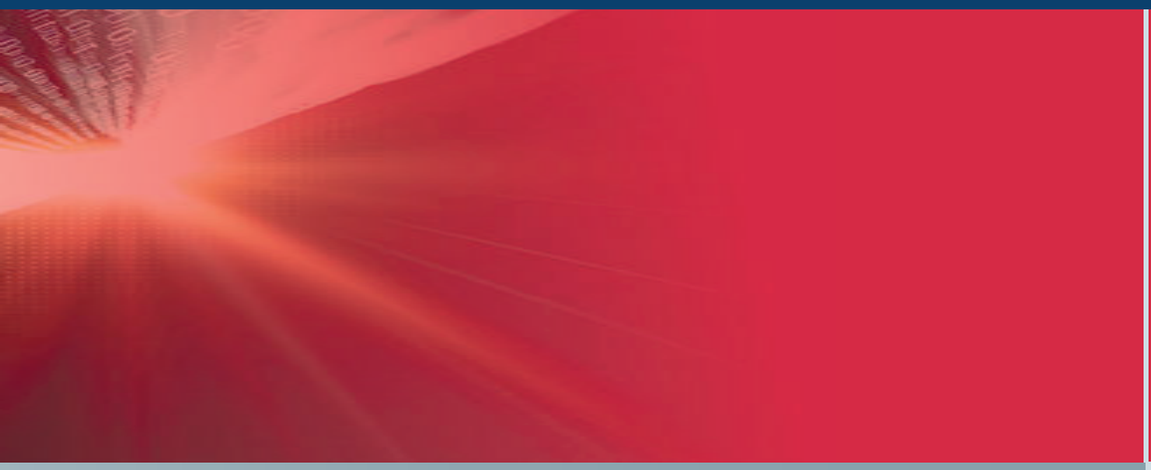


В книге содержится **186 задач** по программированию различной степени сложности. При написании кода автор использует максимально академичный стиль, упрощающий для читателя знакомство с языком Python. Упражнения, для которых в книге предложено решение, снабжены пометкой «Решено»; для каждой задачи указано количество строк в решении.

Задачи в каждой главе сгруппированы по темам:

- введение в программирование;
- принятие решений;
- списки;
- повторения;
- функции;
- списки;
- словари;
- файлы и исключения;
- рекурсия.

Издание будет полезно начинающим программистам, которые обладают базовыми знаниями языка Python. Те, кто изучает этот язык с нуля, могут использовать книгу как дополнение к учебнику.

**Интернет-магазин:**
www.dmkpress.com

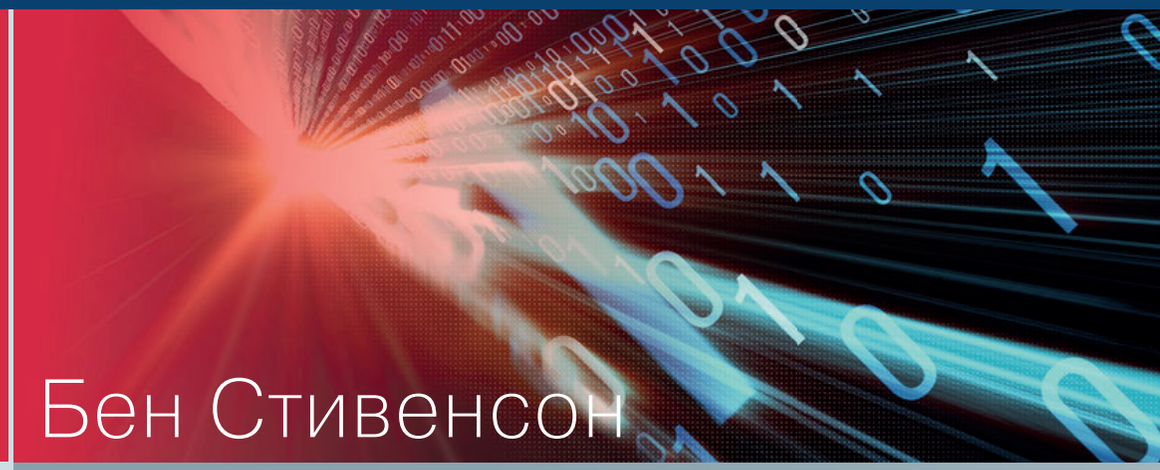
**Оптовая продажа:**
КТК «Галактика»
books@alians-kniga.ru

Springer

ДМК издательство
www.дмк.рф

ISBN 978-5-97060-916-3

Python. Сборник упражнений



Бен Стивенсон

# Python. Сборник упражнений

## Введение в язык Python с задачами и решениями

Springer

ДМК издательство

Бен Стивенсон

# Python
# Сборник упражнений

# The Python Workbook

## A Brief Introduction with Exercises and Solutions

Ben Stephenson

Springer

# Python
# Сборник упражнений

## Введение в язык Python
## с задачами и решениями

**Бен Стивенсон**

ДМК ПРЕСС
издательство
Москва, 2021

**УДК 004.438Python**
**ББК 32.973.22**
**С80**

**Стивенсон Б.**
С80 Python. Сборник упражнений / пер. с англ. А. Ю. Гинько. – М.: ДМК Пресс, 2021. – 238 с.: ил.

**ISBN 978-5-97060-916-3**

Сборник содержит 186 задач по программированию разной степени сложности. Для ряда упражнений изложены решения с подробным разбором фрагментов кода и синтаксических конструкций языка Python.

В книге представлен простой и понятный стиль программирования. Чтобы решить приведенные здесь задачи, достаточно базовых знаний языка Python. По мере изучения материала читатель отрабатывает навык использования таких техник, как условные выражения, циклы, основные функции, списки, словари, рекурсия и работа с файлами.

Издание будет полезно студентам, делающим первые шаги в программировании и желающим продвинуться в этой области. Книга может использоваться и как вводный курс по Python, и как практикум, дополняющий учебник программирования на этом языке.

УДК 004.438Python
ББК 32.973.22



ISBN 978-3-030-18872-6 (англ.)

ISBN 978-5-97060-916-3 (рус.)

*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:<br>Результат:<br>Описание: | `"%d" % x`<br>`"12"`<br>Значение из переменной `x` отображается в формате целого числа |
| Фрагмент кода:<br>Результат:<br>Описание: | `"%f" % y`<br>`"-2.75"`<br>Значение из переменной `y` отображается в формате числа с плавающей запятой |
| Фрагмент кода:<br>Результат:<br>Описание: | `"%d и %f" % (x, y)`<br>`"12 и -2.75"`<br>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:<br>Результат:<br>Описание: | `"%.4f" % x`<br>`"12.0000"`<br>Значение из переменной x отображается в формате числа с плавающей запятой с четырьмя десятичными знаками |
| Фрагмент кода:<br>Результат:<br>Описание: | `"%.1f" % y`<br>`"-2.8"`<br>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:<br>Результат:<br>Описание: | `"%10s" % z`<br>`"    Andrew"`<br>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:<br>Результат:<br>Описание: | `"%4s" % z`<br>`"Andrew"`<br>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:<br>Результат:<br>Описание: | `"%8i%8i" % (x, y)`<br>`"      12      -2"`<br>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.

## *Упражнение 5. Сдаем бутылки*

(*Решено. 15 строк*)

Во многих странах в стоимость стеклотары закладывается определенный депозит, чтобы стимулировать покупателей напитков сдавать пустые бутылки. Допустим, бутылки объемом 1 литр и меньше стоят $0,10, а бутылки большего объема – $0,25.

Напишите программу, запрашивающую у пользователя количество бутылок каждого размера. На экране должна отобразиться сумма, которую можно выручить, если сдать всю имеющуюся посуду. Отформатируйте вывод так, чтобы сумма включала два знака после запятой и дополнялась слева символом доллара.

## *Упражнение 6. Налоги и чаевые*

(*Решено. 17 строк*)

Программа, которую вы напишете, должна начинаться с запроса у пользователя суммы заказа в ресторане. После этого должен быть произведен расчет налога и чаевых официанту. Вы можете использовать принятую в вашем регионе налоговую ставку для подсчета суммы сборов. В качестве чаевых мы оставим 18 % от стоимости заказа без учета налога. На выходе программа должна отобразить отдельно налог, сумму чаевых и итог, включая обе составляющие. Форматируйте вывод таким образом, чтобы все числа отображались с двумя знаками после запятой.

## Упражнение 7. Сумма первых *n* положительных чисел

(*Решено. 11 строк*)

Напишите программу, запрашивающую у пользователя число и подсчитывающую сумму натуральных положительных чисел от 1 до введенного пользователем значения. Сумма первых $n$ положительных чисел может быть рассчитана по формуле:

$$\text{sum} = \frac{(n)(n+1)}{2}.$$

## *Упражнение 8. Сувениры и безделушки*

(*15 строк*)

Интернет-магазин занимается продажей различных сувениров и безделушек. Каждый сувенир весит 75 г, а безделушка – 112 г. Напишите программу, запрашивающую у пользователя количество тех и других покупок, после чего выведите на экран общий вес посылки.

## *Упражнение 9. Сложные проценты*

(*19 строк*)

Представьте, что вы открыли в банке сберегательный счет под 4 % годовых. Проценты банк рассчитывает в конце года и добавляет к сумме счета. Напишите программу, которая запрашивает у пользователя сумму первоначального депозита, после чего рассчитывает и выводит на экран сумму на счету в конце первого, второго и третьего годов. Все суммы должны быть округлены до двух знаков после запятой.

## *Упражнение 10. Арифметика*

(*Решено. 22 строки*)

Создайте программу, которая запрашивает у пользователя два целых числа `a` и `b`, после чего выводит на экран результаты следующих математических операций:

- сумма `a` и `b`;
- разница между `a` и `b`;
- произведение `a` и `b`;
- частное от деления `a` на `b`;
- остаток от деления `a` на `b`;
- десятичный логарифм числа `a`;
- результат возведения числа `a` в степень `b`.

**Подсказка.** Функцию `log10` вы найдете в модуле `math`.

## *Упражнение 11. Потребление топлива*

(*13 строк*)

В США потребление автомобильного топлива исчисляется в *милях на галлон* (miles-per-gallon – MPG). В то же время в Канаде этот показатель обычно выражается в *литрах на 100 км* (liters-per-hundred kilometers – L/100 km). Используйте свои исследовательские способности, чтобы определить формулу перевода первых единиц исчисления в последние. После этого напишите программу, запрашивающую у пользователя показатель потребления топлива автомобилем в американских единицах и выводящую его на экран в канадских единицах.

## *Упражнение 12. Расстояние между точками на Земле*

(*27 строк*)

Как известно, поверхность планеты Земля искривлена, и расстояние между точками, характеризующимися одинаковыми градусами по долготе, может быть разным в зависимости от широты. Таким образом, для вычис-

ления расстояния между двумя точками на Земле одной лишь теоремой Пифагора не обойтись.

Допустим, $(t_1, g_1)$ и $(t_2, g_2)$ – координаты широты и долготы двух точек на поверхности Земли. Тогда расстояние в километрах между ними с учетом искривленности планеты можно найти по следующей формуле:

$$\text{distance} = 6371{,}01 \times \arccos(\sin(t_1) \times \sin(t_2) + \cos(t_1) \times \cos(t_2) \times \cos(g_1 - g_2)).$$

**Примечание.** Число 6371,01 в этой формуле, конечно, было выбрано не случайно и представляет собой среднее значение радиуса Земли в километрах.

Напишите программу, в которой пользователь будет вводить координаты двух точек на Земле (широту и долготу) в градусах. На выходе мы должны получить расстояние между этими точками при следовании по кратчайшему пути по поверхности планеты.

**Подсказка.** Тригонометрические функции в Python оперируют радианами. Таким образом, вам придется введенные пользователем величины из градусов перевести в радианы, прежде чем вычислять расстояние между точками. В модуле `math` есть удобная функция с названием *radians-Функции:radians*, служащая как раз для перевода градусов в радианы.

## *Упражнение 13. Размен*

(*Решено. 35 строк*)

Представьте, что вы пишете программное обеспечение для автоматической кассы в магазине самообслуживания. Одной из функций, заложенных в кассу, должен быть расчет сдачи в случае оплаты покупателем наличными.

Напишите программу, которая будет запрашивать у пользователя сумму сдачи в центах. После этого она должна рассчитать и вывести на экран, сколько и каких монет потребуется для выдачи указанной суммы, при условии что должно быть задействовано минимально возможное количество монет. Допустим, у нас есть в распоряжении монеты достоинством в 1, 5, 10, 25 центов, а также в 1 (loonie) и 2 (toonie) канадских доллара.

**Примечание.** Монета номиналом в 1 доллар была выпущена в Канаде в 1987 году. Свое просторечное название (loonie) она получила от изображения полярной гагары (loon) на ней. Двухдолларовая монета, вышедшая девятью годами позже, была прозвана toonie, как комбинация из слов два (two) и loonie.

## *Упражнение 14. Рост*

(*Решено. 16 строк*)

Многие люди на планете привыкли рассчитывать рост человека в футах и дюймах, даже если в их стране принята метрическая система. Напишите программу, которая будет запрашивать у пользователя количество футов, а затем дюймов в его росте. После этого она должна пересчитать рост в сантиметры и вывести его на экран.

**Подсказка.** Один фут равен 12 дюймам, а один дюйм – 2,54 см.

## *Упражнение 15. Расстояние*

(*20 строк*)

Для этого упражнения вам необходимо будет написать программу, которая будет запрашивать у пользователя расстояние в футах. После этого она должна будет пересчитать это число в дюймы, ярды и мили и вывести на экран. Коэффициенты для пересчета единиц вы без труда найдете в интернете.

## *Упражнение 16. Площадь и объем*

(*15 строк*)

Напишите программу, которая будет запрашивать у пользователя радиус и сохранять его в переменной `r`. После этого она должна вычислить площадь круга с заданным радиусом и объем шара с тем же радиусом. Используйте в своих вычислениях константу `pi` из модуля `math`.

**Подсказка.** Площадь круга вычисляется по формуле area $= \pi r^2$, а объем шара – по формуле volume $= \frac{4}{3}\pi r^3$.

## *Упражнение 17. Теплоемкость*

(*Решено. 23 строки*)

Количество энергии, требуемое для повышения температуры одного грамма материала на один градус Цельсия, называется удельной теплоемкостью материала и обозначается буквой $C$. Общее количество энергии ($q$), требуемое для повышения температуры $m$ граммов материала на $\Delta T$ градусов Цельсия, может быть рассчитано по формуле:

$$q = mC\Delta T.$$

Напишите программу, запрашивающую у пользователя массу воды и требуемую разницу температур. На выходе вы должны получить количество энергии, которое необходимо добавить или отнять для достижения желаемого температурного изменения.

**Подсказка.** Удельная теплоемкость воды равна 4,186 $\frac{Дж}{г \cdot С}$. Поскольку вода обладает плотностью 1 грамм на миллилитр, в данном упражнении можно взаимозаменять граммы и миллилитры.

Расширьте свою программу таким образом, чтобы выводилась также стоимость сопутствующего нагрева воды. Обычно принято измерять электричество в кВт·ч, а не в джоулях. Для данного примера предположим, что электричество обходится нам в 8,9 цента за один кВт·ч. Используйте свою программу для подсчета стоимости нагрева одной чашки кофе.

**Подсказка.** Для решения второй части задачи вам придется найти способ перевода единиц электричества между джоулями и кВт·ч.

## *Упражнение 18. Объем цилиндра*

(*15 строк*)

Объем цилиндра может быть рассчитан путем умножения площади круга, лежащего в его основе, на высоту. Напишите программу, в которой пользователь будет задавать радиус цилиндра и его высоту, а в ответ получать его объем, округленный до одного знака после запятой.

## *Упражнение 19. Свободное падение*

(*Решено. 15 строк*)

Напишите программу для расчета скорости объекта во время его соприкосновения с землей. Пользователь должен задать высоту в метрах, с которой объект будет отпущен. Поскольку объекту не будет придаваться ускорение, примем его начальную скорость за 0 м/с. Предположим, что ускорение свободного падения равно 9,8 м/с$^2$. При известных начальной скорости ($v_i$), ускорении ($a$) и дистанции ($d$) можно вычислить скорость при соприкосновении объекта с землей по формуле $v_f = \sqrt{v_i^2 + 2ad}$.

## *Упражнение 20. Уравнение состояния идеального газа*

(*19 строк*)

Уравнение состояния идеального газа представляет собой математическую аппроксимацию поведения газов в условиях изменения давления, объема и температуры. Обычно соответствующая формула записывается так:

$$PV = nRT,$$

где $P$ – это давление в паскалях, $V$ – объем в литрах, $n$ – количество вещества в молях, $R$ – универсальная газовая постоянная, равная 8,314 Дж/(моль·К), а $T$ – температура по шкале Кельвина.

Напишите программу для измерения количества газа в молях при заданных пользователем давлении, объеме и температуре. Проверьте свою программу путем вычисления количества газа в баллоне для дайвинга. Типичный баллон вмещает 12 л газа под давлением 20 000 000 Па (примерно 3000 фунтов на кв. дюйм). Температуру в комнате примем за 20° по шкале Цельсия или 68° по Фаренгейту.

**Подсказка.** Чтобы перевести температуру из градусов Цельсия в Кельвины, необходимо прибавить к ней 273,15. Из Фаренгейта в Кельвины температура переводится путем вычитания из нее 32, умножения результата на $\frac{5}{9}$ и прибавления тех же 273,15.

## *Упражнение 21. Площадь треугольника*

(*13 строк*)

Площадь треугольника может быть вычислена с использованием следующей формулы, где $b$ – длина основания треугольника, а $h$ – его высота:

$$\text{area} = \frac{b \times h}{2}.$$

Напишите программу, в которой пользователь сможет вводить значения для переменных $b$ и $h$, после чего на экране будет отображена площадь треугольника с заявленными основанием и высотой.

## *Упражнение 22. Площадь треугольника (снова)*

(*16 строк*)

В предыдущем упражнении мы вычисляли площадь треугольника при известных длинах его основания и высоты. Но можно рассчитать площадь

и на основании длин всех трех сторон треугольника. Пусть $s_1$, $s_2$ и $s_3$ – длины сторон, а $s = (s_1 + s_2 + s_3)/2$. Тогда площадь треугольника может быть вычислена по следующей формуле:

$$\text{area} = \sqrt{s \times (s - s_1) \times (s - s_2) \times (s - s_3)}.$$

Разработайте программу, которая будет принимать на вход длины всех трех сторон треугольника и выводить его площадь.

## Упражнение 23. Площадь правильного многоугольника

(*Решено. 14 строк*)

Многоугольник называется правильным, если все его стороны и углы равны. Площадь такой фигуры можно вычислить по следующей формуле, в которой $s$ – длина стороны, а $n$ – количество сторон:

$$\text{area} = \frac{n \times s^2}{4 \times \tan\left(\frac{\pi}{n}\right)}.$$

Напишите программу, которая будет запрашивать у пользователя значения переменных $s$ и $n$ и выводить на экран площадь правильного многоугольника, построенного на основании этих величин.

## Упражнение 24. Единицы времени

(*22 строки*)

Создайте программу, в которой пользователь сможет ввести временной промежуток в виде количества дней, часов, минут и секунд и узнать общее количество секунд, составляющее введенный отрезок.

## Упражнение 25. Единицы времени (снова)

(*Решено. 24 строки*)

В данном упражнении мы развернем ситуацию из предыдущей задачи. На этот раз вам предстоит написать программу, в которой пользователь будет вводить временной промежуток в виде общего количества секунд, после чего на экране должна быть показана та же длительность в формате D:HH:MM:SS, где D, HH, MM и SS – это количество дней, часов, минут и секунд соответственно. При этом последние три значения должны быть выведены в формате из двух цифр, как мы привыкли видеть их на электронных часах. Попробуйте узнать сами, какие символы необходимо ввести в спецификатор формата, чтобы при необходимости числа дополнялись слева не пробелами, а нулями.

## *Упражнение 26. Текущее время*

(*10 строк*)

Модуль `time` в Python включает в себя несколько очень полезных функций для работы со временем. Одна из таких функций – `asctime` – считывает текущее системное время компьютера и возвращает его в удобном для восприятия виде. Используйте эту функцию для отображения на экране текущей даты и времени. Никакого ввода от пользователя на этот раз вам не потребуется.

## *Упражнение 27. Когда Пасха?*

(*33 строки*)

Пасха традиционно празднуется в воскресенье, следующее за первым полнолунием после дня весеннего равноденствия. Поскольку здесь есть зависимость от фазы луны, фиксированной даты для этого праздника в григорианском календаре не существует. Фактически Пасха может выпасть на любую дату между 22 марта и 25 апреля. День и месяц Пасхи для конкретного года можно вычислить по следующему алгоритму:

- в переменную *a* запишите остаток от деления *year* на 19;
- в переменную *b* запишите частное от деления *year* на 100 с округлением вниз;
- в переменную *c* запишите остаток от деления *year* на 100;
- в переменную *d* запишите частное от деления *b* на 4 с округлением вниз;
- в переменную *e* запишите остаток от деления *b* на 4;
- в переменную *f* запишите результат вычисления формулы $\frac{b+8}{25}$ с округлением вниз;
- в переменную *g* запишите результат вычисления формулы $\frac{b-f+1}{3}$ с округлением вниз;
- в переменную *h* запишите остаток от деления выражения $19a + b - d - g + 15$ на 30;
- в переменную *i* запишите частное от деления *c* на 4 с округлением вниз;
- в переменную *k* запишите остаток от деления *c* на 4;
- в переменную *l* запишите остаток от деления выражения $32 + 2e + 2i - h - k$ на 7;
- в переменную *m* запишите результат вычисления формулы

  $$\frac{a + 11h + 22l}{451}$$

  с округлением вниз;

- установите месяц равным результату вычисления формулы

$$\frac{h + l + 7m + 114}{31};$$

- установите день равным единице плюс остаток от деления выражения $h + l - 7m + 114$ на 31.

Напишите программу, реализующую этот алгоритм. Пользователь должен ввести год, для которого его интересует дата Пасхи, и получить ответ на свой вопрос.

## *Упражнение 28. Индекс массы тела*

(*14 строк*)

Напишите программу для расчета *индекса массы тела* (body mass index – BMI) человека. Пользователь должен ввести свой рост и вес, после чего вы используете одну из приведенных ниже формул для определения индекса. Если пользователь вводит рост в дюймах, а вес в фунтах, формула будет следующей:

$$\text{BMI} = \frac{\text{weight}}{\text{height}^2} \times 703.$$

Если же пользователь предпочитает вводить информацию о себе в сантиметрах и килограммах, формула упростится и станет такой:

$$\text{BMI} = \frac{\text{weight}}{\text{height}^2}.$$

## *Упражнение 29. Температура с учетом ветра*

(*Решено. 22 строки*)

Когда в прохладный день еще и дует ветер, температура кажется более низкой, чем есть на самом деле, поскольку движение воздушных масс способствует более быстрому охлаждению теплых предметов, к коим в данном случае можно отнести и человека. Этот эффект известен как охлаждение ветром.

В 2001 году Канада, Великобритания и США договорились об использовании общей формулы для определения коэффициента охлаждения ветром. В формуле, приведенной ниже, $T_a$ – это температура воздуха в градусах Цельсия, а $V$ – скорость ветра в километрах в час. Похожие формулы с другими константами могут использоваться для вычисления коэффициента охлаждения ветром для температур, указанных в градусах по Фаренгейту, и скорости ветра в милях в час или метрах в секунду.

$$\mathrm{WCI} = 13{,}12 + 0{,}6215T_a - 11{,}37V^{0{,}16} + 0{,}3965T_aV^{0{,}16}.$$

Напишите программу, которая будет запрашивать у пользователя температуру воздуха и скорость ветра и выдавать рассчитанный коэффициент охлаждения ветром с округлением до ближайшего целого.

**Примечание.** Принято считать, что коэффициент охлаждения ветром допустимо рассчитывать при температурах, меньших или равных 10 °C, и скорости ветра, превышающей 4,8 км/ч.

## *Упражнение 30. Цельсий в Фаренгейт и Кельвин*

(*17 строк*)

Напишите программу, которая будет запрашивать у пользователя значение температуры в градусах Цельсия и отображать эквивалентный показатель по шкалам Фаренгейта и Кельвина. Необходимые коэффициенты и формулы для проведения расчетов нетрудно найти на просторах интернета.

## *Упражнение 31. Единицы давления*

(*17 строк*)

В данном задании вам предстоит написать программу, которая будет переводить введенное пользователем значение давления в килопаскалях (кПа) в фунты на квадратный дюйм (PSI), миллиметры ртутного столба и атмосферы. Коэффициенты и формулы для перевода найдите самостоятельно.

## *Упражнение 32. Сумма цифр в числе*

(*18 строк*)

Разработайте программу, запрашивающую у пользователя целое четырехзначное число и подсчитывающую сумму составляющих его цифр. Например, если пользователь введет число 3141, программа должна вывести следующий результат: $3 + 1 + 4 + 1 = 9$.

## *Упражнение 33. Сортировка трех чисел*

(*Решено. 19 строк*)

Напишите программу, запрашивающую у пользователя три целых числа и выводящую их в упорядоченном виде – по возрастанию. Используйте функции `min` и `max` для нахождения наименьшего и наибольшего значений. Оставшееся число можно найти путем вычитания из суммы трех введенных чисел максимального и минимального.

## *Упражнение 34. Вчерашний хлеб*

(*Решено. 19 строк*)

Пекарня продает хлеб по $3,49 за буханку. Скидка на вчерашний хлеб составляет 60 %. Напишите программу, которая будет запрашивать у пользователя количество приобретенных вчерашних буханок хлеба. В вывод на экран должны быть включены обычная цена за буханку, цена со скидкой и общая стоимость приобретенного хлеба. Все значения должны быть выведены на отдельных строках с соответствующими описаниями. Используйте для вывода формат с двумя знаками после запятой и выровненным разделителем.

# Глава 2

# Принятие решений

Программы, с которыми вы работали до сих пор, выполнялись строго последовательно. Каждое последующее выражение выполнялось после завершения предыдущего, и все программы работали исключительно построчно сверху вниз. И хотя для простых задач, которые мы решали в первой части книги, такого линейного выполнения программ бывает вполне достаточно, для написания более сложных сценариев потребуется чуть больше.

Конструкции принятия решений позволяют программам при необходимости не выполнять определенные строки кода, тем самым производя ветвление. Выполнение программы по-прежнему происходит построчно сверху вниз, но некоторые строки могут быть просто пропущены. Это позволяет выполнять различные операции в зависимости от условий и серьезно повышает вариативность языка Python в отношении решения задач.

