строк*)

Напишите программу, запрашивающую у пользователя длину и ширину комнаты. После ввода значений должен быть произведен расчет площади комнаты и выведен на экран. Длина и ширина комнаты должны вводиться в формате числа с плавающей запятой. Дополните ввод и вывод единицами измерения, принятыми в вашей стране. Это могут быть футы или метры.

Упражнение 4. Площадь садового участка

(*Решено. 15 строк*)

Создайте программу, запрашивающую у пользователя длину и ширину садового участка в футах. Выведите на экран площадь участка в акрах.

Подсказка. В одном акре содержится 43 560 квадратных футов.