## 2.1. Выражения if

В программах, написанных на языке Python, ветвление осуществляется при помощи ключевого слова *if*. Выражение `if` включает в себя одно или несколько условий, а также тело выражения. При выполнении условного выражения происходит оценка заданного условия, на основании чего принимается решение, будут ли выполняться инструкции в теле выражения. Если результатом условного выражения будет `True` (Истина), то тело выполнится, после чего программа продолжится. Если же в результате проверки условия получится `False` (Ложь), тело будет пропущено, а выполнение программы продолжится с первой строки после тела.

Условия в выражении `if` могут быть достаточно сложными, а результат может принимать значение `True` или `False`. Такие выражения называются булевыми в честь Джорджа Буля (George Boole) (1815–1864) – пионера в области формальной логики.

Выражения `if` часто включают в себя *операторы отношения* (relational operator), сравнивающие значения, переменные или целые сложные вы-

ражения. Операторы отношения, присутствующие в языке Python, перечислены в табл. 2.1.

***Таблица 2.1. Операторы отношения в языке Python***

| Оператор отношения | Значение |
|---|---|
| < | Меньше |
| <= | Меньше или равно |
| > | Больше |
| >= | Больше или равно |
| == | Равно |
| != | Не равно |

Тело выражения `if` может включать в себя одну или несколько инструкций, которые пишутся с отступом от ключевого слова `if`. Блок тела заканчивается, когда отступ снова выравнивается относительно слова `if`. Вы можете сами выбрать, какую величину отступа использовать в своих программах при написании условных выражений. Во всех фрагментах кода в данной книге используются отступы в четыре пробела, но вы можете использовать один или несколько пробелов на свой вкус. Большинство программистов придерживаются одного формата при формировании отступов в теле условных выражений, но в Python такая последовательность в действиях вовсе не обязательна.

В следующем фрагменте кода у пользователя запрашивается число, после чего следуют два условных выражения со своими телами, отделенными от выражения `if` двоеточием.

```
# Запрашиваем значение у пользователя
num = float(input("Введите число: "))
# Сохраняем подходящее значение в переменной result
if num == 0:
    result = "Введен ноль"
if num != 0:
    result = "Введен не ноль"
# Отобразим результат
print(result)
```

## 2.2. Выражения if-else

В предыдущей программе в переменную `result` записывалось одно строковое значение, если пользователь ввел ноль, и другое, если ввел число, отличное от нуля. Чаще всего условные выражения строятся таким образом, что всегда будет выполняться какое-то одно условие из перечисленных. В таких конструкциях два условия одновременно выполняться не

могут, но при этом одно из них всегда будет выполнено. Подобные условия называются *взаимоисключающими* (mutually exclusive).

Выражение *if-else* состоит из части `if` и соответствующего ему тела, а также части *else* – без условия, но со своим телом. Во время запуска программы происходит проверка условия. Если его результат будет `True`, выполнится тело условного выражения `if`, а тело `else` будет пропущено. В противном случае все будет наоборот: тело выражения `if` будет пропущено, а тело `else` выполнится. Тела обоих выражений одновременно не могут быть выполнены, как и не могут быть пропущены. В таких случаях, когда условия являются взаимоисключающими, можно использовать выражение `if-else` вместо двух последовательных `if`. Применение конструкции `if-else` более предпочтительно, поскольку в этом случае необходимо будет написать только одно условие, при выполнении программы будет выполняться только оно, и к тому же в случае возникновения ошибки или изменения логики программы исправлять вам придется лишь одно условие. Наша предыдущая программа, переписанная под использование выражения `if-else`, будет выглядеть следующим образом:

```
# Запрашивает значение у пользователя
num = float(input("Введите число: "))
# Сохраняем подходящее значение в переменной result
if num == 0:
    result = "Введен ноль"
else:
    result = "Введен не ноль"
# Отобразим результат
print(result)
```

При вводе пользователем нуля условное выражение выдаст результат `True`, и в переменную `result` будет записано нужное выражение. В этом случае выражение `else` полностью пропускается. В обратной ситуации, когда пользователь ввел число, отличное от нуля, результатом условного выражения будет `False`, так что пропущено будет тело выражения `if`, а выполнится тело выражения `else`, и в переменную попадет строка о том, что пользователь ввел ненулевое значение. В обоих случаях Python продолжит выполнение с инструкции `print`, в результате чего ответ будет выведен на экран.

## 2.3. Выражения if-elif-else

Выражение `if-elif-else` используется в случае выбора одной из нескольких альтернатив. Выражение начинается с части `if`, за которой следует часть `elif`, а завершается конструкция блоком `else`. Все эти части выраже-

ния должны содержать свои тела с правильными отступами. Кроме того, все инструкции `if` и `elif` должны включать в себя условия, результатом вычисления которых может быть либо `True`, либо `False`.

При выполнении конструкции `if-elif-else` первой запускается проверка условия в блоке `if`. Если результатом будет `True`, выполнится тело этого блока, после чего будут пропущены все оставшиеся части выражения. В противном случае Python перейдет к анализу следующего условия – из блока `elif`. Если оно верно, будет, как и ожидается, выполнено тело этого блока с последующим пропуском остальных инструкций. Если же нет, Python продолжит проверять все условия до конца выражения. Если ни одно из них не даст истины, будет выполнено тело из блока `else`, после чего выполнение программы продолжится.

Давайте расширим наш предыдущий пример, чтобы выполнялась проверка не только на ноль, но также на положительные и отрицательные значения. И хотя возможно решить эту задачу посредством выражений `if` и/или `if-else`, лучше всего подобные сценарии реализовывать при помощи конструкции `if-elif-else`, поскольку одно из трех условий должно выполняться в любом случае.

```
# Запрашиваем значение у пользователя
num = float(input("Введите число: "))
# Сохраняем подходящее значение в переменной result
if num > 0:
    result = "Введено положительное число"
elif num < 0:
    result = "Введено отрицательное число"
else:
    result = "Введен ноль"
# Отобразим результат
print(result)
```

При вводе пользователем положительного значения первое же условие даст результат `True`, а значит, будет выполнено тело этого блока, после чего программа продолжит выполнение с инструкции `print`.

При вводе отрицательного значения первое условие даст результат `False`, а значит, начнутся проверки следующих условий по очереди. Второе условие окажется выполнено, в результате чего будет запущено тело этого блока, после чего программа продолжит выполнение с инструкции `print`.

Наконец, если пользователь ввел нулевое значение, все блоки `if` и `elif` дадут `False`, а значит, будет выполнено тело блока `else`. После этого будет осуществлен вывод результата на экран.

Мы видим, что в конструкции `if-elif-else` всегда будет выполняться только одно тело в зависимости от прописанных условий.

## 2.4. Выражения if-elif

Надо отметить, что блок `else` в конструкции `if-elif-else` является необязательным. Его присутствие гарантирует выполнение одного из блоков выражения в любом случае. Если этот блок отсутствует, это значит, что может не выполниться ни один из блоков. Это произойдет, если ни одно из условий в блоках не даст значения `True`. Вне зависимости от того, будет ли выполнено хоть одно тело в выражении `if-elif`, программа продолжит свое выполнение со следующей за последней частью `elif` инструкции.

## 2.5. Вложенные выражения if

Блоки `if`, `elif` и `else` могут включать в себя любые выражения, включая другие блоки `if`, `if-else`, `if-elif` и `if-elif-else`. Если одно условное выражение появляется в теле другого, оно называется *вложенным* (nested). Следующий фрагмент кода иллюстрирует эту концепцию:

```
# Запрашиваем у пользователя число
num = float(input("Введите число: "))
# Сохраняем нужный нам результат
if num > 0:
    # Определяем, какое прилагательное нам стоит использовать
    adjective = " "
    if num >= 1000000:
        adjective = " очень большое "
    elif num >= 1000:
        adjective = " большое "

    # Сохраняем в переменную положительные числа с правильным прилагательным
    result = "Это" + adjective + "положительное число"
elif num < 0:
    result = "Это отрицательное"
else:
    result = "Это ноль"

# Выводим результат
print(result)
```

Программа начинается с запроса числа у пользователя. Если он ввел число, большее нуля, выполнится тело внешнего блока `if`. В нем мы сначала инициализируем переменную `adjective` пробелом. Затем запускается вложенная конструкция `if-elif`. Если пользователь ввел значение, превышающее или равное миллиону, присвоим переменной `adjective` строку `" очень большое "`. Если введенное значение укладывается в интервал

от 1000 до 999 999, переменная `adjective` получит строковое значение `" большое "`. После этого сохраняем результат в переменную `result`. Блоки `elif` и `else` из внешней условной конструкции выполняться не будут, поскольку уже было выполнено тело блока `if`. Наконец, программа выводит результат на экран.

Если бы пользователь ввел отрицательное значение или ноль, выполнился бы второй или третий блок внешней конструкции `if-elif-else` соответственно. В результате в переменной `result` оказалось бы другое значение, которое и было бы выведено на экран.

## 2.6. Булева логика

*Булевым* называется выражение, возвращающее только `True` или `False`. Это могут быть как сами булевы значения `True` и `False`, так и переменные, хранящие булевы выражения, операторы отношения, а также вызовы функций, возвращающих булево выражение. Булевы выражения также могут включать *булевы операторы* (Boolean operator), комбинирующие булевы значения. В Python предусмотрены три булевых оператора: `not`, `and` и `or`.

Оператор `not` обращает значение булева выражения. Если переменная `x` содержит значение `True`, то выражение `not x` вернет `False`, и наоборот.

Поведение всех булевых выражений может быть сведено в *таблицы истинности* (truth table). Такие таблицы содержат один столбец для булева значения переменной, а во втором показан результат примененного к ней булева выражения. Каждая строка в таблице отражает уникальное сочетание значений `True` и `False` для переменных в выражении. Таблица истинности для выражения, содержащего `n` различных переменных, будет состоять из $2^n$ строк, в каждой из которых будет показан результат вычисления выражения для конкретных значений переменных. Например, таблица истинности для оператора `not`, примененного к одной переменной `x`, будет включать две строки ($2^1$), как показано в табл. 2.2.

***Таблица 2.2. Таблица истинности для оператора `not`***

| `x` | `not x` |
|---|---|
| `False` | `True` |
| `True` | `False` |

Операторы `and` и `or` применяются сразу к двум булевым значениям при определении результата. Булево выражение `x and y` даст `True`, если `x` равен `True` и `y` тоже равен `True`. Если `x` равен `False` или `y` равен `False`, а также если оба значения `x` и `y` равны `False`, выражение `x and y` даст на выходе `False`. Таблица истинности для оператора `and` показана в табл. 2.3. В ней $2^2 = 4$ строки, поскольку оператор `and` применяется к двум переменным.

***Таблица 2.3. Таблица истинности для оператора and***

| x | y | x and y |
|---|---|---|
| False | False | False |
| False | True | False |
| True | False | False |
| True | True | True |

Булево выражение `x or y` даст `True`, если `x` равен `True` или `y` равен `True`, а также если обе переменные `x` и `y` содержат `True`. Это выражение дает `False`, только если `x` и `y` содержат `False`. Таблица истинности для оператора `or` показана в табл. 2.4.

***Таблица 2.4. Таблица истинности для оператора or***

| x | y | x or y |
|---|---|---|
| False | False | False |
| False | True | True |
| True | False | True |
| True | True | True |

В следующем фрагменте кода на Python показано использование булева оператора `or` для определения того, входит ли значение, введенное пользователем, в число первых пяти простых чисел. Операторы `and` и `not` могут быть использованы в похожей манере при построении сложных логических цепочек.

```
# Запрашиваем у пользователя целое число
x = int(input("Введите целое число: "))
# Определяем, входит ли значение в число первых пяти простых чисел
if x == 2 or x == 3 or x == 5 or x == 7 or x == 11:
    print("Это одно из первых пяти простых чисел.")
else:
    print("Это не одно из первых пяти простых чисел.")
```

# 2.7. Упражнения

Представленные здесь упражнения должны быть выполнены с использованием условных конструкций `if`, `if-else`, `if-elif` и `if-elif-else`, включая концепции, которые были изложены в первой главе. При решении некоторых задач вам также может понадобиться встроить блок `if` в другой блок `if`.

## *Упражнение 35. Чет или нечет?*

(*Решено. 13 строк*)

Напишите программу, запрашивающую у пользователя целое число и выводящую на экран информацию о том, является введенное число четным или нечетным.

## *Упражнение 36. Собачий возраст*

(*22 строки*)

Считается, что один год, прожитый собакой, эквивалентен семи человеческим годам. При этом зачастую не учитывается, что собаки становятся абсолютно взрослыми уже к двум годам. Таким образом, многие предпочитают каждый из первых двух лет жизни собаки приравнивать к 10,5 года человеческой жизни, а все последующие – к четырем.

Напишите программу, которая будет переводить человеческий возраст в собачий с учетом указанной выше логики. Убедитесь, что программа корректно работает при пересчете возраста собаки меньше и больше двух лет. Также программа должна выводить сообщение об ошибке, если пользователь ввел отрицательное число.

## *Упражнение 37. Гласные и согласные*

(*Решено. 16 строк*)

Разработайте программу, запрашивающую у пользователя букву латинского алфавита. Если введенная буква входит в следующий список (`a`, `e`, `i`, `o` или `u`), необходимо вывести сообщение о том, что эта буква гласная. Если была введена буква `y`, программа должна написать, что эта буква может быть как гласной, так и согласной. Во всех других случаях должно выводиться сообщение о том, что введена согласная буква.

## *Упражнение 38. Угадайте фигуру*

(*Решено. 31 строка*)

Напишите программу, определяющую вид фигуры по количеству ее сторон. Запросите у пользователя количество сторон и выведите сообщение с указанием вида фигуры. Программа должна корректно обрабатывать и выводить названия для фигур с количеством сторон от трех до десяти включительно. Если введенное пользователем значение находится за границами этого диапазона, уведомите его об этом.

## Упражнение 39. Сколько дней в месяце?

(*Решено. 18 строк*)

Количество дней в месяце варьируется от 28 до 31. Очередная ваша программа должна запрашивать у пользователя название месяца и отображать количество дней в нем. Поскольку годы мы не учитываем, для февраля можно вывести сообщение о том, что этот месяц может состоять как из 28, так и из 29 дней, чтобы учесть фактор високосного года.

## Упражнение 40. Громкость звука

(*30 строк*)

В табл. 2.5 представлен уровень громкости в децибелах для некоторых распространенных источников шума.

***Таблица 2.5. Уровни громкости различных источников***

| Источник звука | Уровень громкости (дБ) |
|---|---|
| Отбойный молоток | 130 дБ |
| Газовая газонокосилка | 106 дБ |
| Будильник | 70 дБ |
| Тихая комната | 40 дБ |

Создайте программу, в которой пользователь будет вводить уровень шума в децибелах. Если введенное им значение будет в точности совпадать с одним из значений в приведенной таблице, необходимо вывести, чему соответствует указанный уровень громкости. Если значение попадет между уровнями в таблице, нужно сообщить, между какими именно. Также программа должна выдавать корректные сообщения, в случае если введенное пользователем значение окажется ниже минимального или больше максимального.

## Упражнение 41. Классификация треугольников

(*Решено. 21 строка*)

Все треугольники могут быть отнесены к тому или иному классу (равнобедренные, равносторонние и разносторонние) на основании длин их сторон. Равносторонний треугольник характеризуется одинаковой длиной всех трех сторон, равнобедренный – двух сторон из трех, а у разностороннего треугольника все стороны разной длины.

Напишите программу, которая будет запрашивать у пользователя длины всех трех сторон треугольника и выдавать сообщение о том, к какому типу следует его относить.

## Упражнение 42. Узнать частоту по ноте

(*Решено. 39 строк*)

В табл. 2.6 перечислены частоты звуков, относящихся к одной октаве, начиная с до.

***Таблица 2.6. Частоты нот одной октавы***

| Нота | Частота (Гц) |
|---|---|
| C4 | 261,63 |
| D4 | 293,66 |
| E4 | 329,63 |
| F4 | 349,23 |
| G4 | 392,00 |
| A4 | 440,00 |
| B4 | 493,88 |

Пусть ваша программа запрашивает у пользователя обозначение ноты и показывает ее частоту согласно приведенной таблице. После этого вы можете доработать свою программу таким образом, чтобы она поддерживала все октавы, начиная от субконтроктавы (C0) до пятой октавы (C8). И хотя можно это реализовать путем добавления бесконечного количества блоков `if`, это будет довольно громоздким, недостаточно элегантным и просто неприемлемым решением данной задачи. Вместо этого при расчетах лучше использовать отношения между одними и теми же нотами в соседствующих октавах. К примеру, частота любой ноты октавы $n$ будет составлять ровно половину от частоты той же ноты октавы $n + 1$. Используя это соотношение, вы без труда сможете добавить в свою программу учет всех нот любой октавы без присутствия бесчисленных условных блоков.

**Подсказка.** Пользователь должен вводить ноту вместе с номером нотации октавы. Начните с разделения буквы, обозначающей ноту, и цифры, соответствующей номеру октавы. Затем определите частоту введенной ноты по представленной выше таблице и разделите ее на $2^{4-x}$, где $x$ – номер октавы в научной нотации, введенный пользователем. Это позволит умножить или разделить на два число из таблицы нужное количество раз.

## Упражнение 43. Узнать ноту по частоте

(*Решено. 42 строки*)

В предыдущем упражнении мы определяли частоту ноты по ее названию и номеру октавы. Теперь выполним обратную операцию. Запроси-

те у пользователя частоту звука. Если введенное значение укладывается в диапазон ±1 Гц от значения в таблице, выведите на экран название соответствующей ноты. В противном случае сообщите пользователю, что ноты, соответствующей введенной частоте, не существует. В данном упражнении можно ограничиться только нотами, приведенными в таблице. Нет необходимости брать в расчет другие октавы.

## *Упражнение 44. Портреты на банкнотах*

(*31 строка*)

Во многих странах существует традиция помещать портреты своих бывших политических лидеров или других выдающихся личностей на банкноты. В табл. 2.7 приведены номиналы банкнот США с изображенными на них людьми.

***Таблица 2.7. Банкноты США***

| Портрет | Номинал банкноты |
|---|---|
| Джордж Вашингтон | $1 |
| Томас Джефферсон | $2 |
| Авраам Линкольн | $5 |
| Александр Гамильтон | $10 |
| Эндрю Джексон | $20 |
| Улисс Грант | $50 |
| Бенджамин Франклин | $100 |

Напишите программу, которая будет запрашивать у пользователя номинал банкноты и отображать на экране имя деятеля, портрет которого размещен на соответствующем денежном знаке. Если банкноты введенного номинала не существует, должно выводиться сообщение об ошибке.

**Примечание.** Хотя банкноты номиналом два доллара очень редко можно встретить в США, официально они существуют и могут быть использованы при любых расчетах. Также в начале прошлого века в Америке были выпущены в обращение банкноты номиналом $500, $1000, $5000 и $10 000, хотя с 1945 года они не печатались, а в 1969 году и вовсе были выведены из обращения. Так что их мы не будем рассматривать в данном упражнении.

## *Упражнение 45. Даты праздников*

(*18 строк*)

В Канаде есть три национальных праздника, отмечающихся в одни и те же даты каждый год. Они приведены в табл. 2.8.

***Таблица 2.8. Канадские праздники***

| Праздник | Дата |
|---|---|
| Новый год | 1 января |
| День Канады | 1 июля |
| Рождество | 25 декабря |

Напишите программу, которая будет запрашивать у пользователя день и месяц. Если введенные данные в точности указывают на один из перечисленных в таблице праздников, необходимо вывести его название. В противном случае сообщить, что на заданную дату праздники не приходятся.

**Примечание.** В Канаде есть два дополнительных национальных праздника: Страстная пятница и Праздник трудящихся, – которые выпадают на разные даты каждый год. Есть свои праздники и в отдельных провинциях – некоторые с фиксированными датами, другие – с плавающими. Мы не будем рассматривать такие праздники в этом упражнении.

## *Упражнение 46. Какого цвета клетка?*

(*22 строки*)

Клетки на шахматной доске идентифицируются буквой и цифрой. Буква определяет положение клетки по горизонтали, а цифра – по вертикали, как показано на рис. 2.1.

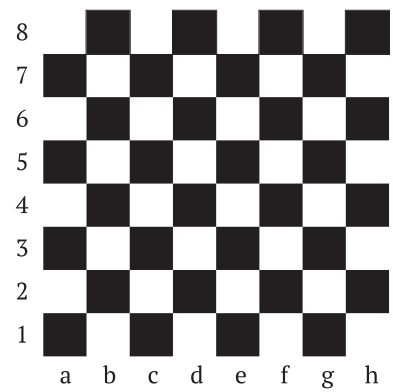



**Рис. 2.1** ❖ Поля на шахматной доске

Ваша программа должна запрашивать у пользователя координаты клетки. Используйте условное выражение для определения того, с какой клет-

ки – белой или черной – начинается столбец. Затем при помощи обычной арифметики необходимо определить цвет конкретной клетки. Например, если пользователь ввел `a1`, программа должна определить, что клетка с этими координатами черная. Если `d5` – белая. Проверку на ошибочность ввода координат клетки выполнять не нужно.

## *Упражнение 47. Определение времени года*

(*Решено. 43 строки*)

Год делится на четыре сезона: зима, весна, лето и осень. Хотя даты смены сезонов каждый год могут меняться из-за особенностей календаря, мы в данном упражнении примем допущения, перечисленные в табл. 2.9.

***Таблица 2.9. Даты смены сезонов***

| Сезон | Первый день |
|---|---|
| Весна | 20 марта |
| Лето | 21 июня |
| Осень | 22 сентября |
| Зима | 21 декабря |

Разработайте программу, запрашивающую у пользователя день и месяц – сначала месяц в текстовом варианте, затем номер дня. На выходе программа должна выдать название сезона, которому принадлежит выбранная дата.

## *Упражнение 48. Знаки зодиака*

(*47 строк*)

В гороскопах, наполняющих газеты и журналы, астрологи пытаются положение солнца в момент рождения человека как-то связать с его судьбой. Всего насчитывается 12 знаков зодиака, и все они приведены в табл. 2.10.

***Таблица 2.10. Знаки зодиака***

| Знак зодиака | Даты | Знак зодиака | Даты |
|---|---|---|---|
| Козерог | 22 декабря – 19 января | Рак | 21 июня – 22 июля |
| Водолей | 20 января – 18 февраля | Лев | 23 июля – 22 августа |
| Рыбы | 19 февраля – 20 марта | Дева | 23 августа – 22 сентября |
| Овен | 21 марта – 19 апреля | Весы | 23 сентября – 22 октября |
| Телец | 20 апреля – 20 мая | Скорпион | 23 октября – 21 ноября |
| Близнецы | 21 мая – 20 июня | Стрелец | 22 ноября – 21 декабря |

Напишите программу, запрашивающую у пользователя дату его рождения и выводящую на экран соответствующий знак зодиака.

## *Упражнение 49. Китайский гороскоп*

(*Решено. 40 строк*)

Китайский гороскоп делит время на 12-летние циклы, и каждому году соответствует конкретное животное. Один из таких циклов приведен в табл. 2.11. После окончания одного цикла начинается другой, то есть 2012 год снова символизирует дракона.

***Таблица 2.11. Китайский гороскоп***

| Год | Животное | Год | Животное |
|---|---|---|---|
| 2000 | Дракон | 2006 | Собака |
| 2001 | Змея | 2007 | Свинья |
| 2002 | Лошадь | 2008 | Крыса |
| 2003 | Коза | 2009 | Бык |
| 2004 | Обезьяна | 2010 | Тигр |
| 2005 | Петух | 2011 | Кролик |

Напишите программу, которая будет запрашивать у пользователя год рождения и выводить ассоциированное с ним название животного по китайскому гороскопу. При этом программа не должна ограничиваться только годами из приведенной таблицы, а должна корректно обрабатывать все годы нашей эры.

## *Упражнение 50. Шкала Рихтера*

(*30 строк*)

В табл. 2.12 приведены диапазоны магнитуд землетрясений по шкале Рихтера с описаниями.

***Таблица 2.12. Шкала Рихтера***

| Магнитуда | Описание землетрясения |
|---|---|
| Меньше 2,0 | Минимальное |
| Больше или равно 2,0 и меньше 3,0 | Очень слабое |
| Больше или равно 3,0 и меньше 4,0 | Слабое |
| Больше или равно 4,0 и меньше 5,0 | Промежуточное |
| Больше или равно 5,0 и меньше 6,0 | Умеренное |
| Больше или равно 6,0 и меньше 7,0 | Сильное |
| Больше или равно 7,0 и меньше 8,0 | Очень сильное |
| Больше или равно 8,0 и меньше 10,0 | Огромное |
| 10.0 и больше | Разрушительное |

Ваша программа должна запрашивать у пользователя магнитуду землетрясения по шкале Рихтера и выводить на экран описание уровня, соот-

ветствующего введенному значению. Например, если пользователь введет значение 5,5, нужно вывести сообщение о том, что этой магнитуде соответствует умеренный уровень землетрясения.

## Упражнение 51. Корни квадратичной функции

(*24 строки*)

Общий вид квадратичной функции одной переменной имеет следующий вид: $f(x) = ax^2 + bx + c$, где $a$, $b$ и $c$ – константы и $a$ не равна нулю. Корни этой функции могут быть извлечены путем нахождения таких значений переменной $x$, для которых будет соблюдаться равенство $ax^2 + bx + c = 0$. Эти значения могут быть вычислены с помощью формулы для корней квадратного уравнения, показанной ниже. Квадратичная функция может иметь от нуля до двух действительных корней.

$$\text{root} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

Часть выражения под квадратным корнем называется дискриминантом. Если дискриминант отрицательный, квадратное уравнение не будет иметь действительных корней. В случае равенства дискриминанта нулю у квадратного уравнения будет ровно один действительный корень. Иначе корней будет два, и выражение необходимо будет вычислить дважды: один раз со знаком плюс, второй со знаком минус для числителя.

Напишите программу, вычисляющую действительные корни квадратичной функции. Сначала вы должны запросить у пользователя значения $a$, $b$ и $c$. После этого должно быть выведено на экран количество действительных корней функции и их значения.

## Упражнение 52. Буквенные оценки – в числовые

(*Решено. 52 строки*)

В разных странах успеваемость студентов в университетах ведется поразному: где-то в качестве оценок используются буквы, где-то цифры. Соответствие между ними приведено в табл. 2.13.

***Таблица 2.13. Оценка успеваемости***

| Буквенная оценка | Числовая оценка | Буквенная оценка | Числовая оценка |
|---|---|---|---|
| A+ | 4,0 | C+ | 2,3 |
| A | 4,0 | C | 2,0 |
| A– | 3,7 | C– | 1,7 |
| B+ | 3,3 | D+ | 1,3 |
| B | 3,0 | D | 1,0 |
| B– | 2,7 | F | 0 |

Напишите программу, которая будет принимать на вход буквенную оценку и выводить на экран соответствующую оценку в числовом выражении. Убедитесь в том, что программа генерирует понятное сообщение об ошибке при неверном вводе.

## *Упражнение 53. Числовые оценки – в буквенные*

(*47 строк*)

В предыдущем упражнении мы переводили буквенные оценки студентов в числовые. Сейчас перевернем ситуацию и попробуем определить буквенный номинал оценки по его числовому эквиваленту. Убедитесь в том, что ваша программа будет обрабатывать числовые значения между указанными в табл. 2.13. В этом случае оценки должны быть округлены до ближайшей буквы. Программа должна выдавать оценку A+, если введенное пользователем значение будет 4,0 и выше.

## *Упражнение 54. Оценка работы*

(*Решено. 30 строк*)

Представьте, что в компании проводится аттестация сотрудников в конце каждого года. Шкала рейтинга начинается на отметке 0,0, и чем лучше оценка, тем выше руководство оценивает сотрудника, а значит, тем больше будет его прибавка к зарплате. Рейтинг, присваиваемый сотрудникам, может составлять значения 0,0, 0,4 или 0,6 и выше. Значения между 0,0 и 0,4, а также между 0,4 и 0,6 никогда не используются. Значения, ассоциированные с каждым рейтингом, показаны в табл. 2.14. Прибавка к зарплате сотрудника рассчитывается как рейтинг, умноженный на $2400,00.

***Таблица 2.14. Таблица рейтингов***

| Рейтинг | Значение |
|---|---|
| 0,0 | Низкий уровень |
| 0,4 | Удовлетворительный уровень |
| 0,6 и выше | Высокий уровень |

Напишите программу, которая будет запрашивать у пользователя рейтинг сотрудника и выводить соответствующее значение из приведенной таблицы. Также необходимо показать сумму прибавки сотрудника. При вводе некорректного значения рейтинга программа должна об этом сообщать.

## *Упражнение 55. Длины волн видимой части спектра*

(*38 строк*)

Длины волн видимой части спектра колеблются от 380 до 750 нанометров (нм). И хотя сам спектр является непрерывным, его принято делить на шесть цветов, как показано в табл. 2.15.

***Таблица 2.15. Длины волн по цветам***

| Цвет | Длина волны (нм) |
|---|---|
| Фиолетовый | Больше или равно 380 и меньше 450 |
| Синий | Больше или равно 450 и меньше 495 |
| Зеленый | Больше или равно 495 и меньше 570 |
| Желтый | Больше или равно 570 и меньше 590 |
| Оранжевый | Больше или равно 590 и меньше 620 |
| Красный | Больше или равно 620 и меньше или равно 750 |

Запросите у пользователя длину волны и выведите на экран соответствующий ей цвет. Если введенное пользователем значение длины волны окажется за пределами видимой части спектра, сообщите об этом.

## *Упражнение 56. Определение частоты*

(*31 строка*)

Электромагнитные волны можно классифицировать по частоте на семь категорий, как показано в табл. 2.16.

***Таблица 2.16. Частоты электромагнитных волн***

| Категория | Диапазон частот (Гц) |
|---|---|
| Радиоволны | Менее $3\times10^{9}$ |
| Микроволны | От $3\times10^{9}$ до $3\times10^{12}$ |
| Инфракрасное излучение | От $3\times10^{12}$ до $4{,}3\times10^{14}$ |
| Видимое излучение | От $4{,}3\times10^{14}$ до $7{,}5\times10^{14}$ |
| Ультрафиолетовое излучение | От $7{,}5\times10^{14}$ до $3\times10^{17}$ |
| Рентгеновское излучение | От $3\times10^{17}$ до $3\times10^{19}$ |
| Гамма-излучение | Более $3\times10^{19}$ |

Напишите программу, которая будет запрашивать у пользователя значение частоты волны и отображать название соответствующего излучения.

## *Упражнение 57. Счет за телефон*

(*44 строки*)

Тарифный план мобильной связи включает в себя 50 минут разговоров и 50 смс-сообщений за $15,00 в месяц. Каждая дополнительная минута стоит $0,25, а каждое дополнительное сообщение – $0,15. Все счета за телефон включают налог на поддержку кол-центров 911 в размере $0,44, и общая сумма, включающая сумму отчислений кол-центрам, облагается налогом в размере 5 %.

Напишите программу, которая будет запрашивать у пользователя количество израсходованных за месяц минут разговора и смс-сообщений

и отображать базовую сумму тарификации, сумму за дополнительные минуты и сообщения, сумму отчислений кол-центрам 911, налог, а также итоговую сумму к оплате. При этом дополнительные звонки и сообщения необходимо выводить на экран только в случае их расходования. Убедитесь в том, что все суммы отображаются в формате с двумя знаками после запятой.

## *Упражнение 58. Високосный год?*

(*Решено. 22 строки*)

В большинстве случаев год насчитывает 365 дней. Но на самом деле нашей планете требуется чуть больше времени, чтобы полностью пройти по своей орбите вокруг Солнца. В результате для компенсации этой разницы был введен дополнительный день в феврале для особых годов, называемых високосными. Определить, високосный год или нет, можно, следуя такому алгоритму:

- если год делится на 400 без остатка, он високосный;
- если год (из оставшихся) делится на 100 без остатка, он НЕ високосный;
- если год (из оставшихся) делится на 4 без остатка, он високосный;
- все остальные года не являются високосными.

Напишите программу, запрашивающую год у пользователя и выводящую сообщение о том, високосный ли он.

## *Упражнение 59. Следующий день*

(*50 строк*)

Разработайте программу, принимающую на вход дату и выводящую на экран дату, следующую за ней. Например, если пользователь введет дату, соответствующую 18 ноября 2019 года, на экран должен быть выведен следующий день, то есть 19 ноября 2019 года. Если входная дата будет представлять 30 ноября, то на выходе мы должны получить 1 декабря. И наконец, если ввести последний день года – 31 декабря 2019-го, пользователь должен увидеть на экране дату 1 января 2020-го. Дату пользователь должен вводить в три этапа: год, месяц и день. Убедитесь, что ваша программа корректно обрабатывает високосные годы.

## *Упражнение 60. На какой день недели выпадает 1 января?*

(*32 строки*)

Следующая формула может быть использована для определения дня недели, соответствующего 1 января заданного года:

$$\text{day\_of\_the\_week} = (\text{year} + \text{floor}((\text{year} - 1)/4) - \text{floor}((\text{year} - 1)/100) + \text{floor}((\text{year} - 1)/400))\%7.$$

В результате мы получим целое число, представляющее день недели от воскресенья (0) до субботы (6).

Используйте эту формулу для написания программы, запрашивающей у пользователя год и выводящей на экран день недели, на который в заданном году приходится 1 января. При этом на экран вы должны вывести не числовой эквивалент дня недели, а его полное название.

## *Упражнение 61. Действительный номерной знак машины?*

(*Решено. 28 строк*)

Допустим, в нашей стране старый формат номерных знаков автомобилей состоял из трех заглавных букв, следом за которыми шли три цифры. После того как все возможные номера были использованы, формат был изменен на четыре цифры, предшествующие трем заглавным буквам.

Напишите программу, запрашивающую у пользователя номерной знак машины и оповещающую о том, для какого формата подходит данная последовательность символов: для старого или нового. Если введенная последовательность не соответствует ни одному из двух форматов, укажите это в сообщении.

## *Упражнение 62. Играем в рулетку*

(*Решено. 45 строк*)

На игровой рулетке в казино 38 выемок для шарика: 18 красных, столько же черных и две зеленые, пронумерованные 0 и 00 соответственно. Красные числа на рулетке: 1, 3, 5, 7, 9, 12, 14, 16, 18, 19, 21, 23, 25, 27, 30 32, 34 и 36. Остальные номера в диапазоне положительных чисел до 36 – черные.

Игрок может сделать самые разные ставки. Мы в данном упражнении будем рассматривать лишь следующие виды ставок:

- одно число (от одного до 36, а также 0 или 00);
- красное или черное;
- четное или нечетное (заметьте, что номера 0 и 00 **не** являются ни четными, ни нечетными);
- от 1 до 18 против от 19 до 36.

Напишите программу, имитирующую игру на рулетке при помощи генератора случайных чисел в Python. После запуска рулетки выведите на экран выпавший номер и все сыгравшие ставки. Например, при выпадении номера 13 на экране должна появиться следующая последовательность строк:

```
Выпавший номер: 13…
Выигравшая ставка: 13
Выигравшая ставка: черное
Выигравшая ставка: нечетное
Выигравшая ставка: от 1 до 18
```

Если на рулетке выпал номер 0, вывод должен быть следующим:

```
Выпавший номер: 0…
Выигравшая ставка: 0
```

Для номера 00 сообщения будут иметь следующий вид:

```
Выпавший номер: 00…
Выигравшая ставка: 00
```

# Глава 3

# Повторения

Как написать программу, в которой одна и та же операция должна выполняться несколько раз? Вы, конечно, можете скопировать фрагмент кода и вставить его в программу столько раз, сколько необходимо, но такое решение будет крайне далеким от идеала. Минусами подобного подхода является то, что вы сможете повторить необходимое действие строго определенное количество раз, а при отладке кода нужно будет вносить правки во все скопированные участки.

В Python для организации циклических действий предусмотрен необходимый арсенал инструкций. Циклы позволяют требуемое количество раз запустить один и тот же фрагмент кода во время выполнения программы. При должном использовании циклические конструкции позволяют значительно повысить эффективность программного кода при минимуме затрат на его разработку.

## 3.1. Циклы while

Цикл *while* дает возможность выполнять набор инструкций сколь угодно долго – до тех пор, пока соответствующее условное выражение возвращает `True`. Как и `if`, выражение `while` обладает своим условием и телом, написанным с отступами. Если условное выражение инструкции `while` возвращает `True`, тело выполняется, после чего снова производится проверка условия. Если оно по-прежнему возвращает истину, тело выполняется вновь, и т. д. Таким образом, тело цикла будет выполняться до тех пор, пока условное выражение не вернет значение `False`. Как только это произошло, тело цикла будет пропущено, и выполнение программы возобновится с первой инструкции, следующей после тела `while`.

Часто в условных выражениях инструкции `while` выполняется сравнение введенного пользователем значения с содержимым другой переменной, в результате чего у пользователя есть возможность ввести ожидаемое значение для выхода из цикла. В этом случае условное выражение инструкции `while` должно вернуть значение `False`, после чего произойдет

пропуск тела цикла и переход к следующей за ним инструкции. Например, в следующем фрагменте кода пользователь будет вводить числа и получать сообщения о том, положительное или отрицательное число он ввел, до тех пор, пока не введет ноль, говорящий программе о необходимости покинуть цикл.

```
# Запрашиваем ввод у пользователя
x = int(input("Введите целое число (0 для выхода): "))
# Запускаем цикл, пока пользователь не введет ноль
while x != 0:
    # Положительное или отрицательное?
    if x > 0:
        print("Это положительное число.")
    else:
        print("Это отрицательное число.")
    # Запрашиваем очередное значение
    x = int(input("Введите целое число (0 для выхода): "))
```

Программа начинается с запроса у пользователя целого числа. Если он введет ноль, условное выражение `x != 0` вернет `False`, и тело цикла будет просто проигнорировано, а программа завершит выполнение без всякого вывода, если не считать запроса на ввод числа. Если же условная конструкция вернет `True`, будет выполнено тело цикла.

В теле цикла `while` введенное пользователем значение сравнивается с нулем при помощи `if`, и на экран выводится соответствующий текст. В конце тела цикла пользователь получает еще одну просьбу ввести число. Поскольку после этого тело цикла заканчивается, будет осуществлен возврат к условному выражению `while` с очередной проверкой ввода. Если на этот раз пользователь ввел ноль, программа будет завершена. В противном случае мы снова войдем в тело цикла. Это будет продолжаться до тех пор, пока пользователь не введет ноль.

## 3.2. Циклы for

Как и `while`, циклы *for* служат для повторного запуска одного и того же блока кода при выполнении определенных условий. При этом механизм проверки условий в цикле `for` значительно отличается.

Тело цикла `for` будет выполняться по разу для каждого элемента коллекции. При этом коллекция может представлять собой диапазон целых чисел, буквы в строке или, как мы увидим позже в этой книге, значения, сохраненные в виде специальных структур вроде списков. Синтаксис инструкции `for` представлен ниже, текст во вставках `<variable>`, `<collection>` и `<body>` необходимо заменить на нужный.

```
for <variable> in <collection>:
    <body>
```

Тело цикла `for`, как и в случае с `while`, может состоять из одной или нескольких инструкций на языке Python, отделенных отступами, и выполняться множество раз.

Перед каждым запуском тела цикла очередной элемент из коллекции копируется в `<variable>`. Эта переменная создается для цикла `for` при его запуске. При этом совсем не обязательно создавать данную переменную при помощи оператора присваивания, а любое значение, которое могло быть записано в ней, будет удалено перед запуском каждой итерации. Переменная может быть использована в теле цикла так же точно, как и любая другая переменная в языке Python.

Коллекция из целых чисел может быть создана при помощи удобной функции Python *range*. С единственным переданным аргументом она вернет диапазон значений, начинающийся с нуля и заканчивающийся переданным аргументом, не включая его самого. Например, функция `range(4)` сгенерирует диапазон из чисел 0, 1, 2 и 3.

При передаче двух аргументов функция `range` вернет диапазон, начинающийся с первого переданного аргумента и заканчивающийся вторым, не включая его. Так, вызов функции `range(4, 7)` даст на выходе диапазон из целых чисел 4, 5 и 6. Если первый аргумент будет больше или равен второму, функция `range` вернет пустой диапазон. В этом случае тело цикла не будет выполнено ни разу, а программа продолжит выполняться с первой инструкции после тела цикла.

Функция `range` может также вызываться с третьим аргументом, отвечающим за шаг итерации при перемещении от начального значения цикла к конечному. Если шаг будет больше нуля, мы получим диапазон, начинающийся со значения, соответствующего первому аргументу, в котором каждый последующий элемент будет отстоять от предыдущего на шаг, заданный третьим аргументом. Отрицательное значение шага позволяет сконструировать нисходящий диапазон значений. К примеру, если вызов функции `range(0, -4)` вернет пустой диапазон, то вариант с третьим аргументом `range(0, -4, -1)` приведет к созданию диапазона, состоящего из чисел 0, −1, −2 и −3. Заметьте, что третий аргумент должен представлять собой целочисленное значение. При необходимости задать дробный шаг лучше будет использовать цикл `while`.

В следующем примере используется цикл `for` совместно с функцией `range` для вывода на экран всех значений, кратных трем, от нуля и до введенного пользователем числа, включая его.

```
# Запрашиваем у пользователя верхнюю границу
limit = int(input("Введите целое число: "))
# Выводим все числа, кратные трем, вплоть до указанного пользователем значения
```

```
print("Все числа, кратные трем, вплоть до", limit, ":")
for i in range(3, limit + 1, 3):
    print(i)
```

При запуске программа попросит пользователя ввести целое число. Предположим, он ввел число 11. После этого на экран будет выведено сообщение, описывающее будущий диапазон, и начнется выполнение цикла.

Диапазон, построенный функцией `range`, будет начинаться с трех и продолжаться до $11 + 1 = 12$ с шагом 3. Таким образом, в него войдут числа 3, 6 и 9. При первом запуске цикла переменной *i* будет присвоено значение 3, которое будет выведено на экран в теле цикла. После этого переменная *i* последовательно получит значения 6 и 9 с выводом их на экран. Следом за девяткой цикл прекратит свою работу за неимением элементов, и выполнение программы продолжится со следующей за телом цикла строки. В нашем случае таковой нет, а значит, программа просто завершит выполнение.

## 3.3. Вложенные циклы

Блок выражений в теле цикла может включать в себя другой цикл, который будет именоваться *вложенным* (nested). При этом в цикл могут быть вложены любые типы циклов. В следующем фрагменте кода цикл `for` вложен в тело цикла `while` для повторения фразы, введенной пользователем, определенное количество раз, пока им не будет введена пустая строка.

```
# Запрашиваем у пользователя сообщение
message = input("Введите сообщение (оставьте его пустым для выхода): ")
# Начало цикла, пока сообщение не станет пустым
while message != "":
    # Запрашиваем количество повторений
    n = int(input("Сколько раз повторить сообщение? "))
    # Показываем сообщение заданное количество раз
    for i in range(n):
        print(message)
    # Запрашиваем следующее сообщение
    message = input("Введите сообщение (оставьте его пустым для выхода): ")
```

Сначала программа запрашивает у пользователя сообщение для вывода на экран. Если сообщение будет непустым, запустится тело цикла `while`, в котором пользователю будет задан новый вопрос о том, сколько раз вывести на экран введенное им сообщение. После этого создается диапазон от нуля до введенного пользователем числа, не включая его, и тело цикла `for` выполняется столько раз, сколько чисел оказалось в диапазоне.

По окончании выполнения цикла `for` пользователю снова предлагается ввести сообщение, и программа переходит к началу внешнего цикла `while`, где вновь выполняется проверка того, что сообщение введено. Если это так, пользователь снова может ввести количество желаемых повторений своего сообщения, и так до момента, пока не оставит вопрос о вводе сообщения без ответа. После этого тело цикла `while` будет пропущено, и программа завершит свое выполнение.

## 3.4. Упражнения

Следующие упражнения должны быть выполнены при помощи циклов. В некоторых из них будет указано, какой именно тип цикла необходимо использовать. В противном случае вы вольны сами выбирать тип применяемого цикла. Некоторые задачи могут быть решены как с применением цикла `for`, так и с использованием `while`. Есть в этом списке и упражнения на применение множественных циклов, которые должны быть вложены друг в друга. Тщательно выбирайте способ решения задачи в каждом отдельном случае.

### *Упражнение 63. Среднее значение*

(*26 строк*)

В данном упражнении вы должны написать программу для подсчета среднего значения всех введенных пользователем чисел. Индикатором окончания ввода будет служить ноль. При этом программа должна выдавать соответствующее сообщение об ошибке, если первым же введенным пользователем значением будет ноль.

**Подсказка.** Поскольку ноль является индикатором окончания ввода, его не нужно учитывать при расчете среднего.

### *Упражнение 64. Таблица со скидками*

(*18 строк*)

В магазине была объявлена скидка размером 60 % на ряд товаров, и для того чтобы покупатели лучше ориентировались, владелец торговой точки решил вывесить отдельную таблицу со скидками с указанием уцененных товаров и их оригинальных цен. Используйте цикл для создания подобной таблицы, в которой будут исходные цены, суммы скидок и новые цены для покупок на сумму $4,95, $9,95, $14,95, $19,95 и $24,95. Убедитесь в том, что суммы скидки и новые цены отображаются с двумя знаками после запятой.

## *Упражнение 65. Таблица соотношения температур*

(*22 строки*)

Напишите программу для вывода таблицы соотношения температур, выраженных в градусах Цельсия и Фаренгейта. В таблице должны размещаться все температуры между 0 и 100 градусами Цельсия, кратные 10. Дополните таблицу подходящими заголовками. Формулу для перевода температуры из градусов Цельсия в градусы Фаренгейта можно легко найти на просторах интернета.

## *Упражнение 66. Никаких центов*

(*Решено. 39 строк*)

4 февраля 2013 года Королевским канадским монетным двором была выпущена последняя монета номиналом в один цент. После вывода центов из обращения все магазины вынуждены были изменить цены на товары таким образом, чтобы они стали кратны пяти центам (расчеты по банковским картам по-прежнему ведутся с учетом центов). И хотя продавцы вольны сами определять политику преобразования цен, большинство из них просто округлили цены до ближайших пяти центов.

Напишите программу, запрашивающую у пользователя цены, пока не будет введена пустая строка. На первой строке выведите сумму всех введенных пользователем сумм, а на второй – сумму, которую покупатель должен заплатить наличными. Эта сумма должна быть округлена до ближайших пяти центов. Вычислить сумму для оплаты наличными можно, получив остаток от деления общей суммы в центах на 5. Если он будет меньше 2,5, следует округлить сумму вниз, а если больше – вверх.

## *Упражнение 67. Найти периметр многоугольника*

(*Решено. 42 строки*)

Напишите программу для расчета периметра заданного многоугольника. Начните с запроса у пользователя координат *x* и *y* первой точки многоугольника. Продолжайте запрашивать координаты следующих точек фигуры, пока пользователь не оставит строку ввода координаты по оси *x* пустой. После ввода каждой пары значений вы должны вычислить длину очередной стороны многоугольника и прибавить полученное значение к будущему ответу. По окончании ввода необходимо вычислить расстояние от последней введенной точки до первой, чтобы замкнуть фигуру, и вывести итоговый результат. Пример ввода координат точек многоугольника и вывода периметра показан ниже. Введенные пользователем значения выделены жирным.

```
Введите первую координату X: 0
Введите первую координату Y: 0
```

```
Введите следующую координату X (Enter для окончания ввода): 1
Введите следующую координату Y: 0
Введите следующую координату X (Enter для окончания ввода): 0
Введите следующую координату Y: 1
Введите следующую координату X (Enter для окончания ввода):
Периметр многоугольника равен 3.414213562373095
```

## Упражнение 68. Средняя оценка

(*62 строки*)

В задаче 52 мы уже создавали таблицу соответствий между оценками в буквенном и числовом выражении. Сейчас вам нужно будет рассчитать среднюю оценку по произвольному количеству введенных пользователем буквенных оценок. Для окончания ввода можно использовать индикатор в виде пустой строки. Например, если пользователь последовательно введет оценки A, затем C+, а после этого B и пустую строку, средний результат должен составить 3,1.

Для расчетов вам может пригодиться математика из упражнения 52. Никаких проверок на ошибки проводить не нужно. Предположим, что пользователь может вводить только корректные оценки или ноль.

## Упражнение 69. Билеты в зоопарк

(*Решено. 38 строк*)

В зоопарке цена входного билета зависит от возраста посетителя. Дети до двух лет допускаются бесплатно. Дети в возрасте от трех до 12 лет могут посещать зоопарк за $14,00. Пенсионерам старше 65 лет вход обойдется в $18,00, а обычный взрослый билет стоит $23,00.

Напишите программу, которая будет запрашивать возраст всех посетителей в группе (по одному за раз) и выводить общую цену билетов для посещения зоопарка этой группой. В качестве индикатора окончания ввода можно по традиции использовать пустую строку. Общую цену билетов стоит выводить в формате с двумя знаками после запятой.

## Упражнение 70. Биты четности

(*Решено. 27 строк*)

Определение бита четности – это простой механизм выявления ошибок при передаче данных в условиях низкой надежности соединения, например по телефонной линии. Идея заключается в том, что после передачи каждых восьми бит следом отправляется бит четности, позволяющий определить наличие ошибки при передаче одного бита из восьми.

При этом можно использовать как контроль четности, так и контроль нечетности. В первом случае бит четности, посылаемый следом за груп-

пой из восьми бит данных, выбирается таким образом, чтобы общее количество переданных единичек в числе восьми бит данных и проверочного бита было четным. Во втором случае их количество должно быть нечетным.

Напишите программу, вычисляющую значение бита четности для групп из восьми бит, введенных пользователем, с использованием контроля четности. Пользователь может вводить комбинации из восьми бит бесконечное количество раз, а индикатором окончания ввода пусть снова будет пустая строка. После каждой введенной группы из восьми бит программа должна выводить на экран сообщение о том, чему должен равняться бит четности: 0 или 1. Также осуществляйте контроль ввода и выводите соответствующее сообщение об ошибке, если пользователь ввел последовательность, отличную от восьми бит.

**Подсказка.** Пользователь должен вводить последовательность в виде строки. После ввода вы можете определить количество нулей и единиц во введенной последовательности при помощи метода `count`. Информацию о работе этого метода можно найти в онлайне.

## *Упражнение 71. Приблизительное число π*

(*23 строки*)

Приблизительное значение числа π можно вычислить по следующей бесконечной формуле:

$$\pi \approx 3 + \frac{4}{2\times3\times4} - \frac{4}{4\times5\times6} + \frac{4}{6\times7\times8} - \frac{4}{8\times9\times10} + \frac{4}{10\times11\times12} - \cdots.$$

Напишите программу, выводящую на экран 15 приближений числа π. В первом приближении должно быть использовано только первое слагаемое приведенного бесконечного ряда. Каждое очередное приближение должно учитывать следующее слагаемое, тем самым увеличивая точность расчета.

## *Упражнение 72. Игра Fizz-Buzz*

(*17 строк*)

Fizz-Buzz – это известная игра, помогающая детям освоить в игровой форме правила деления. Участники садятся в круг, чтобы игра теоретически могла продолжаться бесконечно. Первый игрок говорит «Один» и передает ход тому, кто слева. Каждый следующий игрок должен мысленно прибавить к предыдущему числу единицу и произнести либо его, либо одно из ключевых слов: Fizz, если число без остатка делится на три,

или Buzz, если на пять. Если соблюдаются оба этих условия, он произносит Fizz-Buzz. Игрок, не сумевший сказать правильное слово, выбывает из игры. Последний оставшийся игрок признается победителем.

Разработайте программу, реализующую алгоритм игры Fizz-Buzz применительно к первым 100 числам. Каждый следующий ответ должен отображаться на новой строке.

## *Упражнение 73. Код Цезаря*

(*Решено. 35 строк*)

Одним из первых в истории примеров шифрования считаются закодированные послания Юлия Цезаря. Римскому полководцу необходимо было посылать письменные приказы своим генералам, но он не желал, чтобы в случае чего их прочитали недруги. В результате он стал шифровать свои послания довольно простым методом, который впоследствии стали называть кодом Цезаря.

Идея шифрования была совершенно тривиальной и заключалась в циклическом сдвиге букв на три позиции. В итоге буква A превращалась в D, B – в E, C – в F и т. д. Последние три буквы алфавита переносились на начало. Таким образом, буква X становилась A, Y – B, а Z – C. Цифры и другие символы не подвергались шифрованию.

Напишите программу, реализующую код Цезаря. Позвольте пользователю ввести фразу и количество символов для сдвига, после чего выведите результирующее сообщение. Убедитесь в том, что ваша программа шифрует как строчные, так и прописные буквы. Также должна быть возможность указывать отрицательный сдвиг, чтобы можно было использовать вашу программу для расшифровки фраз.

## *Упражнение 74. Квадратный корень*

(*14 строк*)

Напишите программу, реализующую метод Ньютона для нахождения квадратного корня числа x, введенного пользователем. Алгоритм реализации метода Ньютона следующий:

```
Запрашиваем число x у пользователя
Присваиваем переменной guess значение x / 2
Пока значение переменной guess не будет обладать должной точностью
    Присваиваем переменной guess результат вычисления среднего между guess и x / guess
```

По завершении алгоритма в переменной *guess* будет находиться определенное приближение вычисления квадратного корня из *x*. Качество аппроксимации при этом будет зависеть только от вашего желания. В нашем случае расхождение между значениями *guess* * *guess* и *x* должно составлять не более $10^{-12}$.

## *Упражнение 75. Палиндром или нет?*

(*Решено. 26 строк*)

Строка называется палиндромом, если она пишется одинаково в обоих направлениях. Например, палиндромами в английском языке являются слова «anna», «civic», «level», «hannah». Напишите программу, запрашивающую у пользователя строку и при помощи цикла определяющую, является ли она палиндромом.

**Примечание.** Яибофобия (Aibohphobia) – это безрассудный страх палиндромов. Эти слова в русском и английском сами по себе являются палиндромами, что и привело к их образованию. Напротив, яилифилия (ailihphilia) характеризуется любовью к палиндромам. Объяснять образование этого слова нет нужды.

## *Упражнение 76. Многословные палиндромы*

(*35 строк*)

Помимо слов, существуют целые фразы, являющиеся палиндромами, если не обращать внимания на пробелы. Вот лишь несколько примеров на английском: «go dog», «flee to me remote elf» and «some men interpret nine memos». Русские варианты есть следующие: «А кобыле цена дана, да не целы бока», «А Луна канула» и другие. Расширьте свое решение упражнения под номером 75, чтобы при вынесении решения о том, является ли строка палиндромом, игнорировались пробелы. Также можете поработать над тем, чтобы игнорировались знаки препинания, а заглавные и прописные буквы считались эквивалентными.

## *Упражнение 77. Таблица умножения*

(*Решено. 18 строк*)

В данном упражнении вы создадите программу для отображения стандартной таблицы умножения от единицы до десяти. При этом ваша таблица умножения должна иметь заголовки над первой строкой и слева от первого столбца, как показано в представленном примере. Предполагаемый вывод таблицы умножения показан ниже.

```
    1   2   3   4   5   6   7   8   9  10
1   1   2   3   4   5   6   7   8   9  10
2   2   4   6   8  10  12  14  16  18  20
3   3   6   9  12  15  18  21  24  27  30
4   4   8  12  16  20  24  28  32  36  40
5   5  10  15  20  25  30  35  40  45  50
6   6  12  18  24  30  36  42  48  54  60
7   7  14  21  28  35  42  49  56  63  70
```

```
 8   8  16  24  32  40  48  56  64  72  80
 9   9  18  27  36  45  54  63  72  81  90
10  10  20  30  40  50  60  70  80  90 100
```

Возможно, для выполнения этого упражнения вам придется озаботиться тем, чтобы выводить значения на экран без принудительного перевода каретки на строку ниже. Этого можно добиться, если последним аргументом функции `print` передать `end=""`. Например, инструкция `print("A")` выведет на экран букву A, после чего автоматически перейдет на новую строку, тогда как `print("A", end="")` не станет переводить каретку, что позволит произвести следующий вывод в той же строке.

## *Упражнение 78. Гипотеза Коллатца*

(*Решено. 18 строк*)

Представьте себе последовательность целых чисел, организованную следующим образом.

```
Начинаться последовательность должна с любого положительного числа
Пока последний элемент последовательности не равен единице, выполнять
    Если последний элемент последовательности четный, тогда
        Добавить новый элемент к последовательности путем деления последнего элемента
на два с округлением вниз
    Иначе
        Добавить новый элемент к последовательности путем умножения последнего элемента
на три с добавлением единицы.
```

Гипотеза Коллатца утверждает, что подобная последовательность при условии того, что начинается с положительного числа, рано или поздно завершится единицей. И хотя это так и не было доказано, все указывает на то, что это так и есть.

Напишите программу, которая будет запрашивать у пользователя целое число и выводить все числа, начиная с введенного числа и заканчивая единицей. После этого пользователь должен иметь возможность ввести другое число и снова получить ряд чисел, называемый сиракузской последовательностью. Условием выхода из программы должен быть ввод пользователем нуля или отрицательного числа.

**Примечание.** Гипотеза Коллатца по сей день остается одной из нерешенных проблем математики. Многие пытались представить доказательство, но никто не добился успеха.

## *Упражнение 79. Наибольший общий делитель*

(*Решено. 17 строк*)

Наибольший общий делитель двух положительных чисел представляет собой наибольшее число, на которое без остатка делятся оба числа. Существует несколько алгоритмов, позволяющих определить наибольший общий делитель двух чисел, включая следующий.

```
Инициализируйте переменную d меньшим из чисел n и m
Пока n или m не делятся на d без остатка, выполнять
    Уменьшить d на единицу
Выведите на экран d, это и есть наибольший общий делитель для n и m
```

Напишите программу, запрашивающую у пользователя два положительных целых числа и выводящую для них наибольший общий делитель.

## *Упражнение 80. Простые множители*

(*27 строк*)

Разложение целого числа *n* на простые множители может быть проведено по следующему алгоритму.

```
Инициализируйте переменную factor значением 2
Пока значение factor меньше или равно n, выполнять
    Если n без остатка делится на factor, тогда
        Сохранить factor как простой множитель числа n
        Разделить n на factor с округлением вниз
    Иначе
        Увеличить factor на единицу
```

Напишите программу, которая будет запрашивать целое число у пользователя. Если пользователь введет значение, меньшее двух, необходимо вывести соответствующее сообщение об ошибке. Иначе нужно перечислить в столбец список простых множителей заданного числа, которые при перемножении дали бы исходное число. Например:

```
Введите целое число (2 или больше): 72
Простые множители числа 72:
2
2
2
3
3
```

## *Упражнение 81. Двоичное число в десятичное*

(*18 строк*)

Напишите программу, которая будет преобразовывать двоичные значения (по основанию 2) в десятичные (по основанию 10). Пользователь должен ввести число в двоичном виде как строку, а программа – преобразовать его посимвольно в десятичный вид и вывести на экран с соответствующим сообщением.

## *Упражнение 82. Десятичное число в двоичное*

(*Решено. 27 строк*)

Напишите программу, которая будет преобразовывать десятичные значения (по основанию 10) в двоичные (по основанию 2). Запросите целое число у пользователя и, следуя алгоритму, приведенному ниже, преобразуйте его в двоичную запись. По завершении выполнения программы в переменной *result* должно оказаться двоичное представление исходного числа. Выведите результат на экран с соответствующим сообщением.

```
Инициализируйте переменную result пустой строкой
Пусть в переменной q хранится число, которое нужно преобразовать
Повторять
    Переменной r присвоить остаток от деления q на 2
    Преобразовать r в строку и добавить ее в начало переменной result
    Разделить q на 2 с отбрасыванием остатка и присвоить полученное значение переменной q
Пока q не станет равно нулю
```

## *Упражнение 83. Максимальное число в последовательности*

(*Решено. 34 строки*)

Это упражнение преследует цель идентификации количества смен максимального значения в коллекции случайных чисел. Ряд должен быть заполнен числами от 1 до 100. При этом последовательность может содержать дубликаты, а некоторых чисел из диапазона от 1 до 100 в ней может не быть.

Сделайте паузу и подумайте с листочком в руках, как вы решили бы эту задачу. Многие стали бы сравнивать каждое очередное выпавшее число с текущим максимумом в последовательности и при необходимости обновлять максимум. Это вполне подходящий способ, который приведет к правильному результату при соблюдении алгоритма. А как вы думаете, сколько раз обновится максимум на протяжении генерирования всей последовательности?

На этот вопрос можно ответить при помощи теории вероятностей, но мы попробуем провести необходимые симуляции. Для начала в вашей

программе должно выбираться случайное число из диапазона от 1 до 100. Сразу сохраните это значение как максимальное. Далее сгенерируйте еще 99 случайных чисел в том же диапазоне. Для каждого значения выполняйте ту же проверку и при необходимости обновляйте максимум, попутно увеличивая переменную, хранящую количество «смен лидера», на единицу. Выводите каждое сгенерированное число на новой строке, добавляя примечание в случае, если на этом шаге обновлялся максимум.

После вывода всех значений на экран вы должны также оповестить пользователя о максимальном числе в ряду и количестве его обновлений. Частичный вывод программы приведен ниже. Запустите свою программу несколько раз. Оправдались ли ваши ожидания относительно количества смен максимального значения?

```
30
74 <== Обновление
58
17
40
37
13
34
46
52
80 <== Обновление
37
97 <== Обновление
45
55
73
…

Максимальное значение в ряду: 100
Количество смен максимального значения: 5
```

## *Упражнение 84. Подбрасываем монетку*

(*47 строк*)

Какое минимальное количество раз вы должны подбросить монетку, чтобы три раза подряд выпал либо орел, либо решка? А какое максимальное количество попыток может для этого понадобиться? А в среднем? В данном упражнении мы выясним это, создав симулятор подбрасывания виртуальной монетки.

Напишите программу, использующую для подброса монетки генератор случайных чисел Python. Монетка при этом должна быть правильной формы, что означает равную вероятность выпадения орла и решки. Подбрасывать монетку необходимо до тех пор, пока три раза подряд не вы-

падет одно значение, вне зависимости от того, орел это будет или решка. Выводите на экран букву `O` всякий раз, когда выпадает орел, и `P` – когда выпадает решка. При этом для одной симуляции бросков все выпавшие значения необходимо размещать на одной строке. Также необходимо известить пользователя о том, сколько попыток потребовалось, чтобы получить нужный результат.

Программа должна выполнить десять симуляций и в конце представить среднее количество подбрасываний монетки, требуемое для достижения нужного нам результата. Пример вывода программы показан ниже:

```
O P P P (попыток: 4)
O O P P O P O P P O O P O P P O P P P (попыток: 19)
P P P (попыток: 3)
P O O O (попыток: 4)
O O O (попыток: 3)
P O P P O P O O P P O O P O P O O O (попыток: 18)
O P P O O O (попыток: 6)
P O P P P (попыток: 5)
P P O P P O P O P O O O (попыток: 12)
P O P P P (попыток: 5)
Среднее количество попыток: 7,9.
```

# Глава 4

# Функции

С ростом программ, которые мы пишем, возрастает необходимость в их максимальном упрощении и облегчении их поддержки и сопровождения. Одним из способов добиться этого является дробление программ на отдельные блоки, именуемые *функциями* (function).

Функции в программах служат сразу нескольким целям. Они позволяют написать код лишь раз и впоследствии обращаться к нему из разных мест. Кроме того, с их помощью можно легко тестировать различные решения. Также функции помогают скрыть (или, по крайней мере, отложить в сторону) детали реализации программного кода после окончания разработки части программного комплекса. С помощью функций программист имеет возможность написать необходимое количество строк кода, запрятать их в единый блок, именуемый функцией, и отложить для использования в будущем. К этому коду в дальнейшем можно будет обращаться из других блоков и программ. При этом сам блок функции со всеми входящими в него инструкциями именуется *определением* (defining), а обращение к нему извне – *вызовом* (calling) функции. После вызова и выполнения функции управление возвращается в основную программу, и она продолжается с первой после вызова функции инструкции.

Программы, которые вы писали до этого, изобиловали вызовами функций вроде `print`, `input`, `int` и `float`. Все эти функции были заранее написаны разработчиками языка Python и могут быть вызваны в любой программе на этом языке. В данной главе вы научитесь объявлять и вызывать собственные функции в дополнение к уже написанным и помещенным в стандартные библиотеки Python.

Определение функции начинается с ключевого слова *def*, следом за которым идет имя функции, открывающая и закрывающая скобки и знак двоеточия. Далее следует набор инструкций с отступами (как в случае с циклами и условными выражениями), которые и составляют тело функции. Функция завершается тогда, когда отступы в строках восстанавливаются до уровня, на котором стоит определение функции с ключевым словом `def`. Например, в следующем программном коде представлена функция, рисующая прямоугольник из звездочек.

```
def drawBox():
    print("**********")
    print("*        *")
    print("*        *")
    print("**********")
```

Сами по себе эти строки кода, появившись в программе, не ведут к рисованию прямоугольника из звездочек на экране, поскольку мы не вызывали функцию, а только объявили ее. Объявление функции откладывает на будущее заключенные в ней строки кода, ассоциируя их с выбранным именем функции – в нашем случае это `drawBox`, – но не выполняет их немедленно. Программа на языке Python, состоящая исключительно из объявлений функций, имеет право на существование, но при запуске не будет генерировать никакого выхода.

Функцию `drawBox` можно вызвать в программе, обратившись к ней по имени со скобками. Добавление следующей строки к коду (без отступов) позволит осуществить вызов функции `drawBox` и вывести на экран прямоугольник.

```
drawBox()
```

Если следом вставить еще одну такую строку кода, на экран будет выведено два прямоугольника, а если три, то три. Функции могут быть вызваны в программе столько раз, сколько необходимо для решения той или иной задачи, и из самых разных мест. При каждом очередном вызове функции выполняется ее тело. По окончании выполнения функции программа продолжается с инструкции, следующей за строкой вызова функции.

## 4.1. Функции с параметрами

Функция `drawBox` работает отменно. При вызове она исправно выводит на экран прямоугольник, что от нее и требуется. Но вы не находите, что наша функция обладает недостаточной гибкостью, а значит, и пользы от нее не так много? Было бы интереснее, если бы у нас была возможность выводить на экран прямоугольники разной формы.

Для большей гибкости функции могут принимать *аргументы* (argument) посредством передачи им значений в круглых скобках. На входе в функцию значениями этих аргументов инициализируются *переменные параметров* (parameter variables), прописанные в круглых скобках при объявлении функции. Количество параметров, прописанное при объявлении функции, определяет число аргументов, которые должны быть переданы при вызове функции.

Нашу функцию `drawBox` можно сделать куда более полезной, если снабдить ее двумя входными параметрами. Пусть это будут ширина и высота

прямоугольника соответственно. В теле функции переменные, переданные в качестве параметров, будут использованы для определения характеристик выводимого на экран прямоугольника, как показано ниже. Выражение `if` и функцию `quit` мы используем для немедленного выхода из функции, если ей переданы неправильные значения в качестве параметров.

```
## Рисуем прямоугольник из звездочек, заполненный пробелами
# @param width - ширина прямоугольника
# @param height - высота прямоугольника
def drawBox(width, height):
    # Слишком маленькие прямоугольники не могут быть нарисованы при помощи этой функции
    if width < 2 or height < 2:
        print("Ошибка: ширина или высота прямоугольника слишком малы.")
        quit()

    # Рисуем верхнюю линию прямоугольника
    print("*" * width)

    # Рисуем стороны прямоугольника
    for i in range(height - 2):
        print("*" + " " * (width - 2) + "*")

    # Рисуем нижнюю линию прямоугольника
    print("*" * width)
```

Теперь при вызове функции `drawBox` ей должны быть переданы два аргумента, обозначенные в виде параметров при ее объявлении. В момент вызова функции значение первого аргумента будет записано в переменную первого параметра (`width`), а значение второго – в переменную второго параметра (`height`). К примеру, следующий вызов функции приведет к отображению прямоугольника шириной 15 единиц и высотой четыре единицы. Чтобы нарисовать прямоугольник с другими характеристиками, нужно вызвать нашу функцию еще раз и передать в нее уже другие аргументы.

```
drawBox(15, 4)
```

Сейчас наша функция `drawBox` рисует границы прямоугольника исключительно звездочками, а внутренняя его часть заполнена пробелами. И хотя выглядит он неплохо, иногда может возникнуть необходимость в рисовании сторон прямоугольника и его заполнении другими символами. С этой целью мы добавим к объявлению функции `drawBox` еще два параметра, как показано в обновленной версии программы ниже. Пример вызова функции с альтернативными символами для сторон и заполнения прямоугольника показан в конце фрагмента кода.

```
## Рисуем прямоугольник
# @param width - ширина прямоугольника
```

```
# @param height - высота прямоугольника
# @param outline - символ для рисования сторон прямоугольника
# @param fill - символ для заливки прямоугольника
def drawBox(width, height, outline, fill):
    # Слишком маленькие прямоугольники не могут быть нарисованы при помощи этой функции
    if width < 2 or height < 2:
        print("Ошибка: ширина или высота прямоугольника слишком малы.")
        quit()

    # Рисуем верхнюю линию прямоугольника
    print(outline * width)

    # Рисуем стороны прямоугольника
    for i in range(height - 2):
        print(outline + fill * (width - 2) + outline)

    # Рисуем нижнюю линию прямоугольника
    print(outline * width)

# Демонстрируем работу обновленной функции
drawBox(14, 5, "@", ".")
```

При вызове новой версии функции `drawBox` программист обязательно должен передавать третий и четвертый параметры в дополнение к первым двум. Это может быть приемлемо, если значения параметров постоянно меняются, но если, к примеру, звездочка и пробел используются в большинстве случаев для рисования прямоугольников, можно применить иной подход, добавив к некоторым параметрам функции *значения по умолчанию* (default value) при ее объявлении. Для этого используется оператор присваивания, как показано в примере ниже.

```
def drawBox(width, height, outline="*", fill=" "):
```

В таком виде функция `drawBox` может вызываться с двумя, тремя или четырьмя аргументами. В первом случае значения первого и второго переданных аргументов будут присвоены переменным ширины и высоты будущего прямоугольника, тогда как переменным `outline` и `fill` будут присвоены значения по умолчанию, а именно звездочка и пробел соответственно. Переменные параметров получают значения по умолчанию только тогда, когда для них не переданы соответствующие аргументы при вызове функции.

Теперь давайте рассмотрим следующий вызов функции.

```
drawBox(14, 5, "@", ".")
```

В вызов функции включены четыре аргумента. Первые два соответствуют ширине и высоте прямоугольника и присваиваются соответствующим переменным в теле функции. Третий аргумент характеризует символ для

линий фигуры. Поскольку в этом случае мы передали на вход функции фактическое желаемое значение для параметра, отвечающего за тип линий прямоугольника, значение переменной по умолчанию будет заменено на знак `@`. То же самое произойдет и с четвертым аргументом – соответствующая ему переменная параметра будет инициализирована точкой. В результате фигура, которая будет построена при помощи подобного вызова функции, обретет следующий вид.

```
@@@@@@@@@@@@@@
@............@
@............@
@............@
@@@@@@@@@@@@@@
```

# 4.2. Переменные в функциях

Переменные, объявляемые внутри функции, называются *локальными* (local). Это означает, что область действия этих переменных ограничивается данной функцией, и к ним можно обратиться только в теле этой функции. После завершения работы функции локальные переменные очищаются из памяти, а значит, и обратиться к ним больше не получится. В функции `drawBox` задействовано сразу несколько локальных переменных. В их число входят и переменные параметров вроде `width` и `fill`, и даже внутренняя переменная `i` из цикла `for` внутри функции. Ко всем этим переменным можно обращаться только внутри тела функции. Переменные, создаваемые и инициализируемые внутри функции при помощи оператора присваивания, также классифицируются как локальные.

# 4.3. Возвращаемые значения

Наша функция выводит прямоугольник на экран. И хотя она принимает на вход целых четыре аргумента, никаких вычислений, результат которых стоило бы сохранить в переменной и использовать позже, внутри функции не производится. Но многие функции пишутся именно для этого. Например, функция `sqrt` из модуля `math` вычисляет квадратный корень из переданного в нее аргумента и возвращает его для дальнейшего использования. Так же точно функция `input` считывает ввод с клавиатуры и возвращает введенное значение для дальнейшей его обработки. Многие функции, которые вы будете писать сами, тоже будут возвращать значения.

Для возврата значения из функции существует ключевое слово *return*, после которого идет само значение. Дойдя до инструкции с `return`, функ-

ция мгновенно прекращает свое выполнение, и управление передается инструкции, следующей сразу за вызовом функции. Например, следующее выражение прекращает выполнение функции, возвращая при этом значение 5.

```
return 5
```

Функции, возвращающие значение, обычно появляются в выражениях справа от оператора присваивания, но могут быть вызваны и в других контекстах, когда необходимо получить рассчитанное значение. Например, подобные функции могут быть частью условного выражения `if` или цикла `while`, а также подаваться на вход других функций, таких как `print` или `range`.

В функциях, не предназначенных для возвращения результата, совсем не обязательно использовать ключевое слово `return`, поскольку возврат в основную программу автоматически произойдет после выполнения последней инструкции в теле функции. Но программист вправе использовать слово `return` без следующего за ним значения для преждевременного выхода из функции. Кроме того, в любой функции, вне зависимости от того, возвращает она значение или нет, ключевое слово `return` может появляться множество раз. Выход из подобной функции произойдет по достижении первого из этих слов.

Рассмотрим еще один пример. Геометрическая прогрессия представляет собой последовательность чисел, начинающуюся с числа $a$ и продолжающуюся до бесконечности. При этом каждый следующий член последовательности, за исключением первого, получается путем умножения предыдущего на $r$, называемое знаменателем прогрессии. В результате прогрессию можно выразить как последовательность $a, ar, ar^2, ar^3, \ldots$. При $r$, равном единице, сумма первых $n$ элементов геометрической прогрессии составит $a \times n$. В противном случае сумму первых $n$ элементов можно вычислить по следующей формуле:

$$\text{sum} = \frac{a(1 - r^n)}{1 - r}.$$

Несложно написать функцию на языке Python, вычисляющую эту сумму для любой геометрической прогрессии при известном $n$. Функция будет принимать на вход три параметра – $a$, $r$ и $n$ – и возвращать сумму первых $n$ элементов последовательности. Код функции показан ниже.

```
## Вычислить сумму первых n элементов геометрической прогрессии
# @param a - первый элемент последовательности
# @param r - знаменатель последовательности
# @param n - количество элементов, сумму которых необходимо получить
# @return s - сумма первых n элементов
def sumGeometric(a, r, n):
```

```
    # Вычисляем и возвращаем сумму первых n элементов при r, равном 1
    if r == 1:
        return a * n
    # Вычисляем и возвращаем сумму первых n элементов при r, не равном 1
    s = a * (1 - r ** n) / (1 - r)
    return s
```

Функция начинается с использования условной конструкции `if` для определения того, равняется ли `r` единице. Если это так, сумму первых `n` элементов можно вычислить по простой формуле `a * n`, после чего вернуть результат вызывающей программе, не выполняя оставшийся код функции. Если значение переменной `r` не равно единице, тело условного выражения пропускается, а вместо него в переменную `s` помещается результат подсчета нужной нам суммы по формуле, описанной выше. После этого результат возвращается в виде переменной `s`, и выполнение функции на этом заканчивается.

В следующей программе мы рассмотрим пример использования нашей функции `sumGeometric` для вычисления суммы первых `n` элементов геометрической прогрессии, пока пользователь не введет для переменной `a` нулевое значение. Каждое значение суммы рассчитывается отдельно, возвращается и присваивается переменной с именем `total`. Полученная сумма выводится на экран, после чего происходит следующий запрос пользователю.

```
def main():
    # Запрашиваем значение переменной a для первой последовательности
    init = float(input("Введите значение переменной a (0 для выхода): "))

    # Пока вводимое значение не равно нулю
    while init != 0:
        # Запрашиваем знаменатель и количество элементов
        ratio = float(input("Введите знаменатель последовательности, r: "))
        num = int(input("Введите количество элементов, n: "))
        # Вычисляем и отображаем результат
        total = sumGeometric(init, ratio, num)
        print("Сумма первых", num, "элементов равна", total)
        # Запрашиваем значение переменной a для следующей последовательности
        init = float(input("Введите значение переменной a (0 для выхода): "))

# Вызываем основную функцию
main()
```

## 4.4. Импорт функций в другие программы

Одним из преимуществ написания функций является то, что впоследствии вы можете обращаться к ним из различных мест. Если объявление

функции и ее вызов располагаются в одном файле, никаких проблем нет. В этом случае вам достаточно вызвать функцию по имени с круглыми скобками и аргументами.

Но однажды вам непременно захочется воспользоваться написанной ранее функцией в новом проекте. Многие программисты, даже самые опытные, предпочитают просто копировать код функций из старых файлов и вставлять в новые, но это не лучший подход. Копирование функции приводит к ее полному дублированию, и если вам вдруг придется внести изменения в работу функции, вы вынуждены будете делать это во всех файлах. Гораздо лучше просто импортировать функцию из старого файла в новый так же точно, как мы импортировали функции из встроенных в Python модулей, – при помощи ключевого слова `import` с последующим указанием имени файла (без расширения .py), в котором находятся нужные функции. Это позволит обращаться к нужным функциям, но при этом будет также запущена программа из старого файла. В некоторых ситуациях это допустимо, но не всегда. В подобном случае обычно создается функция с именем `main`, содержащая необходимые выражения для решения задачи. В конце кода ставится вызов функции `main`. И наконец, вставляется условная конструкция, приведенная ниже, предотвращающая запуск функции `main` при выполнении импорта из другого файла.

```
if __name__ == "__main__":
    main()
```

Такую структуру файла стоит использовать всякий раз, когда вы знаете, что написанные в нем функции в будущем могут пригодиться в других программах.

## 4.5. Упражнения

Функции позволяют нам именовать целые блоки кода на языке Python и вызывать их впоследствии из разных мест. Такой подход дает целый ряд преимуществ по сравнению с программами, написанными без функций, включая возможность повторного использования кода и дополнительные опции для локального тестирования программы. Функции также позволяют программисту вынести за скобки детали реализации того или иного блока кода и сосредоточиться на более важных вещах. Эффективное использование функций позволит вам улучшить свои программы в целом, особенно когда речь идет о решении довольно серьезных задач. Все упражнения из данной главы должны быть выполнены с применением функций.

## *Упражнение 85. Вычисляем длину гипотенузы*

(*23 строки*)

Напишите функцию, принимающую на вход длины двух катетов прямоугольного треугольника и возвращающую длину гипотенузы, рассчитанную по теореме Пифагора. В главной программе должен осуществляться запрос длин сторон у пользователя, вызов функции и вывод на экран полученного результата.

## *Упражнение 86. Плата за такси*

(*22 строки*)

Представьте, что сумма за пользование услугами такси складывается из базового тарифа в размере $4,00 плюс $0,25 за каждые 140 м поездки. Напишите функцию, принимающую в качестве единственного параметра расстояние поездки в километрах и возвращающую итоговую сумму оплаты такси. В основной программе должен демонстрироваться результат вызова функции.

**Подсказка.** Цены на такси могут меняться со временем. Используйте константы для представления базового тарифа и плавающей ставки, чтобы программу можно было легко обновлять при изменении цен.

## *Упражнение 87. Расчет стоимости доставки*

(*23 строки*)

Интернет-магазин предоставляет услугу экспресс-доставки для части своих товаров по цене $10,95 за первый товар в заказе и $2,95 – за все последующие. Напишите функцию, принимающую в качестве единственного параметра количество товаров в заказе и возвращающую общую сумму доставки. В основной программе должны производиться запрос количества позиций в заказе у пользователя и отображаться на экране сумма доставки.

## *Упражнение 88. Медиана трех значений*

(*Решено. 43 строки*)

Напишите функцию, которая будет принимать на вход три числа в качестве параметров и возвращать их медиану. В основной программе должен производиться запрос к пользователю на предмет ввода трех чисел, а также вызов функции и отображение результата.

**Подсказка.** Медианой называется число, находящееся ровно посередине отсортированной по возрастанию последовательности. Его можно получить путем реализации условных блоков `if` или с применением творческого подхода к математике и статистике.

## *Упражнение 89. Переводим целые числа в числительные*

(*47 строк*)

Такие слова, как первый, второй, третий, являются числительными. В данном упражнении вам необходимо написать функцию, принимающую на вход в качестве единственного аргумента целое число и возвращающую строковое значение, содержащее соответствующее числительное (на английском языке). Ваша функция должна обрабатывать числа в диапазоне от 1 до 12. Если входящее значение выходит за границы этого диапазона, вывод должен оставаться пустым. В основной программе запустите цикл по натуральным числам от 1 до 12 и выведите на экран соответствующие им числительные. Ваша программа должна запускаться только в том случае, если она не импортирована в виде модуля в другой файл.

## *Упражнение 90. Двенадцать дней Рождества*

(*Решено. 52 строки*)

«Двенадцать дней Рождества» (The Twelve Days of Christmas) – старая английская песня, построение которой базируется на постоянно увеличивающемся списке подарков в каждый из 12 дней Рождества. В первый день был послан один подарок, в следующий – второй и т. д. Первые три куплета песни приведены ниже. Полностью текст песни можно без труда найти в интернете.

On the first day of Christmas
my true love sent to me:
A partridge in a pear tree.

On the second day of Christmas
my true love sent to me:
Two turtle doves,
And a partridge in a pear tree.

On the third day of Christmas
my true love sent to me:
Three French hens,
Two turtle doves,
And a partridge in a pear tree.

Напишите программу, которая будет сама строить куплеты этой песенки. В программе должна присутствовать функция для отображения одного куплета. В качестве входного параметра она должна принимать порядковый номер дня, а в качестве результата возвращать готовый куплет. Далее в основной программе эта функция должна быть вызвана 12 раз подряд. Каждая строка с очередным подарком должна присутствовать в вашей программе лишь раз, за исключением строки «A partridge in a pear tree». В этом случае вы можете отдельно хранить такой вид строки для первого куплета и слегка измененный («And a partridge in a pear tree») – для всех последующих. Импортируйте свою функцию из упражнения 89 для выполнения этого задания.

## *Упражнение 91. Григорианский календарь в порядковый*

(*72 строки*)

Порядковая дата содержит номер года и порядковый номер дня в этом году – оба в целочисленном формате. При этом год может быть любым согласно григорианскому календарю, а номер дня – числом в интервале от 1 до 366 (чтобы учесть високосные годы). Порядковые даты удобно использовать при расчете разницы в днях, когда счет ведется именно в днях, а не месяцах. Например, это может касаться 90-дневного периода возврата товара для покупателей, расчета срока годности товаров или прогнозируемой даты появления малыша на свет.

Напишите функцию с именем `ordinalDate`, принимающую на вход три целых числа: день, месяц и год. Функция должна возвращать порядковый номер заданного дня в указанном году. В основной программе у пользователя должны запрашиваться день, месяц и год соответственно и выводиться на экран порядковый номер дня в заданном году. Программа должна запускаться только в том случае, если она не импортирована в виде модуля в другой файл.

## *Упражнение 92. Порядковая дата в григорианский календарь*

(*103 строки*)

Разработайте функцию, принимающую в качестве единственного параметра порядковую дату, включающую в себя год и день по порядку. В качестве результата функция должна возвращать день и месяц, соответствующие переданной порядковой дате. Убедитесь, что ваша функция корректно обрабатывает високосные годы.

Используйте эту функцию, а также функцию `ordinalDate`, написанную при выполнении упражнения 91, для разработки основной программы. Для начала должен производиться запрос порядковой даты у пользова-

теля. После этого программа должна вычислить вторую дату, отстоящую от первой на определенное количество дней. Например, ваша программа могла бы запрашивать у пользователя порядковую дату, когда был приобретен товар, и выводить последнюю дату, когда можно осуществить возврат (согласно определенным правилам возврата товаров). Или вы могли бы спрогнозировать дату появления ребенка на свет на основании срока беременности в 280 дней. Удостоверьтесь, что программа корректно обрабатывает ситуации, когда заданная дата и расчетная находятся в разных годах.

## *Упражнение 93. Центрируем строку*

(*Решено. 29 строк*)

Напишите функцию, которая будет принимать в качестве параметров строку *s*, а также ширину окна в символах – *w*. Возвращать функция должна новую строку, в которой в начале добавлено необходимое количество пробелов, чтобы первоначальная строка оказалась размещена по центру заданного окна. Новая строка должна формироваться по следующему принципу:

- если длина исходной строки *s* больше или равна ширине заданного окна, возвращаем ее в неизменном виде;
- в противном случае должна быть возвращена строка *s* с ведущими пробелами в количестве `(len(s) - w) // 2` штук.

В вашей основной программе должен осуществляться пример вывода нескольких строк в окнах разной ширины.

## *Упражнение 94. Треугольник ли?*

(*33 строки*)

Всем известно, что из трех веточек разной длины далеко не всегда можно составить треугольник, соединив их концы. Например, если все они будут длиной 6 см, можно без труда построить равносторонний треугольник. Но если одна веточка будет длиной 6 см, а остальные две длиной 2 см, треугольник просто не получится. Правило здесь простое: если длина одной стороны больше или равна сумме двух оставшихся сторон, треугольник НЕ образуется. Иначе это возможно.

Напишите функцию для определения возможности построения треугольника на основании длин трех его потенциальных сторон. Функция должна принимать три числовых параметра и возвращать булево значение. Если длина хотя бы одной из трех сторон меньше или равна нулю, функция должна вернуть `False`. В противном случае необходимо выполнить проверку на допустимость построения треугольника на основании введенных длин сторон и вернуть соответствующее значение. Напиши-

те основную программу, запрашивающую у пользователя длины сторон и выводящую на экран информацию о том, может ли при заданных значениях получиться треугольник.

## *Упражнение 95. Озаглавим буквы*

(*Решено. 68 строк*)

Многие в своих сообщениях не ставят заглавные буквы, особенно если используют для набора мобильные устройства. Создайте функцию, которая будет принимать на вход исходную строку и возвращать строку с восстановленными заглавными буквами. По существу, ваша функция должна:

- сделать заглавной первую букву в строке, не считая пробелы;
- сделать заглавной первую букву после точки, восклицательного или вопросительного знака, не считая пробелы;
- если текст на английском языке, сделать заглавными буквы «i», которым предшествует пробел или за которыми следует пробел, точка, восклицательный или вопросительный знак.

Реализация такого рода автоматической корректуры исключит большую часть ошибок с регистром букв. Например, строку «what time do i have to be there? what's the address? this time i'll try to be on time!» ваша функция должна преобразовать в более приемлемый вариант «What time do I have to be there? What's the address? This time I'll try to be on time!». В основной программе запросите у пользователя исходную строку, обработайте ее при помощи своей функции и выведите на экран итоговый результат.

## *Упражнение 96. Является ли строка целым числом?*

(*Решено. 30 строк*)

В данном упражнении вам предстоит написать функцию с именем `isInteger`, определяющую, представляет ли введенная строка целочисленное значение. При проверке вы можете игнорировать ведущие и замыкающие пробелы в строке. После исключения лишних пробелов строку можно считать представляющей целое число, если ее длина больше или равна одному символу и она целиком состоит из цифр. Возможен также вариант с ведущим знаком «+» или «−», после которого должны идти цифры.

В основной программе у пользователя должна запрашиваться исходная строка и выводиться сообщение о том, можно ли введенное значение воспринимать как целое число. Убедитесь, что основная программа не будет запускаться, если файл импортирован в другой файл в качестве модуля.

**Подсказка.** При работе с этим заданием вам, вероятно, понадобятся методы `lstrip`, `rstrip` и/или `strip`. Их описание можно найти в интернете.

## *Упражнение 97. Приоритеты операторов*

(*30 строк*)

Напишите функцию с именем `precedence`, которая будет возвращать целое число, представляющее собой приоритет или старшинство математического оператора. В качестве единственного параметра эта функция будет принимать строку, содержащую оператор. На выходе функция должна давать 1 для операторов «+» и «-», 2 для «*» и «/» и 3 для «^». Если строка, переданная в функцию, не содержит ни один из перечисленных операторов, должно быть возвращено значение –1. Дополните функцию основной программой, в которой будет выполняться запрос оператора у пользователя и выводиться на экран его приоритет или сообщение об ошибке, если был осуществлен неверный ввод. Программа должна запускаться только в том случае, если она не импортирована в виде модуля в другой файл.

**Примечание.** В данном упражнении, как и во всех последующих в этой книге, мы будем использовать оператор «^» (крышечка) для возведения в степень. Применение этого символа вместо стандартного для Python «**» позволит облегчить написание программы, поскольку в этом случае все операторы будут состоять из одного символа.

## *Упражнение 98. Простое число?*

(*Решено. 28 строк*)

Простое число представляет собой число, большее единицы, которое без остатка делится лишь на само себя и единицу. Напишите функцию для определения того, является ли введенное число простым. Возвращаемое значение должно быть либо `True`, либо `False`. В основной программе, как и ожидается, пользователь должен ввести целое число и получить ответ о том, является ли оно простым. Убедитесь, что основная программа не будет запускаться, если файл импортирован в другой файл в качестве модуля.

## *Упражнение 99. Следующее простое число*

(*27 строк*)

В данном упражнении вам нужно написать функцию с именем `nextPrime`, которая находит и возвращает первое простое число, большее введенного числа *n*. Само число *n* должно передаваться в функцию в качестве единственного параметра. В основной программе запросите у пользователя это значение и выведите на экран первое простое число, большее заданного. Для решения этой задачи импортируйте функцию, созданную в упражнении 98.

## *Упражнение 100. Случайный пароль*

(*Решено. 33 строки*)

Напишите функцию, которая будет генерировать случайный пароль. В пароле должно быть от 7 до 10 символов, при этом каждый символ должен быть случайным образом выбран из диапазона от 33 до 126 в таблице ASCII. Ваша функция не должна принимать на вход параметры, а возвращать будет сгенерированный пароль. В основной программе вы должны просто вывести созданный случайным образом пароль. Программа должна запускаться только в том случае, если она не импортирована в виде модуля в другой файл.

**Подсказка.** При решении этого упражнения вам, возможно, понадобится функция `chr`. Полную информацию о ней можно найти в интернете.

## *Упражнение 101. Случайный номерной знак*

(*45 строк*)

Представьте, что в вашем регионе устаревшим является формат номерных автомобильных знаков из трех букв, следом за которыми идут три цифры. Когда все номера такого шаблона закончились, было решено обновить формат, поставив в начало четыре цифры, а за ними три буквы.

Напишите функцию, которая будет генерировать случайный номерной знак. При этом номера в старом и новом форматах должны создаваться примерно с одинаковой вероятностью. В основной программе нужно сгенерировать и вывести на экран случайный номерной знак.

## *Упражнение 102. Проверка пароля на надежность*

(*Решено. 40 строк*)

В данном упражнении вам необходимо написать функцию, проверяющую введенный пароль на надежность. Определим как надежный пароль, состоящий минимум из восьми символов и включающий хотя бы по одной букве в верхнем и нижнем регистрах и как минимум одну цифру. Функция должна возвращать `True`, если переданный в качестве параметра пароль отвечает требованиям надежности. В противном случае возвращаемым значением должно быть `False`. В основной программе необходимо запросить у пользователя пароль и оповестить его о том, является ли он достаточно надежным. Программа должна запускаться только в том случае, если она не импортирована в виде модуля в другой файл.

## *Упражнение 103. Случайный надежный пароль*

(*22 строки*)

Используя решения из упражнений 100 и 102, напишите программу, генерирующую случайный надежный пароль и выводящую его на экран. Посчитайте, с какого раза удастся создать пароль, отвечающий нашим требованиям надежности, и выведите на экран количество попыток. Импортируйте функции из предыдущих упражнений и вызывайте их при необходимости для решения этой задачи.

## *Упражнение 104. Шестнадцатеричные и десятичные числа*

(*41 строка*)

Напишите две функции с именами `hex2int` и `int2hex` для конвертации значений из шестнадцатеричной системы счисления (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E и F) в десятичную (по основанию 10) и обратно. Функция `hex2int` должна принимать на вход строку с единственным символом в шестнадцатеричной системе и преобразовывать его в число от нуля до 15 в десятичной системе, тогда как функция `int2hex` будет выполнять обратное действие – принимать десятичное число из диапазона от 0 до 15 и возвращать шестнадцатеричный эквивалент. Обе функции должны принимать единственный параметр со входным значением и возвращать преобразованное число. Удостоверьтесь, что функция `hex2int` корректно обрабатывает буквы в верхнем и нижнем регистрах. Если введенное пользователем значение выходит за допустимые границы, вы должны вывести сообщение об ошибке.

## *Упражнение 105. Произвольные системы счисления*

(*Решено. 71 строка*)

Напишите программу, которая позволит пользователю преобразовывать числа из одной системы счисления в другую произвольным образом. Ваша программа должна поддерживать все системы счисления в диапазоне от 2 до 16 как для входных, так и для выходных данных. Если пользователь выберет систему с основанием, выходящим за границы допустимого, на экран должна быть выведена ошибка. Разделите код программы на несколько функций, включая функцию, конвертирующую число из произвольной системы счисления в десятичную, и обратную функцию, переводящую значение из десятичной системы в произвольную. В основной программе необходимо запросить у пользователя исходную систему счисления, целевую систему, а также число для преобразования. При выполнении данного упражнения вам могут пригодиться функции из заданий 81, 82 и 104.

## *Упражнение 106. Дни в месяце*

(*47 строк*)

Напишите функцию для определения количества дней в конкретном месяце. Ваша функция должна принимать два параметра: номер месяца в виде целого числа в диапазоне от 1 до 12 и год, состоящий из четырех цифр. Убедитесь, что функция корректно обрабатывает февраль високосного года. В основной программе запросите у пользователя номер месяца и год и отобразите на экране количество дней в указанном месяце. При решении этой задачи вам может пригодиться написанная вами функция из упражнения 58.

## *Упражнение 107. Максимальное сокращение дробей*

(*Решено. 46 строк*)

Напишите функцию, принимающую на вход два целочисленных параметра, представляющих числитель и знаменатель дроби. В теле функции должно выполняться максимально возможное сокращение дроби, а полученные в итоге числитель и знаменатель должны быть возвращены исходной программе. Например, если на вход функции передать числа 6 и 63, числитель и знаменатель итоговой дроби должны быть 2 и 21. В основной программе нужно запросить у пользователя числитель и знаменатель исходной дроби, передать их в функцию и вывести на экран результат.

**Подсказка.** В упражнении 79 вы писали функцию для определения наибольшего общего делителя для двух целых чисел. Воспользуйтесь ей в этом задании.

## *Упражнение 108. Переводим меры*

(*Решено. 87 строк*)

Во многих кулинарных книгах до сих пор можно встретить рецепты, в которых ингредиенты отмеряются стаканами, чайными и столовыми ложками. И хотя при наличии этих нехитрых предметов таким рецептам следовать довольно удобно, бывает трудно быстро преобразовать подобные меры при приготовлении рождественского ужина на огромную семью. Например, если в рецепте сказано взять четыре столовые ложки того или иного ингредиента, то при увеличении выхода в четыре раза можно просто отсчитать 16 столовых ложек. Однако гораздо проще было бы привести эту меру к одному стакану.

Напишите функцию, выражающую заданный объем ингредиентов с использованием минимально возможных замеров. Функция должна принимать в качестве параметра количество единиц измерения, а также их тип (стакан, столовая или чайная ложка). На выходе мы должны получить

строку, представляющую указанное количество вещества, с задействованием минимального количества действий и предметов. Например, если на вход функции вы подали объем, равный 59 чайным ложкам, возвращенная строка должна быть такой: «1 cup, 3 tablespoons, 2 teaspoons».

**Примечание.** Используйте в этом упражнении английское написание мер: cup, tablespoon и teaspoon, добавляя к ним во множественном числе окончание *s*.

**Подсказка.** Один стакан вмещает 16 столовых ложек, а одна столовая ложка эквивалентна трем чайным ложкам.

## *Упражнение 109. Магические даты*

(*Решено. 26 строк*)

Магическими называются даты, в которых произведение дня и месяца составляет последние две цифры года. Например, 10 июня 1960 года – магическая дата, поскольку $10 \times 6 = 60$. Напишите функцию, определяющую, является ли введенная дата магической. Используйте написанную функцию в главной программе для отображения всех магических дат в XX веке. Возможно, вам пригодится здесь функция, разработанная в упражнении 106.

# Глава 5

# Списки

До сих пор все переменные, которые мы создавали, хранили единственное значение. При этом само значение могло быть абсолютно любого типа – целочисленного, строкового, булевого и т. д. И хотя для более или менее простых задач это приемлемо, когда речь заходит о больших наборах данных, такого подхода становится недостаточно. И тогда на арену выходят *списки* (list), позволяющие хранить в одной переменной множество значений.

Переменная списка создается так же, как и все остальные, – при помощи оператора присваивания. Единственное отличие в случае со списком состоит в том, что его значения справа от знака равенства должны быть заключены в квадратные скобки и разделены запятыми. Например, в следующем выражении создается список с именем `data`, состоящий из четырех чисел с плавающей запятой. В следующей строке осуществляется вывод списка на экран при помощи функции `print`. При этом будут отображены все четыре числа, поскольку они хранятся в одной переменной `data`.

```
data = [2.71, 3.14, 1.41, 1.62]
print(data)
```

Список может содержать ноль или больше элементов. Пустой список, не содержащий значений, обозначается как `[]` (закрывающая квадратная скобка следует непосредственно за открывающей). Как целочисленные переменные могут быть инициализированы нулевым значением, которое впоследствии может измениться, так и списки могут изначально создаваться пустыми, а позже – по ходу выполнения программы – пополняться элементами.

## 5.1. Доступ к элементам списка

Каждое значение в списке именуется *элементом*. Элементы списка пронумерованы последовательными целыми числами, начиная с нуля. Каждое число идентифицирует конкретный элемент списка и называется *индек-*

*сом* этого элемента. В предыдущем фрагменте кода число 2,71 соответствует индексу 0, а 1,62 – индексу 3.

Обратиться к конкретному элементу списка можно при помощи имени переменной, хранящей список, с последующим индексом, заключенным в квадратные скобки. Например, показанная ниже запись выведет на экран число 3,14. Обратите внимание, что индекс 1 соответствует не первому, а второму в списке элементу.

```
data = [2.71, 3.14, 1.41, 1.62]
print(data[1])
```

Изменить значение конкретного элемента списка можно при помощи обычного оператора присваивания. При этом слева от него должно стоять имя переменной, в которой хранится список, с индексом нужного элемента в квадратных скобках, а справа – новое значение. В результате выполнения этой операции значение будет присвоено соответствующему элементу с удалением предыдущего содержимого. Остальные элементы списка затронуты не будут.

Посмотрите на следующий фрагмент кода. Здесь мы создали список из четырех элементов, после чего изменили значение элемента с индексом 2 на 2,30. В результате выполнения функции `print` будет выведено актуальное содержимое списка со следующими значениями: 2,71, 3,14, 2,30 и 1,62.

```
data = [2.71, 3.14, 1.41, 1.62]
data[2] = 2.30
print(data)
```

# 5.2. Циклы и списки

Инструкция `for` позволяет проходить по элементам любой коллекции. При этом коллекцией может являться как диапазон целых чисел, созданный при помощи функции `range`, так и список. В следующем фрагменте кода цикл `for` используется для суммирования элементов из списка `data`.

```
# Инициализируем переменные data и total
data = [2.71, 3.14, 1.41, 1.62]
total = 0

# Суммируем значения в списке
for value in data:
    total = total + value

# Выводим сумму
print("Сумма значений элементов списка:", total)
```

Данная программа начинается с инициализации переменных `data` и `total`. Затем наступает черед цикла `for`. На первой итерации цикла переменной `value` присваивается значение первого элемента списка `data`, после чего выполняется тело цикла, в котором текущая сумма значений элементов списка увеличивается на `value`.

После выполнения тела цикла начинается следующая итерация, в процессе которой переменной `value` присваивается значение очередного элемента списка `data`, и тело цикла выполняется вновь. Цикл будет выполняться до тех пор, пока не будут пройдены все элементы списка. В результате будет рассчитана сумма значений всех элементов списка. После этого сумма будет выведена на экран, и на этом выполнение программы прекратится.

Иногда необходимо пройти в цикле непосредственно по индексам списка, а не по значениям элементов. Для этого нужно сначала вычислить общее количество элементов, находящихся в списке. Сделать это можно при помощи функции *len*. Данная функция принимает на вход единственный аргумент в виде списка и возвращает текущее количество элементов в нем. Если в списке нет ни одного элемента, функция `len` ожидаемо вернет ноль.

Функцию `len` бывает удобно использовать вместе с функцией `range` для создания коллекции целых чисел, являющихся допустимыми индексами списка. Это можно сделать, передав в качестве единственного аргумента в функцию `range` длину списка. Поднабор индексов можно собрать, если передать функции `range` второй параметр. В следующем фрагменте кода демонстрируется использование цикла `for` для прохода по всем индексам списка `data`, за исключением первого, для определения позиции элемента с наибольшим значением в коллекции.

```
# Инициализируем переменные data и largest_pos
data = [1.62, 1.41, 3.14, 2.71]
largest_pos = 0

# Находим позицию элемента с наибольшим значением
for i in range(1, len(data)):
    if data[i] > data[largest_pos]:
        largest_pos = i

# Отображаем результат
print("Наибольшее значение", data[largest_pos], \
    "находится в элементе с индексом", largest_pos)
```

Программа начинается с инициализации переменных `data` и `largest_pos`. После этого с использованием функции `range` создается коллекция, по которой будет проходить цикл `for`. При этом первым аргументом передается единица, а вторым – длина списка, равная четырем. В результате

в коллекции оказываются последовательные целые числа от единицы до трех, которые адресуют все элементы списка data, кроме первого, что нам и нужно.

На первой итерации цикла for внутренней переменной цикла i присваивается значение 1, после чего в первый раз выполняется тело цикла. Внутри него выполняется сравнение элементов списка data с индексом i и largest_pos. Поскольку первых из них меньше, булево выражение возвращает значение False, и тело блока if пропускается.

Далее управление передается в начало цикла for, и на этот раз переменной i присваивается значение 2. Тело цикла выполняется во второй раз. В этом случае значение элемента с индексом i оказывается больше по сравнению с largest_pos, в результате чего переменной largest_pos присваивается значение i, то есть 2.

На третьей итерации переменная i получает значение 3. Проверка приводит к пропуску тела блока if, и программа завершает свое выполнение, выводя на экран значение 3,14, расположенное в списке data на индексной позиции 2.

Циклы while также можно использовать со списками. Например, в следующем фрагменте кода цикл while используется для определения индекса первого элемента списка с положительным значением. В цикле используется внешняя переменная i, содержащая индекс текущего элемента в списке, начиная с нуля. Значение переменной i увеличивается по ходу выполнения программы, пока не будет достигнут конец списка или найден элемент с положительным значением.

```
# Инициализируем переменные
data = [0, -1, 4, 1, 0]

# Цикл, пока i является допустимым индексом списка и значение по этому индексу
# не положительное
i = 0
while i < len(data) and data[i] <= 0:
    i = i + 1

# Если i меньше длины списка, значит, положительное значение было найдено
# В противном случае i будет равна длине списка, а это означает, что
# положительных чисел в списке нет
if i < len(data):
    print("Первое положительное число в списке располагается по индексу", i)
else:
    print("Список не содержит положительных чисел")
```

Данная программа также начинается с инициализации переменных data и i. После этого сразу запускается цикл while. Переменная i, равная нулю, меньше длины списка, и значение элемента, располагающееся по

этому индексу, меньше или равно нулю, в результате чего в первый раз выполняется тело цикла, в котором `i` увеличивается на единицу.

Управление возвращается к началу цикла `while`, и условие проверяется снова. Результатом по-прежнему будет `True`, и тело цикла выполнится вновь, а переменная `i` сменит значение с единицы на двойку.

На третьем проходе по циклу условное выражение вернет False, поскольку значение в третьей позиции больше нуля. Тело цикла пропускается, и выполнение программы продолжается с условной инструкции `if`, следующей за ним. Поскольку текущее значение `i` меньше, чем длина списка, на экран будет выведена информация о найденном положительном элементе с его индексом.

## 5.3. Дополнительные операции со списками

Списки могут расширяться и сокращаться в процессе выполнения программы. При этом новый элемент может быть вставлен в любое место списка, и также любой элемент из списка может быть удален по значению или по индексу. Кроме того, в Python представлены удобные механизмы для определения того, находится ли элемент в списке, поиска индекса первого вхождения элемента в список, перестановки членов списка и других важных задач.

Добавление и удаление элементов из списка выполняется путем вызова соответствующих методов у объекта, представляющего список. Подобно функциям, *методы* ассоциируются с блоками кода, которые могут быть вызваны применительно к конкретному объекту. При этом синтаксис вызова метода несколько отличается от функций.

Метод списка может быть вызван по имени, которое должно следовать за именем списка и точкой. Он также может быть вызван применительно к безымянному списку элементов, заключенных в квадратные скобки, но такой подход применяется довольно редко. Как и функция, метод сопровождается круглыми скобками после имени, в которых указываются передаваемые аргументы через запятую. Некоторые методы возвращают результат, который может быть присвоен переменной, передан в качестве аргумента в другую функцию или метод либо использоваться как часть вычисления – подобно результату, возвращаемому функцией.

### 5.3.1. Добавление элементов в список

Элементы могут добавляться в конец списка при помощи метода *append*. Метод принимает один аргумент, являющийся элементом, который будет добавлен в список. Рассмотрим следующий фрагмент кода.

```
data = [2.71, 3.14, 1.41, 1.62]
data.append(2.30)
print(data)
```

В первой строке кода создается список с именем `data`, состоящий из четырех элементов. Далее следует вызов метода `append` применительно к списку `data`, в результате чего к концу списка добавляется элемент со значением 2,30, тем самым расширяя длину списка с четырех до пяти. Наконец, в последней строке осуществляется вывод на экран обновленного списка, состоящего из элементов 2,71, 3,14, 1,41, 1,62 и 2,30.

Если необходимо вставить новый элемент в произвольное место в списке, можно воспользоваться методом `insert`. Данный метод требует передачи двух параметров, представляющих индекс, по которому необходимо вставить значение, и сам вставляемый элемент. После вставки элемента в список все его члены, расположенные справа от добавленного значения, обретут новый индекс, на единицу больший предыдущего. Например, в следующем фрагменте кода происходит вставка элемента 2,30 в середину списка `data`, а не в конец. После выполнения этого кода на экран будет выведено новое содержимое списка в виде: [2.71, 3.14, 2.30, 1.41, 1.62].

```
data = [2.71, 3.14, 1.41, 1.62]
data.insert(2, 2.30)
print(data)
```

## 5.3.2. Удаление элементов из списка

Метод *pop* может быть использован для удаления из списка элемента, находящегося в определенной позиции. Индекс удаляемого элемента передается в метод в качестве необязательного параметра. Если этот параметр пропустить, будет удален последний элемент из списка. Метод `pop` возвращает элемент, который был извлечен из списка. Если его значение может понадобиться для проведения последующих расчетов, можно сохранить его в переменную, поставив метод `pop` справа от знака присваивания. Вызов метода `pop` применительно к пустому списку вернет ошибку по причине попытки извлечь из списка элемент с индексом, находящимся за его пределами.

Удалить элемент из списка можно также при помощи вызова метода *remove*. Единственным параметром этого метода является значение удаляемого элемента (в отличие от метода `pop`, который оперирует индексами). При запуске метод `remove` удаляет из списка первое вхождение элемента с указанным значением. Если элемент с таким значением в списке найден не будет, метод вернет ошибку.

Рассмотрим еще один пример. В нем мы создадим список из четырех элементов, после чего удалим два из них. Первый вызов функции `print`

выведет на экран список из оставшихся двух элементов 2,71 и 3,14, поскольку элементы 1,62 и 1,41 были удалены из списка. Далее на экран будет выведено значение 1,41, соответствующее элементу, извлеченному из списка посредством вызова метода pop.

```
data = [2.71, 3.14, 1.41, 1.62]

data.remove(1.62) # Удаляем значение 1.62 из списка
last = data.pop() # Извлекаем последний элемент из списка

print(data)
print(last)
```

## 5.3.3. Изменение порядка следования элементов в списке

Бывает, что в списке содержатся нужные элементы, но расположены они не так, как требуется для решения той или иной задачи. Два элемента в списке можно поменять местами при помощи временной переменной и нескольких инструкций, как показано на примере ниже.

```
# Создаем список
data = [2.71, 3.14, 1.41, 1.62]

# Меняем местами элементы в списке с индексами 1 и 3
temp = data[1]
data[1] = data[3]
data[3] = temp

# Отображаем модифицированный список
print(data)
```

Изначально переменная списка data инициализируется значениями [2.71, 3.14, 1.41, 1.62]. После этого элемент списка с индексом 1, представляющий значение 3,14, присваивается временной переменной temp. Затем значение элемента с индексом 3 отправляется на место элемента с индексом 1, после чего операцию завершает присваивание значения из временной переменной элементу с индексом 3. На выходе мы получаем список с теми же элементами, но в измененном порядке: [2.71, 1.62, 1.41, 3.14].

Есть еще два способа изменить порядок следования элементов в списке, а именно воспользоваться специальными методами *reverse* и *sort*. Первый из них, как ясно из названия, меняет на противоположный порядок, в котором расположены элементы в списке, а второй сортирует элементы в порядке возрастания. Оба упомянутых метода могут быть вызваны

применительно к спискам вовсе без аргументов. Вообще, список может быть отсортирован только в том случае, если его элементы могут сравниваться при помощи оператора <. Этот оператор в Python реализован для сравнения элементов самых разных типов, включая целые числа, числа с плавающей запятой, строки, списки и многие другие.

В следующем примере мы попросим пользователя ввести перечень значений и сохраним их в список. После этого отсортируем список и выведем его на экран.

```
# Создаем пустой список
values = []

# Запрашиваем числа у пользователя и собираем список, пока он не оставит строку пустой
line = input("Введите число (Enter для завершения): ")
while line != "":
    num = float(line)
    values.append(num)
    line = input("Введите число (Enter для завершения: ")

# Сортируем список по возрастанию
values.sort()

# Отображаем значения
for v in values:
    print(v)
```

## 5.3.4. Поиск в списке

Иногда бывает необходимо выяснить, содержится ли тот или иной элемент в заданном списке. В других ситуациях нам требуется получить индекс конкретного элемента в списке, который точно в нем присутствует. Оператор `in` и функция `index` в Python помогут нам справиться с этими задачами.

Оператор *in* используется для определения факта вхождения элемента в список. Искомое значение при этом помещается слева от оператора, а список, в котором будет осуществляться поиск, – справа. Такое булево выражение возвращает `True`, если элемент входит в заданный список, и `False` в обратном случае.

Метод *index* применяется для идентификации позиции искомого элемента в списке, значение которого передается в метод в качестве единственного аргумента. На выходе будет получен индекс первого вхождения элемента в список. Если искомого значения в списке не окажется, метод вернет ошибку. Так что программисты зачастую сначала определяют при помощи оператора `in`, есть ли элемент в списке, а уже затем прибегают к помощи метода `index` для поиска индекса интересующего элемента.

В следующем фрагменте кода мы продемонстрируем сразу несколько методов и операторов, о которых говорили в этой главе. Начинается программа с запроса у пользователя целых чисел и добавления их в список. После этого пользователь должен ввести дополнительное целое число. Если в сформированном списке присутствует этот элемент, на экран будет выведен индекс его первого появления. В противном случае должно появиться сообщение о неудачном поиске.

```
# Запрашиваем целые числа и собираем их в список, пока не будет введена пустая строка
data = []
line = input("Введите целое число (Enter для окончания): ")
while line != "":
    n = int(line)
    data.append(n)
    line = input("Введите целое число (Enter для окончания): ")

# Запрашиваем у пользователя дополнительное число
x = int(input("Введите дополнительное целое число: "))
# Отображаем индекс первого вхождения этого элемента в список (если он там есть)
if x in data:
    print("Первое вхождение", x, "в списке - по индексу:", data.index(x))
else:
    print(x, "не находится в списке")
```

# 5.4. Списки как возвращаемые значения и аргументы

Функции могут возвращать списки. Как и значения других типов, списки из функций возвращаются при помощи ключевого слова `return`. После этого выполнение функции завершается, а управление передается следующей после вызова функции инструкции. Полученный таким образом список может быть сохранен в переменную или использован в расчетах.

Списки также могут быть переданы в функции в качестве аргументов. Здесь тоже нет никаких отличий от значений других типов – имя списка передается в функцию в круглых скобках, как любой другой аргумент. Внутри списка переданный аргумент также сохраняется в переменной соответствующего параметра.

Переменные параметров, представляющие собой списки, могут быть использованы в теле функции так же точно, как переменные любых других типов. Однако, в отличие от целых чисел, чисел с плавающей запятой, булевых выражений и строк, изменения, произведенные над элементами в списке внутри функции, могут повлиять на содержимое списка в вы-

зывающем коде. В частности, изменения, сделанные в переданном списке посредством методов вроде `append`, `pop` или `sort`, окажут влияние на содержимое переданного в функцию аргумента и полученного на входе параметра.

То же самое можно сказать и об изменении отдельных элементов списка путем присвоения им новых значений с указанием в квадратных скобках конкретных индексов. При этом присвоение нового значения всему списку в целом, когда слева от знака равенства находится только имя списка, влияет лишь на переменную параметра внутри функции, тогда как переменная, представляющая входной аргумент, остается неизменной.

Различия в поведении списков в сравнении с переменными других типов применительно к их передаче функциям не случайны. Они обусловлены серьезными техническими доводами, выходящими за рамки данной книги.

# 5.5. Упражнения

Все упражнения из данной главы должны быть решены при помощи списков. В программах, которые вы напишете, будут создаваться списки, модифицироваться, а также по ним будет выполняться поиск. Будут и задания, в которых переменные, хранящие списки, должны передаваться в функции в качестве аргументов или возвращаться из них.

## *Упражнение 110. Порядок сортировки*

(*Решено. 22 строки*)

Напишите программу, которая будет запрашивать у пользователя целочисленные значения и сохранять их в виде списка. Индикатором окончания ввода значений должен служить ноль. Затем программа должна вывести на экран все введенные пользователем числа (кроме нуля) в порядке возрастания – по одному значению в строке. Используйте для сортировки либо метод `sort`, либо функцию `sorted`.

## *Упражнение 111. Обратный порядок*

(*20 строк*)

Напишите программу, которая, как и в предыдущем случае, будет запрашивать у пользователя целые числа и сохранять их в виде списка. Индикатором окончания ввода значений также должен служить ноль. На этот раз необходимо вывести на экран введенные значения в порядке убывания.

## *Упражнение 112. Удаляем выбросы*

(*Решено. 44 строки*)

При анализе собранных по результатам научных экспериментов данных зачастую возникает необходимость избавиться от экстремальных значений, прежде чем продолжать двигаться дальше. Напишите функцию, создающую копию списка с исключенными из него `n` наибольшими и наименьшими значениями и возвращающую ее в качестве результата. Порядок следования элементов в измененном списке не обязательно должен в точности совпадать с источником.

В основной программе должна быть продемонстрирована работа вашей функции. Для начала попросите пользователя ввести целые числа, затем соберите их в список и вызовите написанную вами ранее функцию. Выведите на экран измененную версию списка вместе с оригинальной. Если пользователь введет менее четырех чисел, должно быть отображено соответствующее сообщение об ошибке.

## *Упражнение 113. Избавляемся от дубликатов*

(*Решено. 21 строка*)

В данном упражнении вам предстоит разработать программу, в которой у пользователя будет запрошен список слов, пока он не оставит строку ввода пустой. После этого на экране должны быть показаны слова, введенные пользователем, но без повторов, – каждое по одному разу. При этом слова должны быть отображены в том же порядке, в каком их вводили с клавиатуры. Например, если пользователь на запрос программы введет следующий список слов:

```
first
second
first
third
second
```

программа должна вывести:

```
first
second
third
```

## *Упражнение 114. Отрицательные, положительные и нули*

(*Решено. 36 строк*)

Напишите программу, запрашивающую у пользователя целые числа, пока он не оставит строку ввода пустой. После окончания ввода на экран должны быть выведены сначала все отрицательные числа, которые были вве-

дены, затем нулевые и только после этого положительные. Внутри каждой группы числа должны отображаться в той последовательности, в которой были введены пользователем. Например, если он ввел следующие числа: 3, −4, 1, 0, −1, 0 и −2, вывод должен оказаться таким: −4, −1, −2, 0, 0, 3 и 1. Каждое значение должно отображаться на новой строке.

## *Упражнение 115. Список собственных делителей*

(*36 строк*)

Собственным делителем числа называется всякий его делитель, отличный от самого числа. Напишите функцию, которая будет возвращать список всех собственных делителей заданного числа. Само это число должно передаваться в функцию в виде единственного аргумента. Результатом функции будет перечень собственных делителей числа, собранных в список. Основная программа должна демонстрировать работу функции, запрашивая у пользователя число и выводя на экран список его собственных делителей. Программа должна запускаться только в том случае, если она не импортирована в виде модуля в другой файл.

## *Упражнение 116. Совершенные числа*

(*Решено. 35 строк*)

Целое число *n* называется совершенным, если сумма всех его собственных делителей равна самому числу *n*. Например, 28 – это совершенное число, поскольку его собственными делителями являются 1, 2, 4, 7 и 14, а $1 + 2 + 4 + 7 + 14 = 28$.

Напишите функцию для определения того, является ли заданное число совершенным. Функция будет принимать на вход единственный параметр и возвращать `True`, если он представляет собой совершенное число, и `False` – если нет. Разработайте небольшую программу, которая будет использовать функцию для идентификации и вывода на экран всех совершенных чисел в диапазоне от 1 до 10 000. При решении этой задачи импортируйте функцию, написанную в упражнении 115.

## *Упражнение 117. Только слова*

(*38 строк*)

В данном упражнении вы напишете программу, которая будет выделять слова из строки, введенной пользователем. Начните с создания функции, принимающей на вход единственный строковый параметр. В качестве результата она должна возвращать список слов из строки с удаленными знаками препинания, в число которых должны входить точки, запятые, восклицательный и вопросительный знаки, дефисы, апострофы, двоеточия и точки с запятыми. При этом не нужно избавляться от знаков

препинания, стоящих внутри слова, таких как апостроф, служащий в английском языке для обозначения сокращений. Например, если на вход функции дать строку `"Contractions include: don't, isn't, and wouldn't."`, функция должна вернуть следующий список: `["Contractions", "include", "don't", "isn't", "and", "wouldn't"]`.

В основной программе, как обычно, должна происходить демонстрация вашей функции. Запросите у пользователя строку и выведите на экран все составляющие ее слова с удаленными знаками препинания. Вам понадобятся написанные при решении заданий 118 и 167 функции, так что убедитесь, что основная программа выполняется только в случае, если файл не импортирован в качестве модуля.

## *Упражнение 118. Словесные палиндромы*

(*34 строки*)

В упражнениях 75 и 76 мы уже имели дело со словами, являющимися палиндромами. Тогда мы анализировали буквы в слове с начала и конца, игнорируя пробелы и знаки препинания, чтобы понять, совпадает ли его написание в прямом и обратном направлениях. И хотя палиндромами обычно называют слова, это понятие вполне можно расширить. Например, английская фраза «Is it crazy how saying sentences backwards creates backwards sentences saying how crazy it is?» является словесным палиндромом, поскольку если читать ее по словам, игнорируя при этом знаки препинания и заглавные буквы, в обоих направлениях она будет звучать одинаково. Еще примеры английских словесных палиндромов: «Herb the sage eats sage, the herb» и «Information school graduate seeks graduate school information».

Напишите программу, которая будет запрашивать строку у пользователя и оповещать его о том, является ли она словесным палиндромом. Не забывайте игнорировать знаки препинания при выявлении результата.

## *Упражнение 119. Ниже и выше среднего*

(*44 строки*)

Напишите программу, которая будет запрашивать у пользователя числа, пока он не пропустит ввод. Сначала на экран должно быть выведено среднее значение введенного ряда чисел, после этого друг за другом необходимо вывести список чисел ниже среднего, равных ему (если такие найдутся) и выше среднего. Каждый список должен предваряться соответствующим заголовком.

## *Упражнение 120. Форматирование списка*

(*Решено. 41 строка*)

Обычно при написании перечислений и списков мы разделяем их элементы запятыми, а перед последним ставим союз «и», как показано ниже:

```
яблоки
яблоки и апельсины
яблоки, апельсины и бананы
яблоки, апельсины, бананы и лимоны
```

Напишите функцию, которая будет принимать на вход список из строк и возвращать собранную строку из его элементов в описанной выше манере. Хотя в представленном примере количество элементов списка ограничивается четырьмя, ваша функция должна уметь обрабатывать списки любой длины. В основной программе запросите у пользователя несколько элементов списка, отформатируйте их должным образом при помощи функции и выведите на экран.

## *Упражнение 121. Случайные лотерейные номера*

(*Решено. 28 строк*)

Для выигрыша главного приза необходимо, чтобы шесть номеров на лотерейном билете совпали с шестью числами, выпавшими случайным образом в диапазоне от 1 до 49 во время очередного тиража. Напишите программу, которая будет случайным образом подбирать шесть номеров для вашего билета. Убедитесь в том, что среди этих чисел не будет дубликатов. Выведите номера билетов на экран по возрастанию.

## *Упражнение 122. «Поросячья латынь»*

(*32 строки*)

«Поросячьей латынью» называют молодежный жаргонный язык, производный от английского. И хотя корни этого новообразованного языка неизвестны, упоминание о нем есть как минимум в двух документах, датированных XIX веком, а это значит, что ему уже больше сотни лет. Для перевода слова с английского на «поросячью латынь» нужно сделать следующее:

- если слово начинается с согласной буквы (включая `y`), то все буквы с начала слова и до первой гласной (за исключением `y`) переносятся в конец слова и дополняются сочетанием букв `ay`. Например, слово `computer` будет преобразовано в `omputercay`, а слово `think` – в `inkthay`;
- если слово начинается с гласной буквы (не включая `y`), к концу слова просто добавляется `way`. К примеру, слово `algorithm` превратится в `algorithmway`, а `office` – в `officeway`.

Напишите программу, которая будет запрашивать у пользователя строку. После этого она должна переводить введенный текст на «поросячью латынь» и выводить его на экран. Вы можете сделать допуск о том, что все слова пользователь будет вводить в нижнем регистре и разделять их пробелами.

## Упражнение 123. «Поросячья латынь» (продолжение)

(*51 строка*)

Расширьте свое решение упражнения 122, чтобы ваш анализатор корректно обрабатывал символы в верхнем регистре и знаки препинания, такие как запятая, точка, а также восклицательный и вопросительный знаки. Если в оригинале слово начинается с заглавной буквы, то в переводе на «поросячью латынь» оно также должно начинаться с заглавной буквы, тогда как буквы, перенесенные в конец слов, должны стать строчными. Например, слово `Computer` должно быть преобразовано в `Omputercay`. Если в конце слова стоит знак препинания, он там же и должен остаться после выполнения перевода. То есть слово в конце предложения `Science!` необходимо трансформировать в `Iencescay!`.

## Упражнение 124. Линия наилучшего соответствия

(*41 строка*)

Линией наилучшего соответствия называется прямая, проходящая на наименьшем удалении от набора из $n$ точек. В данном упражнении мы предположим, что каждая точка в коллекции обладает координатами $x$ и $y$. Символы $\bar{x}$ и $\bar{y}$ мы будем использовать для подсчета средних значений по осям $x$ и $y$ соответственно. Линия наилучшего соответствия представлена формулой $y = mx + b$, где $m$ и $b$ вычисляются по следующим формулам:

$$m = \frac{\sum xy - \frac{\left(\sum x\right)\left(\sum y\right)}{n}}{\sum x^2 - \frac{\left(\sum x\right)^2}{n}};$$

$$b = \bar{y} - m\bar{x}.$$

Напишите программу, которая будет запрашивать у пользователя координаты коллекции точек. При этом пользователь должен вводить сначала координату $x$, а затем $y$. Ввод координат может продолжаться до тех пор, пока пользователь не оставит пустым ввод координаты $x$. Отобразите формулу, характеризующую линию наилучшего соответствия, вида $y = mx + b$ путем замены переменных $m$ и $b$ на значения, вычисленные по предыдущим формулам. Например, если пользователь введет три точки (1, 1), (2, 2.1) и (3, 2.9), итоговая формула должна приобрести вид $y = 0{,}95x + 0{,}1$.

## *Упражнение 125. Тасуем колоду карт*

(*Решено. 49 строк*)

Стандартная игральная колода состоит из 52 карт. Каждая карта соответствует одной из четырех мастей (пики, червы, бубны и трефы) и одному из 13 номиналов (от 2 до 10, валет (J), дама (Q), король (K) и туз (A)).

Таким образом, каждая игральная карта может быть представлена при помощи двух символов. Первый из них указывает на номинал карты (от 2 до 9, T (десятка), J, Q, K или A), а второй – на масть ($s$ = пики (spades), $h$ = червы (hearts), $d$ = бубны (diamonds) и $c$ = трефы (clubs)). В табл. 5.1 представлены некоторые из возможных обозначений игральных карт.

***Таблица 5.1. Игральные карты***

| Карта | Обозначение |
|---|---|
| Валет пик | Js |
| Двойка треф | 2c |
| Десятка бубен | Td |
| Туз червей | Ah |
| Девятка пик | 9s |

Начните с написания функции `createDeck`. В ней должны использоваться циклы для создания полной колоды карт путем сохранения в список двухсимвольных аббревиатур всех 52 карт. Именно этот список и будет возвращаемым из данной функции значением. На вход функция `createDeck` принимать параметры не будет.

Напишите вторую функцию с именем `shuffle`, которая будет случайным образом перетасовывать карты в списке. Одна из техник тасования колоды заключается в проходе по элементам и перестановке их с любым другим случайным элементом в этом списке. Вы должны создать свой собственный цикл для тасования карт в колоде, а не пользоваться стандартной функцией shuffle языка Python.

Используйте обе созданные функции в основной программе, в которой должна отображаться колода карт до и после тасования. Убедитесь, что основная программа выполняется только в случае, если файл не импортирован в качестве модуля.

**Примечание.** Хороший алгоритм тасования игральной колоды должен быть беспристрастным, что означает равную вероятность расположения каждой из карт в колоде после тасования. Однако алгоритм, предложенный в этом упражнении и предполагающий обмен позициями между каждой из карт в колоде с любой другой случайной

картой, не является таковым. В частности, карты, которые появляются позже в исходном списке, с большой вероятностью окажутся ближе к концу и в перетасованном списке. Как это ни странно, беспристрастной будет версия алгоритма, в которой при последовательном проходе по элементам каждый из них будет меняться позициями не со случайным элементом из всего списка, а со случайным элементом в диапазоне от позиции текущей карты и до конца колоды.

## *Упражнение 126. Раздача карманных карт*

(*44 строки*)

Во многих карточных играх после процедуры тасования колоды каждый игрок получает на руки определенное количество карт. Напишите функцию `deal`, принимающую на вход три параметра: количество игроков, количество раздаваемых каждому из них карт и саму колоду. Функция должна возвращать список рук, которые были розданы игрокам. При этом каждая рука, в свою очередь, тоже является списком из входящих в нее карт.

Во время раздачи карт игрокам функция параллельно должна удалять розданные карты из переданной ей третьим параметром колоды. Также принято раздавать карты каждому игроку по одной строго по очереди. Придерживайтесь этих принципов и при написании своей функции.

Воспользуйтесь своими наработками из упражнения 125 при построении структуры основной программы. Вам необходимо создать колоду карт, перетасовать ее и раздать четырем игрокам по пять карт. Выведите на экран карманные карты всех игроков, находящихся в раздаче, а также оставшиеся в колоде карты.

## *Упражнение 127. Список уже отсортирован?*

(*41 строка*)

Напишите функцию, показывающую, отсортирован ли переданный ей в качестве параметра список (по возрастанию или убыванию). Функция должна возвращать `True`, если список отсортирован, и `False` в противном случае. В основной программе запросите у пользователя последовательность чисел для списка, после чего выведите сообщение о том, является ли этот список отсортированным изначально.

**Примечание.** Убедитесь в том, что вы правильно обрабатываете пустые списки, а также списки, состоящие из единственного элемента.

## *Упражнение 128. Подсчитать элементы в списке*

(*Решено. 48 строк*)

В стандартной библиотеке языка Python присутствует функция `count`, позволяющая подсчитать, сколько раз определенное значение встречается в списке. В данном упражнении вы создадите новую функцию `countRange`, которая будет подсчитывать количество элементов в списке, значения которых больше или равны заданному минимальному порогу и меньше максимального. Функция должна принимать три параметра: список, минимальную границу и максимальную границу. Возвращать она будет целочисленное значение, большее или равное нулю. В основной программе реализуйте демонстрацию вашей функции для нескольких списков с разными минимальными и максимальными границами. Удостоверьтесь, что программа будет корректно работать со списками, содержащими как целочисленные значения, так и числа с плавающей запятой.

## *Упражнение 129. Разбиение строки на лексемы*

(*Решено. 47 строк*)

*Разбиение строки на лексемы* (Tokenizing) представляет собой процесс преобразования исходной строки в список из подстрок, называемых *лексемами* (token). Зачастую со списком лексем работать бывает проще, чем со всей исходной строкой, поскольку в ней могут присутствовать неравномерные разрывы. Кроме того, иногда бывает непросто на лету определить, где заканчивается одна лексема и начинается другая.

В математических выражениях лексемами являются, например, операторы, числа и скобки. Здесь и далее мы будем причислять к списку операторов следующие: *, /, ^, – и +. Операторы и скобки легко идентифицировать, поскольку эти лексемы всегда состоят ровно из одного символа и никогда не являются составной частью других лексем. Числа выделить бывает сложнее, поскольку эти лексемы могут состоять из нескольких символов. Любая непрерывная последовательность цифр должна восприниматься как одна числовая лексема.

Напишите функцию, принимающую в качестве единственного входного параметра строку, содержащую математическое выражение, и преобразующую ее в список лексем. Каждая лексема должна быть либо оператором, либо числом, либо скобкой. Для простоты реализации в данном упражнении мы будем оперировать только целочисленными значениями. Функция должна возвращать созданный список лексем.

При решении поставленной задачи вы можете принять допущение о том, что входная строка всегда будет содержать математическое выражение, состоящее из скобок, чисел и операторов. При этом в вашей функции должно быть предусмотрено, что лексемы могут отделяться друг от друга разным количеством пробелов, а могут и не отделяться вовсе. В основной

программе продемонстрируйте работу функции, запросив у пользователя исходную строку и выведя на экран список составляющих ее лексем. Убедитесь, что основная программа выполняется только в случае, если файл не импортирован в качестве модуля.

## *Упражнение 130. Унарные и бинарные операторы*

(*Решено. 45 строк*)

Математические операторы бывают *унарными* (unary) и *бинарными* (binary). Унарные операторы взаимодействуют с одним значением, тогда как бинарные – с двумя. Например, в выражении `2 * -3` оператор `*` является бинарным, поскольку взаимодействует с двумя числами: 2 и −3. При этом сам оператор `-` здесь унарный, ведь он применяется только к одному числу 3.

Одного лишь символа оператора недостаточно, чтобы определить, является ли он унарным или бинарным. Например, хотя в предыдущем случае оператор `-` был унарным, в выражении `2 - 3` он приобретет роль бинарного. Подобная неоднозначность, также характерная для оператора сложения, должна быть устранена до применения других операций к элементам списка лексем математического выражения.

Напишите функцию для поиска унарных операторов `+` и `-` в списке лексем и их замены на сочетание символов `u+` и `u-` соответственно. Функция должна принимать в качестве единственного параметра список лексем математического выражения и возвращать его копию с произведенной заменой унарных операторов. Оператор `+` или `-` можно идентифицировать как унарный в одном из двух случаев: если он идет первым в списке или если ему предшествует другой оператор либо открывающая скобка. Во всех остальных случаях оператор может считаться бинарным.

В основной программе продемонстрируйте работу функции. Запросите у пользователя строку с математическим выражением, разбейте ее на лексемы, выделите в отдельный список унарные операторы и выведите их на экран.

## *Упражнение 131. Инфиксная запись – в постфиксную*

(*63 строки*)

Математические выражения часто записываются в *инфиксной форме* (infix form), когда оператор ставится между операндами, с которыми взаимодействует. И хотя такая форма записи наиболее распространена, существует и другая, именуемая *постфиксной* (postfix form), в которой оператор ставится после операндов. Например, инфиксной форме записи выражения `3 + 4` будет соответствовать постфиксный вариант `3 4 +`. Чтобы преобразовать инфиксную форму записи в постфиксную, необходимо выполнить следующий алгоритм действий.

```
Создаем новый пустой список operators
Создаем новый пустой список postfix

Для каждой лексемы в инфиксном выражении
    Если лексема представляет собой целое число, то
        Добавляем лексему к списку postfix
    Если лексема представляет собой оператор, то
        Пока список operators не пустой и
                последний элемент в operators не открывающая скобка и
                precedence(лексема) < precedence(последний элемент в operators), делаем
            Удаляем последний элемент из списка operators и добавляем его к postfix
        Добавляем лексему к списку operators
    Если лексема представляет собой открывающую скобку, то
        Добавляем лексему к списку operators
    Если лексема представляет собой закрывающую скобку, то
        Пока последний элемент в operators не является открывающей скобкой, делаем
            Удаляем последний элемент из списка operators и добавляем его к postfix
        Удаляем открывающую скобку из operators

Пока список operators не пустой, делаем
    Удаляем последний элемент из списка operators и добавляем его к postfix

Возвращаем postfix в качестве результата алгоритма
```

Используйте свои наработки из упражнений 129 и 130 для разделения математических выражений на лексемы и поиска в них унарных операторов. После этого используйте алгоритм, приведенный выше, для преобразования выражения из инфиксной формы в постфиксную. Код, реализующий этот алгоритм, должен быть заключен в функцию, принимающую на вход список лексем инфиксного выражения (с помеченными унарными операторами). Возвращать функция будет список лексем в постфиксном выражении. В основной программе продемонстрируйте работу функции по преобразованию инфиксной формы записи математического выражения в постфиксную. Запросите у пользователя выражение инфиксного типа и выведите на экран его постфиксный аналог.

Цель перевода математического выражения из одного вида в другой будет понятна вам, когда вы прочитаете текст упражнения 132. Кроме того, вам могут понадобиться наработки из заданий 96 и 97 при решении этого упражнения. И если первое решение вы можете использовать как есть, то решение задания 97 необходимо будет немного расширить, чтобы возвращался правильный приоритет для унарных операторов. Унарные операторы должны обладать более высоким приоритетом по сравнению с операциями умножения и деления, но более низким, если сравнивать с операцией возведения в степень.

## *Упражнение 132. Выполнение постфиксных выражений*

(*63 строки*)

Математические выражения, записанные в постфиксной форме, выполнять легче, чем те же выражения в инфиксной, поскольку в них нет скобок и не нужно учитывать старшинство операторов. Выражения в постфиксной форме могут быть выполнены при помощи реализации следующего алгоритма.

```
Создаем новый пустой список values

Для каждой лексемы в постфиксном выражении
    Если лексема представляет собой целое число, то
        Преобразуем лексему в целочисленный тип и добавляем к списку values
    Если лексема представляет собой унарный оператор -, то
        Удаляем последний элемент из списка values
        Применяем операцию логического НЕ к элементу и добавляем результат к списку values
    Если лексема представляет собой бинарный оператор, то
        Удаляем последний элемент из списка values и называем его right
        Удаляем последний элемент из списка values и называем его left
        Вычисляем результат применения оператора к операндам left и right
        Добавляем результат к списку values
Возвращаем первый элемент списка values в качестве значения выражения
```

Напишите программу, запрашивающую у пользователя математическое выражение в инфиксном виде, преобразующую его в постфиксную форму, выполняющую полученное выражение и выводящую на экран результат. Используйте при решении задачи свои наработки из упражнений 129, 130 и 131, а также алгоритм, приведенный выше.

> **Примечание.** В алгоритмах, предложенных для решения упражнений 131 и 132, не предусмотрена обработка возможных ошибок. В результате ваши программы могут выдавать ошибки или выполняться неправильно, если пользователь введет что-то неожиданное. Эти алгоритмы могут быть расширены и дополнены по желанию во избежание возникновения ошибок. Сами ошибки можно корректно обрабатывать и выводить соответствующие сообщения на экран. Если вам интересно попробовать это сделать, никто вас останавливать не будет.

## *Упражнение 133. Содержит ли список подмножество элементов?*

(*44 строки*)

Подмножеством элементов, или *подсписком* (sublist), мы будем называть список, являющийся составной частью большего списка. Подсписок может содержать один элемент, множество элементов, а также быть пустым.

Например, [1], [2], [3] и [4] являются подсписками списка [1, 2, 3, 4]. Список [2, 3] также входит в состав [1, 2, 3, 4], но при этом список [2, 4] не является подсписком [1, 2, 3, 4], поскольку в исходном списке числа 2 и 4 не соседствуют друг с другом. Пустой список может быть рассмотрен как подсписок для любого списка. Таким образом, список [] является подсписком [1, 2, 3, 4]. Также список является подсписком самого себя, то есть [1, 2, 3, 4] – это подсписок для [1, 2, 3, 4].

В рамках данного упражнения вам необходимо написать функцию isSublist, определяющую, является ли один список подсписком другого. На вход функции должны поступать два списка – larger и smaller. Функция должна возвращать значение True только в том случае, если список smaller является подсписком списка larger. Напишите также основную программу для демонстрации работы функции.

## *Упражнение 134. Все подсписки заданного списка*

(*Решено. 41 строка*)

Используя определение подсписка из упражнения 133, напишите функцию, возвращающую список, содержащий все возможные подсписки заданного. Например, в число подсписков списка [1, 2, 3] входят следующие: [], [1], [2], [3], [1, 2], [2, 3] и [1, 2, 3]. Заметьте, что ваша функция должна вернуть как минимум один пустой список, гарантированно являющийся подсписком для любого списка. Напишите основную программу, демонстрирующую работу функции применительно к нескольким исходным спискам.

## *Упражнение 135. Решето Эратосфена*

(*Решено. 33 строки*)

Решето Эратосфена – алгоритм, изобретенный более 2000 лет назад и служащий для нахождения всех простых чисел от 2 до некоторого целого числа *n*. Описание этого алгоритма приведено ниже.

```
Выписываем все целые числа от 0 до заданной границы
Вычеркиваем 0 и 1 как непростые числа

Устанавливаем значение переменной p, равное 2
Пока p меньше указанного числа, делать
    Вычеркиваем все числа, кратные p, но не его само
    Устанавливаем значение p, равное следующему невычеркнутому числу

Выводим все числа, оставшиеся незачеркнутыми
```

Ценность данного алгоритма заключается в том, что на бумаге очень легко вычеркнуть все числа, кратные определенному. Для компьютера это также не самая сложная задача – с этим может прекрасно справиться

инструкция `for` с третьим параметром, переданным функции `range`. Мы знаем, что вычеркнутые числа на листочке не являются простыми, но физически они никуда с листа не деваются и должны участвовать в дальнейшем алгоритме. Так что и в компьютерной симуляции не стоит «вычеркивать» элемент путем его удаления из списка – вместо этого лучше будет присвоить ему значение 0. После завершения алгоритма все ненулевые числа в списке и будут простыми.

Напишите программу на Python, реализующую указанный выше алгоритм для отображения простых чисел в интервале от двух до значения, введенного пользователем. Если алгоритм будет реализован правильно, ваша программа справится с выводом всех простых чисел от двух до миллиона всего за пару секунд.

**Примечание.** Приведенный в данном упражнении алгоритм поиска простых чисел, названный в честь Эратосфена, был далеко не единственным вкладом греческого математика в науку. Ему также приписывают вычисление длины окружности Земли и градус наклона ее оси. Кроме того, с 235 г. до н. э. он служил хранителем знаменитой Александрийской библиотеки.

# Глава 6

# Словари

Между списками и словарями много общего. Как и списки, *словари* (dictionary) позволяют хранить большое количество однородной информации в одной переменной. Каждый элемент списка обладает собственным уникальным целочисленным индексом, и эти индексы располагаются по возрастанию, начиная с нуля. Так же и в словарях каждое *значение* (value) строго ассоциировано с *ключом* (key), при этом ключи обладают гораздо большей гибкостью по сравнению с индексами списков. Ключи словарей могут быть как целочисленными, так и числами с плавающей запятой или строками. Еще стоит отметить, что числовые ключи в словарях совсем не обязательно должны начинаться с нуля и располагаться в строгой последовательности по возрастанию. Будучи строковыми, ключи могут содержать любую текстовую информацию, включая пустые строки. Единственное условие, объединяющее индексы в списках с ключами в словарях, состоит в их уникальности.

Каждый ключ в словаре должен быть ассоциирован ровно с одним значением, которое может быть целочисленным, числом с плавающей запятой, строкой или булевым значением. Кроме того, значение само по себе может представлять список или словарь. Ключ словаря с ассоциированным с ним значением часто упоминается как *пара ключ-значение* (key-value pair). Хотя ключи в словаре должны быть строго уникальными, подобное ограничение не распространяется на его значения. Следовательно, одинаковые значения могут быть ассоциированы с разными ключами одного и того же словаря.

Начиная с версии Python 3.7 все пары ключ-значение в словарях хранятся в том порядке, в котором были добавлены. В более ранних версиях Python такая зависимость не была гарантирована. Каждая новая пара ключ-значение при добавлении в словарь отправляется в его конец. Механизма для того, чтобы вставить новую пару в середину словаря, просто не существует. Удаление любой пары ключ-значение не оказывает влияния на порядок следования оставшихся элементов в словаре.

Переменная, хранящая словарь, создается так же, как и остальные, – при помощи оператора присваивания. Пустой словарь, не содержащий пар

ключ-значение, может быть объявлен как сочетание открывающей и закрывающей фигурных скобок – {}. Если вам необходимо создать непустой словарь, перечислите в фигурных скобках пары ключ-значение через запятую. При этом сами ключи и ассоциированные с ними значения должны разделяться двоеточием. В следующем фрагменте кода создается словарь с тремя парами ключ-значение, где ключ – это строка, а значение – число с плавающей запятой. Каждая пара в этом словаре представляет конкретную математическую константу. После создания словаря можно вывести его на экран при помощи стандартной функции `print`.

```
constants = {"pi": 3.14, "e": 2.71, "root 2": 1.41}
print(constants)
```

# 6.1. Чтение, добавление и изменение словарей

Получить доступ к значению в словаре можно подобно тому, как мы обращались к элементам списков. Если индекс нужного элемента в списке нам известен, для обращения к нему достаточно написать имя списка вместе с этим индексом, заключенным в квадратные скобки. Так же и со словарями – при известном ключе можно получить доступ к ассоциированному с ним значению посредством указания имени словаря и ключа в тех же квадратных скобках.

Изменение значения в словаре и добавление новой пары ключ-значение выполняются при помощи оператора присваивания. При этом имя словаря с ключом в квадратных скобках помещается слева от знака равенства, а ассоциированное с этим ключом значение – справа. Если элемент с таким ключом уже существует в словаре, его значение будет перезаписано. В противном случае будет создана новая пара ключ-значение. Рассмотрим эти операции на примере.

```
# Создадим словарь с двумя парами ключ-значение
results = {"pass": 0, "fail": 0}

# Добавим третью пару ключ-значение
results["withdrawal"] = 1

# Обновим значения двух ключей в словаре
results["pass"] = 3
results["fail"] = results["fail"] + 1

# Выведем значения, ассоциированные с ключами fail, pass и withdrawal соответственно
print(results["fail"])
```

```
print(results["pass"])
print(results["withdrawal"])
```

При запуске данной программы будет создан словарь `results` с двумя ключами: `pass` и `fail`. Значения, ассоциированные с этими ключами, мы сделали нулевыми. Далее словарь пополняется еще одной парой с ключом `withdrawal` и значением 1. После этого значение, ассоциированное с ключом `pass`, меняется на 3 при помощи оператора присваивания. В следующей строке происходит обращение к значению ключа `fail`, к которому прибавляется единица, и новое значение сохраняется в паре с тем же ключом `fail`, заменяя предыдущее значение. При отображении значений первой будет выведена единица (значение, ассоциированное с ключом `fail`), затем тройка, соответствующая ключу `pass`, и снова единица – на этот раз представляющая значение ключа `withdrawal`.

## 6.2. Удаление пары ключ-значение

Пару ключ-значение из словаря можно удалить при помощи знакомого уже нам метода `pop`. В качестве аргумента этому методу должен быть передан ключ элемента, который требуется удалить. При выполнении метод удалит из списка как ключ, так и ассоциированное с ним значение. В отличие от списка, извлечь последнюю пару ключ-значение из словаря при помощи вызова метода `pop` без аргументов невозможно.

Метод `pop` возвращает значение, ассоциированное с ключом удаленной пары из словаря. Это значение может быть сохранено в переменной посредством оператора присваивания или использовано в других целях – например, в арифметическом выражении или для передачи в качестве аргумента в другую функцию либо метод.

## 6.3. Дополнительные операции со словарями

После заполнения словаря парами ключ-значение может понадобиться выполнить различные действия с данными. Например, вам может потребоваться информация о том, сколько всего пар ключ-значение находится в словаре и присутствует ли в нем пара с конкретным ключом или значением. В Python реализовано сразу несколько функций, методов и операторов, позволяющих извлекать нужные сведения о словарях.

К примеру, функция `len`, которой мы пользовались для определения размера списка, вполне применима и к словарям – она выдает текущее

количество пар ключ-значение, присутствующих в словаре. В качестве единственного параметра функции следует передать переменную, представляющую словарь, и на выходе мы получим количество пар ключ-значение в указанном словаре. Если словарь на данный момент является пустым, функция вернет ноль.

Также знакомый нам оператор `in` может быть использован для определения того, входит ли в словарь интересующий нас ключ или значение. Для поиска ключа в словаре достаточно написать условное выражение, поместив ключ слева от оператора `in`, а имя словаря – справа. Оператор вернет `True` при наличии искомого ключа в словаре и `False` в обратном случае. Результат оператора `in` может быть использован везде, где допустимо применять булевы значения, например в условных блоках инструкций `if` или `while`.

Для определения наличия в словаре нужного вам значения необходимо использовать оператор `in` совместно с методом `values`. В этом случае искомое значение помещается слева от оператора `in`, а имя словаря с примененным к нему методом `values` – справа. В следующем фрагменте кода мы определим, присутствует ли в списке значений словаря `d` значение, находящееся в переменной `x`.

```
if x in d.values():
    print("В словаре d есть как минимум одно значение", x)
else:
    print("В словаре d не присутствует значение", x)
```

# 6.4. Циклы и словари

Цикл `for` может быть использован для осуществления итераций по ключам словаря, как показано ниже. На каждой итерации ключ словаря сохраняется во внутренней переменной цикла `k`.

```
# Создаем словарь
constants = {"pi": 3.14, "e": 2.71, "root 2": 1.41}

# Выводим на экран все ключи и значения в отформатированном виде
for k in constants:
    print("Значение, ассоциированное с ключом", k, ": ", constants[k])
```

Сначала создается словарь с именем `constants`, в котором хранятся некоторые математические константы в виде пар ключ-значение. Цикл `for` открывает итерации по словарю. На первой итерации ключ `pi` сохраняется во временную переменную цикла `k`, и выполняется тело цикла с выводом на экран первой пары ключ-значение из списка. После этого цикл запускается вновь, и в переменную `k` уже попадает ключ `e`. В теле цикла

на экран выводится соответствующая строка со значением 2,71. На заключительном проходе по словарю мы узнаем, что квадратный корень из двух равен 1,41.

Циклом `for` также можно воспользоваться для осуществления итераций по значениям словаря. Это легко реализовать при помощи метода `values`, не принимающего аргументов и создающего коллекцию значений словаря, к которому он применен. В следующем фрагменте кода мы рассчитаем сумму всех сохраненных в нашем словаре математических констант. Выражение `constants.values()` вернет коллекцию, состоящую из значений 3,14, 2,71 и 1,41. Каждое из этих значений во время итераций сохраняется во временной переменной цикла `v`, что позволяет рассчитать их сумму вовсе без обращения к ключам.

```
# Создаем словарь
constants = {"pi": 3.14, "e": 2.71, "root 2": 1.41}

# Рассчитаем сумму значений в словаре
total = 0
for v in constants.values():
    total = total + v

# Выводим результат
print("Сумма значений составляет", total)
```

Иногда бывает удобнее обращаться к словарям при помощи циклов `while`, а не `for`. В следующем фрагменте кода пользователь должен ввести пять уникальных значений с клавиатуры, после чего на экране будут показаны все введенные данные со значением счетчика, который увеличивается на единицу всякий раз, когда пользователь вводит строку, уже присутствующую в словаре.

```
# Считаем, сколько раз пользователь ввел каждое значение
counts = {}

# Цикл, пока количество уникальных значений в словаре не достигнет пяти
while len(counts) < 5:
    s = input("Введите строку: ")

    # Если в словаре уже есть такой ключ, увеличиваем count на 1.
    # Иначе добавляем пару к словарю со значением count, равным 1.
    if s in counts:
        counts[s] = counts[s] + 1
    else:
        counts[s] = 1

# Выводим все строки и счетчики
for k in counts:
    print(k, "появилась в словаре", counts[k], "раз")
```

Программа начинается с создания пустого словаря. После этого запускается цикл `while` с проверкой на количество пар ключ-значение в словаре. Поскольку словарь пока пуст, условное выражение вернет `True`, и будет запущено тело цикла.

На каждой итерации пользователь будет вводить строку с клавиатуры. После этого при помощи оператора `in` будет определено, присутствует ли введенный пользователем ключ в списке. Если да, то ассоциированный с этим ключом счетчик (значение) будет увеличен на единицу. В противном случае словарь пополнится новой парой ключ-значение. Цикл будет продолжаться, пока пользователь не введет пять уникальных строк с клавиатуры. После этого на экран будут выведены все пары ключ-значение из списка.

## 6.5. Словари как аргументы и возвращаемые значения функций

Словари могут быть переданы в функцию подобно значениям других типов. Как и в случае со списками, изменение переменной параметра, хранящей переданный словарь, может привести к модификации источника. Например, удаление, изменение или добавление пары ключ-значение неизбежно приведет к соответствующему изменению аргумента. В то же время присвоение переменной параметра другого значения (когда слева от знака равенства указывается только переменная без квадратных скобок) аргумента не затронет. Как и в случае с переменными других типов, функция может возвращать словарь при помощи ключевого слова `return`.

## 6.6. Упражнения

Несмотря на то что многие упражнения из данного раздела могут быть решены при помощи списков и выражений `if`, большинство из них имеют решения и с применением словарей. Поскольку мы в этой главе говорим главным образом о словарях, используйте их в своих программах вместе с другими конструкциями языка Python, которые вы изучили ранее в книге.

### *Упражнение 136. Поиск по значению*

(*Решено. 45 строк*)

Напишите функцию с названием `reverseLookup`, которая будет осуществлять поиск всех ключей в словаре по заданному значению. Функция должна принимать в качестве параметров словарь и значение для поиска

и возвращать список ключей (он может быть пустым) из этого словаря, соответствующих переданному значению.

В основной программе продемонстрируйте работу функции путем создания словаря и поиска в нем всех ключей по заданному значению. Убедитесь, что функция работает корректно при наличии нескольких ключей для искомого значения, одного ключа и их отсутствии. Ваша программа должна запускаться только в том случае, если она не импортирована в виде модуля в другой файл.

## Упражнение 137. Две игральные кости

(*Решено. 43 строки*)

В данном упражнении мы будем симулировать 1000 выбрасываний игральных костей. Начнем с написания функции, выполняющей случайное выбрасывание двух обычных шестигранных костей. Эта функция не будет принимать входных параметров, а возвращать должна число, выпавшее в сумме на двух костях.

В основной программе реализуйте симуляцию тысячи выбрасываний костей. Программа должна хранить все результаты с частотой их выпадения. После завершения процесса должна быть показана итоговая таблица с результатами, похожая на ту, что представлена в табл. 6.1. Выразите частоту выпадения каждого из чисел в процентах вместе с ожидаемым результатом согласно теории вероятностей.

***Таблица 6.1. Выбрасывание игральных костей***

| Исход | Процент симуляции | Ожидаемый процент |
|---|---|---|
| 2 | 2,90 | 2,78 |
| 3 | 6,90 | 5,56 |
| 4 | 9,40 | 8,33 |
| 5 | 11,90 | 11,11 |
| 6 | 14,20 | 13,89 |
| 7 | 14,20 | 16,67 |
| 8 | 15,00 | 13,89 |
| 9 | 10,50 | 11,11 |
| 10 | 7,90 | 8,33 |
| 11 | 4,50 | 5,56 |
| 12 | 2,60 | 2,78 |

## Упражнение 138. Текстовые сообщения

(*21 строка*)

Если помните, на старых мобильных телефонах текстовые сообщения набирались при помощи цифровых кнопок. При этом одна кнопка была ас-

социирована сразу с несколькими буквами, а выбор зависел от количества нажатий на кнопку. Однократное нажатие приводило к появлению первой буквы в соответствующем этой кнопке списке, последующие нажатия меняли ее на следующую. Список символов, ассоциированных с цифровой панелью, приведен в табл. 6.2.

***Таблица 6.2. Символы, соответствующие кнопкам на старых телефонах***

| Кнопка | Символы |
|---|---|
| 1 | . , ? ! : |
| 2 | A B C |
| 3 | D E F |
| 4 | G H I |
| 5 | J K L |
| 6 | M N O |
| 7 | P Q R S |
| 8 | T U V |
| 9 | W X Y Z |
| 0 | Пробел |

Напишите программу, отображающую последовательность кнопок, которую необходимо нажать, чтобы на экране телефона появился текст, введенный пользователем. Создайте словарь, сопоставляющий символы с кнопками, которые необходимо нажать, а затем воспользуйтесь им для вывода на экран последовательности кнопок в соответствии с введенным пользователем сообщением по запросу. Например, на ввод строки `Hello, World!` ваша программа должна откликнуться следующим выводом: `443355 555566611096667775553111`1. Удостоверьтесь, что ваша программа корректно обрабатывает строчные и прописные буквы. При преобразовании букв в цифры игнорируйте символы, не входящие в указанный перечень, такие как точка с запятой или скобки.

## *Упражнение 139. Азбука Морзе*

(*15 строк*)

Азбука Морзе зашифровывает буквы и цифры при помощи точек и тире. В данном упражнении вам необходимо написать программу, в которой соответствие символов из азбуки Морзе будет храниться в виде словаря. В табл. 6.3 приведена та часть азбуки, которая вам понадобится при решении этого задания.

В основной программе вам необходимо запросить у пользователя строку. После этого программа должна преобразовать его в соответствующую последовательность точек и тире, вставляя пробелы между отдельными

символами. Символы, не представленные в таблице, можно игнорировать. Например, сообщение `Hello, World!` может быть представлено следующей последовательностью: …. . .–.. .–.. ––– .–– ––– .–. .–.. –..

***Таблица 6.3. Азбука Морзе***

| Символ | Код | Символ | Код | Символ | Код | Символ | Код |
|---|---|---|---|---|---|---|---|
| A | .– | J | .––– | S | … | 1 | .–––– |
| B | –… | K | –.– | T | – | 2 | ..––– |
| C | –.–. | L | .–.. | U | ..– | 3 | …–– |
| D | –.. | M | –– | V | …– | 4 | ….– |
| E | . | N | –. | W | .–– | 5 | ….. |
| F | ..–. | O | ––– | X | –..– | 6 | –…. |
| G | ––. | P | .––. | Y | –.–– | 7 | ––… |
| H | …. | Q | ––.– | Z | ––.. | 8 | –––.. |
| I | .. | R | .–. | 0 | ––––– | 9 | ––––. |

**Примечание.** Азбука Морзе была изобретена в XIX веке для передачи информации посредством телеграфа. Она широко используется и сегодня, более чем через 160 лет после ее создания.

## *Упражнение 140. Почтовые индексы*

(*24 строки*)

Первый, третий и пятый символы в канадском почтовом индексе представляют собой буквы, а второй, четвертый и шестой – цифры. Провинцию или территорию, которой принадлежит индекс, можно определить по первому символу индекса, как показано в табл. 6.4. Символы D, F, I, O, Q, U, W и Z в настоящее время не используются в почтовых индексах Канады.

Второй символ в почтовом индексе определяет, расположен ли интересующий нас адрес в городе или в сельской местности. Если на этом месте стоит ноль, значит, это сельская местность, иначе город.

Напишите программу, которая будет запрашивать почтовый индекс у пользователя и отображать провинцию или территорию, которой он принадлежит, с указанием того, городская это территория или сельская. Например, если пользователь введет индекс `T2N1N4`, программа должна определить, что речь идет о городе на территории провинции Альберта. А индекс `X0A1B2` соответствует сельской местности в провинции Нунавут или в Северо-Западных территориях. Используйте словарь для хранения информации о соответствии первого символа индекса конкретной провинции или территории. Выведите на экран соответствующее сообщение

об ошибке, если индекс начинается с символа, который не используется для этих целей, или второй символ не является цифрой.

***Таблица 6.4. Почтовые индексы Канады***

| Провинция/территория | Первая буква (буквы) индекса |
|---|---|
| Ньюфаундленд | A |
| Новая Шотландия | B |
| Остров Принца Эдуарда | C |
| Нью-Брансуик | E |
| Квебек | G, H и J |
| Онтарио | K, L, M, N и P |
| Манитоба | R |
| Саскачеван | S |
| Альберта | T |
| Британская Колумбия | V |
| Нунавут | X |
| Северо-Западные территории | X |
| Юкон | Y |

## *Упражнение 141. Английская пропись*

(*65 строк*)

Несмотря на то что популярность оплаты по чекам за последние годы серьезно снизилась, некоторые компании до сих пор используют этот способ для ведения взаиморасчетов с сотрудниками и поставщиками. Сумма на чеках обычно указывается дважды: один раз цифрами, второй – прописью на английском языке. Повторение суммы двумя разными формами записи призвано не позволить недобросовестным сотрудникам или поставщикам изменять сумму на чеках перед их обналичиванием.

В данном упражнении вам необходимо написать функцию, принимающую в качестве входного параметра число от 0 до 999 и возвращающую строку прописью. Например, если значение параметра будет равно 142, функция должна вернуть следующую строку: «one hundred forty two». Используйте один или несколько словарей вместо условных конструкций `if/elif/else` для выработки решения этой задачи. Напишите основную программу, в которой пользователь будет вводить числовое значение, а на экран будет выводиться соответствующая сумма прописью.

## *Упражнение 142. Уникальные символы*

(*Решено. 16 строк*)

Напишите программу, определяющую и выводящую на экран количество уникальных символов во введенной пользователем строке. Например,

в строке `Hello, World!` содержится десять уникальных символов, а в строке `zzz` – один. Используйте словарь или набор для решения этой задачи.

## *Упражнение 143. Анаграммы*

(*Решено. 39 строк*)

Анаграммами называются слова, образованные путем взаимной перестановки букв. В английском языке, например, анаграммами являются слова «live» и «evil», а в русском – «выбор» и «обрыв». Напишите программу, которая будет запрашивать у пользователя два слова, определять, являются ли они анаграммами, и выводить на экран ответ.

## *Упражнение 144. Снова анаграммы*

(*48 строк*)

Понятие анаграмм не ограничивается словами, а может быть расширено до целых предложений. Например, строки «William Shakespeare» и «I am a weakish speller» являются полными анаграммами, если игнорировать пробелы и заглавные буквы.

Расширьте свою программу из упражнения 143, добавив возможность проверки на анаграммы целых фраз. При анализе не обращайте внимания на знаки препинания, заглавные буквы и пробелы.

## *Упражнение 145. Эрудит*

(*Решено. 18 строк*)

В известной игре Эрудит (Scrabble™) каждой букве соответствует определенное количество очков. Общая сумма очков, которую получает игрок, составивший это слово, складывается из очков за каждую букву, входящую в его состав. Чем более употребимой является буква в языке, тем меньше очков начисляется за ее использование. В табл. 6.5 приведены все соответствия букв и очков из английской версии игры.

***Таблица 6.5. Стоимость букв в английской версии игры Эрудит***

| Очки | Буквы |
|---|---|
| 1 | A, E, I, L, N, O, R, S, T и U |
| 2 | D и G |
| 3 | B, C, M и P |
| 4 | F, H, V, W и Y |
| 5 | K |
| 8 | J и X |
| 10 | Q и Z |

Напишите программу, рассчитывающую и отображающую количество очков за собранное слово. Создайте словарь для хранения соответствий между буквами и очками и используйте его в своем решении.

**Примечание.** На игровом поле Эрудита присутствуют специальные клетки, удваивающие и утраивающие стоимость буквы или всего слова. В данном упражнении мы для простоты реализации проигнорируем этот факт.

## *Упражнение 146. Карточка лото*

(*Решено. 58 строк*)

Карточка для игры в лото состоит из пяти колонок, в каждой из которых – пять номеров. Колонки помечены буквами `B`, `I`, `N`, `G` и `O`. Под каждой буквой могут быть номера в своем диапазоне из 15 чисел. А именно под буквой `B` могут присутствовать числа от 1 до 15, под `I` – от 16 до 30, под `N` – от 31 до 45 и т. д.

Напишите функцию, которая будет создавать случайную карточку лото и сохранять ее в словаре. Ключами словаря будут буквы `B`, `I`, `N`, `G` и `O`, а значениями – списки из пяти чисел, располагающихся в колонке под каждой буквой. Создайте еще одну функцию для отображения созданной карточки лото на экране со столбцами с заголовками. В основной программе создайте карту лото случайным образом и выведите ее на экран. Ваша программа должна запускаться только в том случае, если она не импортирована в виде модуля в другой файл.

**Примечание.** Как вы знаете, в настоящих карточках для игры в лото присутствуют пустые клетки в столбцах. Мы не будем реализовывать эту особенность в нашей программе.

## *Упражнение 147. Проверка карточки*

(*102 строки*)

Карточка для игры в лото считается выигравшей, если в ней на одной линии расположились пять выпавших номеров. Обычно игроки зачеркивают номера на своих карточках. В данном упражнении мы будем обнулять в словаре выпавшие номера.

Напишите функцию, принимающую на вход карточку в качестве параметра. Если карточка содержит последовательность из пяти нулей (по вертикали, горизонтали или диагонали), функция должна возвращать `True`, в противном случае – `False`.

В основной программе вы должны продемонстрировать на примере работу функции, создав и отобразив несколько карточек с указанием того, какие из них выиграли. В вашем примере должно быть как минимум по одной карточке с выигрышем по вертикали, горизонтали и диагонали, а также карточки, на которые выигрыш не выпал. При решении этой задачи воспользуйтесь функциями из упражнения 146.

**Подсказка.** Поскольку отрицательных чисел на карточке лото быть не может, нахождение линии со всеми нулями сводится к поиску последовательности чисел, сумма которых равна нулю. Так вам будет легче решить это упражнение.

## *Упражнение 148. Играем в лото*

(*88 строк*)

В данном упражнении мы напишем программу, выполняющую симуляцию игры в лото с одной картой. Начните с генерирования списка из всех возможных номеров для выпадения (от B1 до O75). После этого перемешайте номера в хаотичном порядке, воспользовавшись функцией `shuffle` из модуля `random`. Вытаскивайте по одному номеру из списка и зачеркивайте номера, пока карточка не окажется выигравшей. Проведите 1000 симуляций и выведите на экран минимальное, максимальное и среднее количество извлечений номеров, требующееся для выигрыша. При решении этой задачи вы можете воспользоваться функциями из упражнений 146 и 147.

# Глава 7

# Файлы и исключения

Все программы, которые вы писали до сих пор, считывали данные с клавиатуры. В связи с этим пользователю необходимо было повторять ввод каждый раз, когда программа запускалась. Но это бывает не слишком удобно, особенно если речь идет о большом массиве входных данных. Также все ваши предыдущие программы выводили результаты исключительно на экран. Это нормально, если дело касается нескольких строчек, предназначенных для пользователя программы. Но если выходных данных будет много или они могут потребоваться в дальнейшем для анализа в другой программе, вариант с выводом на экран просто не подойдет. Эффективная работа с файлами решит обе перечисленные проблемы.

Файлы можно назвать относительно постоянным местом хранения информации. Данные, которые записываются в файл в процессе выполнения программы, остаются в нем после завершения ее работы и даже после выключения компьютера. Это позволяет хранить в файлах информацию, которая может быть необходимой на протяжении определенного периода времени или служить источником данных для сторонних программ, запускаемых многократно. Вы наверняка сталкивались в работе с текстовыми файлами, электронными таблицами, изображениями и видео. Да и сами программы на языке Python хранятся в файлах.

Файлы обычно делятся на *текстовые* и *двоичные*, или бинарные. В текстовых файлах хранятся исключительно последовательности битов, представляющие собой символы в определенной кодировке, например ASCII или UTF-8. Подобные файлы можно просматривать и редактировать при помощи текстовых редакторов. Все программы, которые вы писали до сих пор, сохранялись на диске в виде текстовых файлов.

Подобно текстовым, двоичные файлы также хранят последовательность битов, представляющую определенные данные. Отличие состоит в том, что эти данные не ограничиваются одними только символами. Файлы, содержащие изображения, звук или видео, являются типичными примерами двоичных файлов. В данной книге мы ограничимся работой с текстовыми файлами, поскольку их легко создавать и просматривать в ва-

шем любимом редакторе. При этом большинство действий и принципов, применяемых к текстовым файлам, прекрасно работают и в отношении двоичных файлов.

# 7.1. Открытие файлов

Чтобы начать процесс чтения данных из файла, его предварительно следует открыть. То же самое необходимо сделать и перед записью информации в файл. Открыть файл в Python можно при помощи функции *open*.

Функция `open` принимает два аргумента. Первый из них указывает имя файла, который необходимо открыть. Второй аргумент также является текстовым и характеризует *режим доступа* (access mode) к файлу. Мы будем работать с тремя режимами доступа к файлу: на чтение (r), запись (w) и добавление (a).

В качестве выходного значения функция `open` возвращает *файловый объект* (file object). При этом функция с аргументами обычно указывается справа от знака присваивания, а переменная, характеризующая файловый объект, – слева, как показано ниже.

```
inf = open("input.txt", "r")
```

После открытия файла к соответствующему ему файловому объекту можно применять методы, позволяющие извлекать информацию. То же самое можно сказать и о записи данных в файл – методы будут другие, а принципы те же. Эти методы мы подробно рассмотрим в следующих разделах главы. После завершения чтения или записи файл должен быть закрыт, для чего применяется соответствующий метод *close*.

# 7.2. Чтение из файла

Существуют разные методы для чтения данных из открытого ранее файла. Все они могут применяться только при условии, что файл открыт в режиме чтения. Попытка прочитать данные из файла, который открыт на запись или добавление, приведет к возникновению ошибки.

Метод *readline* позволяет считать одну строку из открытого файла и вернуть ее в качестве строкового значения – подобно тому, как функция `input` считывает пользовательский ввод с клавиатуры. Каждый очередной вызов метода `readline` будет возвращать следующую строку из файла. При достижении конца файла метод вернет пустую строку.

Представьте, что у вас есть файл, в котором на каждой строке располагаются числа. В следующем фрагменте кода мы подсчитаем их сумму.

```
# Запрашиваем у пользователя имя файла и открываем его
fname = input("Введите имя файла: ")
inf = open(fname, "r")

# Инициализируем сумму
total = 0

# Подсчитываем сумму
line = inf.readline()
while line != "":
    total = total + float(line)
    line = inf.readline()

# Закрываем файл
inf.close()

# Отображаем результат
print("Сумма значений в файле", fname, "равна", total)
```

В начале программы мы запрашиваем у пользователя имя файла, который необходимо открыть. После этого указанный файл открывается в режиме чтения, а ссылка на него возвращается в переменную `inf`. Затем инициализируем общую сумму нулем и считываем первую строку из файла.

В цикле `while` указано условное выражение, пропускающее в тело цикла только в случае, если считанная строка не является пустой. В самом теле считанная из файла строка преобразуется в число с плавающей запятой и добавляется к общей сумме. После этого из файла считывается следующая строка, и управление снова передается условному выражению цикла.

По достижении конца файла метод `readline` вернет пустую строку, которая будет сохранена в переменной `line`. Это приведет к тому, что условное выражение вернет значение `False`, и цикл завершится. После закрытия файла на экран выводится общая сумма.

Иногда бывает полезно считать все данные из файла за один заход, а не построчно. Это можно сделать при помощи методов *read* и *readlines*. При этом метод `read` возвращает все содержимое файла как одну длинную строку, которую при дальнейшей обработке приходится вручную разбивать на составляющие элементы. Метод `readlines`, в свою очередь, возвращает содержимое файла в виде списка строк. После завершения чтения можно пройти по созданному списку при помощи цикла и на каждой итерации извлекать по одной строке. В следующем фрагменте кода демонстрируется метод `readlines` для подсчета суммы значений из исходного файла. Здесь происходит полное считывание содержимого файла вместо получения каждой отдельной строки в цикле.

```
# Запрашиваем у пользователя имя файла и открываем его
fname = input("Введите имя файла: ")
```

```
inf = open(fname, "r")

# Инициализируем сумму
total = 0
lines = inf.readlines()

# Подсчитываем сумму
for line in lines:
    total = total + float(line)

# Закрываем файл
inf.close()

# Отображаем результат
print("Сумма значений в файле", fname, "равна", total)
```

# 7.3. Символы конца строки

В следующем примере метод `readline` используется для последовательного вывода всех строк из файла. Каждой строке будет предшествовать ее порядковый номер и двоеточие.

```
# Запрашиваем у пользователя имя файла и открываем его
fname = input("Введите имя файла для отображения: ")
inf = open(fname, "r")

# Инициализируем порядковый номер строки
num = 1

# Отображаем строки из файла, предваряя их порядковыми номерами
line = inf.readline()
while line != "":
    print("%d: %s" % (i, line))

    # Увеличиваем номер строки и считываем следующую строку
    num = num + 1
    line = inf.readline()

# Закрываем файл
inf.close()
```

При запуске данной программы вы будете удивлены тем, что она выведет – после каждой строки будет следовать еще одна строка, пустая. Причина этого в том, что каждая строка в текстовом файле оканчивается одним или несколькими символами конца строки.

**Примечание.** Символы конца строки и их количество отличаются в разных операционных системах. К счастью, в Python существует автоматическая поддержка этих различий, и текстовый файл, созданный в любой операционной системе, может быть корректно прочитан в Python.

Символы конца строки необходимы для того, чтобы любая программа, считывающая файл, могла без труда определить, где заканчивается одна строка и начинается другая. Если бы не эти символы, при любом считывании данных все содержимое файла оказывалось бы в вашей переменной, а при открытии файла в текстовом редакторе все строки были бы неразрывно связаны.

Маркер окончания строки может быть удален после чтения из файла при помощи метода *rstrip*. Этот метод, который может быть применен к абсолютно любой строке, избавляет ее от всех примыкающих справа *пробельных символов* (whitespace character), в число которых входят пробел, символ табуляции и маркеры конца строки. Метод возвращает копию исходной строки без этих специальных символов в конце строки.

Обновленный фрагмент кода программы представлен ниже. Здесь используется метод `rstrip` для удаления символов конца строки, что позволило избавиться от пустых строк при выводе содержимого файла на экран.

```
# Запрашиваем у пользователя имя файла и открываем его
fname = input("Введите имя файла для отображения: ")
inf = open(fname, "r")

# Инициализируем порядковый номер строки
num = 1

# Отображаем строки из файла, предваряя их порядковыми номерами
line = inf.readline()
while line != "":
    # Удаляем символы конца строки и отображаем строку на экране
    line = line.rstrip()
    print("%d: %s" % (i, line))

    # Увеличиваем номер строки и считываем следующую строку
    num = num + 1
    line = inf.readline()

# Закрываем файл
inf.close()
```

# 7.4. Запись в файл

Когда файл открывается в режиме записи, происходит создание нового файла. В случае если файл с таким именем уже существует, он будет предварительно удален, а все данные, содержащиеся в нем, будут потеряны. Если открыть существующий файл в режиме добавления, любые данные, которые будут записываться в него, будут добавляться в конец файла. При отсутствии на диске файла попытка его открытия в режиме добавления приведет к созданию нового файла с таким именем.

Для пополнения информацией файла, предварительно открытого на запись или добавление, можно использовать метод *write*. Этот метод принимает в качестве входного параметра строку, которая будет записана в открытый файл. Данные других типов могут быть преобразованы в строковый тип при помощи функции *str*. Несколько значений могут быть записаны в файл путем их предварительного склеивания в одну длинную строку или посредством многократного вызова метода `write`.

В отличие от функции `print`, метод `write` автоматически не вставляет символ перевода строки при записи в файл. Таким образом, разработчику необходимо самому позаботиться о том, чтобы значения, которые должны находиться в файле на разных строках, были явно разделены символами конца строки. В языке Python используется сочетание символов `\n` для обозначения маркера конца строки. Эти символы, входящие в список *escape-последовательностей* (escape sequence), могут присутствовать в строке сами по себе или как ее составная часть.

В следующем фрагменте кода мы запишем числа от единицы до введенного пользователем значения в файл. Операция конкатенации и escape-последовательность `\n` используются так, чтобы числа располагались строго на своих строках.

```
# Запрашиваем имя файла у пользователя и открываем его на запись
fname = input("В каком файле сохранить последовательность чисел? ")
outf = open(fname, "w")

# Запрашиваем максимальное значение
limit = int(input("Введите максимальное число: "))

# Пишем числа в файл - каждое в своей строке
for num in range(1, limit + 1):
    outf.write(str(num) + "\n")

# Закрываем файл
outf.close()
```

# 7.5. Аргументы командной строки

Обычно компьютерные программы запускаются путем двойного щелчка по соответствующему ярлыку. Но программы также могут быть запущены из командной строки путем ввода соответствующей команды в окно терминала или окно управления командной строкой. Например, во многих операционных системах программу на Python, сохраненную в файле `test.py`, можно запустить, написав `test.py` или `python test.py` в соответствующем окне.

Запуск программы из командной строки открывает дополнительные возможности для передачи ей параметров. Значения, необходимые программе для выполнения своей задачи, могут быть переданы непосредственно в запускающей строке после расширения файла `.py`. Такая возможность бывает очень полезна при написании скриптов, запускающих другие программы с целью автоматизации каких-либо процессов, а также при создании программ, которые должны запускаться по определенному расписанию.

Все аргументы командной строки, передаваемые в программу, сохраняются во внутренней переменной `argv`, располагающейся в модуле `sys`. По своей сути эта переменная является списком, и каждый входной параметр размещается в нем в виде отдельного строкового элемента. При этом любой элемент из списка может быть преобразован в нужный тип данных при помощи стандартных функций вроде `int` и `float`. Первым элементом списка `argv` является имя файла Python, который в данный момент запущен. Следом за ним идут переданные посредством командной строки параметры.

На примере следующей программы мы продемонстрируем доступ к аргументам командной строки непосредственно из кода. Программа начинается с вывода на экран количества элементов в переменной `argv` и имени запущенного файла. После этого на экран выводятся все переданные программе параметры. Если программа была запущена без параметров, будет показано соответствующее сообщение.

```
# Для доступа к переменным командной строки необходимо импортировать модуль sys
import sys

# Отображаем количество аргументов (включая название файла .py)
print("Программа насчитывает", len(sys.argv), \
        "аргументов командной строки.")

# Выводим на экран имя файла .py
print("Имя запущенного файла .py: ", sys.argv[0])

# Определяем, есть ли другие переданные параметры
if len(sys.argv) > 1:
```

```
    # Отображаем все переданные параметры, за исключением имени файла
    print("Остальные аргументы:")
    for i in range(1, len(sys.argv)):
        print(" ", sys.argv[i])
else:
    print("Дополнительных аргументов нет.")
```

Аргументы командной строки можно использовать для передачи программе любых входных значений, которые можно физически ввести в командную строку, то есть строк, целочисленных значений и чисел с плавающей запятой. Эти значения могут быть использованы в программе, как и любые другие. Взгляните на обновленный пример программы, суммирующей перечисленные в строках числа. Здесь имя файла мы не будем запрашивать у пользователя, а передадим посредством командной строки.

```
# Импортируем системный модуль
import sys

# Отслеживаем, чтобы программа обязательно запускалась с одним переданным параметром
if len(sys.argv) != 2:
    print("Имя файла необходимо передать в качестве", \
            "аргумента.")
    quit()

# Открываем на чтение файл, имя которого было передано в командной строке
inf = open(sys.argv[1], "r")

# Инициализируем сумму нулем
total = 0

# Суммируем значения в файле
line = inf.readline()
while line != "":
    total = total + float(line)
    line = inf.readline()

# Закрываем файл
inf.close()

# Выводим результат
print("Сумма значений в файле", sys.argv[1], "составляет", total)
```

# 7.6. Исключения

Во время выполнения программы много что может пойти не так: пользователь может ввести строковое значение вместо числового, может возникнуть ошибка деления на ноль, пользователь может запросить на открытие

файл, которого не существует, да мало ли что. Все подобные ошибки называются *исключениями* (exception). По умолчанию программы, написанные на языке Python, прекращают свое выполнение при возникновении исключений. Но мы всегда можем предотвратить такое поведение, если заранее предпримем ряд мер.

Разработчик имеет возможность указать программе, в каком месте может возникнуть исключение, чтобы была возможность его перехватить и соответствующим образом обработать, тем самым защитив программу от аварийного завершения. Для этого в языке Python предусмотрено два ключевых слова, с которыми мы раньше не сталкивались: это *try* и *except*. Код, который может вызвать возникновение исключения, предусмотрительно помещается в блок `try`, следом за которым располагается альтернативный блок `except`. Когда в блоке try возникает исключение, выполнение программы немедленно переносится в соответствующий блок `except` без выполнения оставшихся инструкций в блоке `try`.

В каждом блоке `except` может быть явно указан тип исключения, для перехвата которого предназначен этот блок. Это можно сделать, указав необходимый тип исключения непосредственно после ключевого слова `except`. В этом случае только указанный тип исключения будет обрабатываться при помощи данного блока. Без указания конкретного типа исключения блок `except` будет перехватывать все возникающие исключения, которые ранее не были обработаны другими блоками `except`, принадлежащими тому же блоку `try`. Необходимо четко понимать, что инструкции из блока `except` будут выполнены только в случае возникновения исключения. Если же выполнение блока `try` пройдет нормально, следующие за ним блоки `except` будут просто пропущены, и выполнение программы продолжится с первой строки кода, следующей за последним блоком `except` соответствующего блока `try`.

Все программы, которые мы писали до сих пор в этой главе, аварийно завершали свою работу при вводе пользователем несуществующего имени файла. Это объясняется тем, что в данный момент возникало исключение `FileNotFoundError`, которое не было перехвачено. В следующем фрагменте кода мы исправим это упущение и обработаем данное исключение с выводом соответствующего сообщения. После указанного фрагмента можно добавить любой код для обработки информации из открытого файла.

```
# Запрашиваем имя файла у пользователя
fname = input("Введите имя файла: ")

# Попытка открыть файл
try:
    inf = open(fname, "r")
except FileNotFoundError:
    # Показываем сообщение и выходим из программы, если возникла проблема при открытии
    # файла
```

```
    print("Невозможно открыть файл '%s'. Выходим...")
    quit()
```

Данная версия программы будет аккуратно закрываться с предупреждением, если файл, к которому хочет обратиться пользователь, не существует по указанному адресу. И хотя иногда такое поведение программы будет приемлемым, чаще мы захотим дать пользователю возможность повторно ввести имя файла для открытия. При этом и со второй попытки у него может не получиться ввести правильное имя файла. Организуем цикл, успешный выход из которого возможен только после ввода корректного имени файла. Обратите внимание, что оба блока – `try` и `except` – находятся внутри цикла `while`.

```
# Запрашиваем имя файла у пользователя
fname = input("Введите имя файла: ")

file_opened = False
while file_opened == False:
    # Попытка открыть файл
    try:
        inf = open(fname, "r")
        file_opened = True
    except FileNotFoundError:
        # Показываем сообщение и запрашиваем имя файла повторно
        print("Файл '%s' не найден. Попробуйте еще.")
        fname = input("Введите имя файла: ")
```

В начале программы у пользователя запрашивается имя файла. После этого переменной `file_opened` присваивается значение `False`, и цикл запускается в первый раз. В теле цикла размещается блок `try` с двумя строками кода. В первой из них выполняется попытка открытия файла. Если файла нет, будет сгенерировано исключение типа `FileNotFoundError`, в результате чего выполнение программы продолжится в соответствующем блоке `except`, тогда как вторая строка в блоке `try` так и останется невыполненной. В блоке `except` на экран выводится сообщение об ошибке и производится повторный запрос имени файла у пользователя.

Выполнение программы возвращается к первой строке цикла, где происходит проверка условия `file_opened == False`. Это условие возвращает истину, и программа вновь входит в блок `try`, пытаясь открыть файл, имя которого пользователь ввел повторно. Если и такого файла не существует, все повторится согласно описанному выше алгоритму. Если же файл найти удалось, исключения не возникает, и выполнение программы переходит ко второй строке в блоке `try`, где переменной `file_opened` присваивается значение `True`. В результате блок `except` пропускается, цикл возвращается на начало и тут же завершается, поскольку условие цикла не выполняется.

Концепция, описанная в данной главе, может быть использована для идентификации и обработки всех возможных исключений, которые могут возникать в процессе выполнения программы. Наличие блоков `try` и `except` позволит вам уберечь свою программу от аварийного завершения и должным образом обрабатывать все возникающие ошибки.

# 7.7. Упражнения

В большинстве упражнений из данной главы вам придется работать с файлами. В каких-то из них содержание файлов будет не важно, в других вам придется заранее создать и заполнить их при помощи любого текстового редактора. Будут в этой главе и упражнения на чтение конкретных данных, таких как список слов, имен или химических элементов. Эти наборы данных можно скачать с сайта автора по следующей ссылке: http://www.cpsc.ucalgary.ca/~bdstephe/PythonWorkbook.

## *Упражнение 149. Отображаем начало файла*

(*Решено. 40 строк*)

В операционных системах на базе Unix обычно присутствует утилита с названием `head`. Она выводит первые десять строк содержимого файла, имя которого передается в качестве аргумента командной строки. Напишите программу на Python, имитирующую поведение этой утилиты. Если файла, указанного пользователем, не существует, или не задан аргумент командной строки, необходимо вывести соответствующее сообщение об ошибке.

## *Упражнение 150. Отображаем конец файла*

(*Решено. 35 строк*)

Продолжая тему предыдущего упражнения, в тех же операционных системах на базе Unix обычно есть и утилита с названием `tail`, которая отображает последние десять строк содержимого файла, имя которого передается в качестве аргумента командной строки. Реализуйте программу, которая будет делать то же самое. Так же, как и в упражнении 149, в случае отсутствия файла, указанного пользователем, или аргумента командной строки вам нужно вывести соответствующее сообщение.

Данную задачу можно решить сразу несколькими способами. Например, можно все содержимое файла целиком загрузить в список и затем выбрать из него последние десять элементов. А можно дважды прочитать содержимое файла: первый раз, чтобы посчитать количество строк, а второй – чтобы отобразить последние десять из них. При этом оба пере-

численных подхода нежелательны, если речь идет о файлах достаточного объема. Существует решение, требующее единственного чтения файла и сохранения всех десяти строк за раз. В качестве дополнительного задания разработайте такой алгоритм.

## *Упражнение 151. Сцепляем файлы*

(*Решено. 28 строк*)

Продолжаем тему операционных систем на базе Unix, в которых обычно также есть утилита с названием `cat`, что является сокращением от concatenate (сцепить). Эта утилита выводит на экран объединенное содержимое нескольких файлов, имена которых передаются ей в качестве аргументов командной строки. При этом файлы сцепляются в том порядке, в котором указаны в аргументах.

Напишите программу на Python, имитирующую работу этой утилиты. В процессе работы программа должна выдавать сообщения о том, какие файлы открыть не удается, и переходить к следующим файлам. Если программа была запущена без аргументов командной строки, на экран должно быть выведено соответствующее сообщение об ошибке.

## *Упражнение 152. Нумеруем строки в файле*

(*23 строки*)

Напишите программу, которая будет считывать содержимое файла, добавлять к считанным строкам порядковый номер и сохранять их в таком виде в новом файле. Имя исходного файла необходимо запросить у пользователя, так же, как и имя целевого файла. Каждая строка в созданном файле должна начинаться с ее номера, двоеточия и пробела, после чего должен идти текст строки из исходного файла.

## *Упражнение 153. Самое длинное слово в файле*

(*39 строк*)

В данном упражнении вы должны написать программу, которая будет находить самое длинное слово в файле. В качестве результата программа должна выводить на экран длину самого длинного слова и все слова такой длины. Для простоты принимайте за значимые буквы любые непробельные символы, включая цифры и знаки препинания.

## *Упражнение 154. Частота букв в файле*

(*43 строки*)

Одна из техник декодирования простейших алгоритмов шифрования заключается в применении частотного анализа. Иными словами, вы просто

анализируете зашифрованный текст, подсчитывая частоту употребления всех букв. Затем можно использовать операции подстановки для замены наиболее популярных символов на часто используемые в языке буквы (в английском это, например, буквы E и T).

Напишите программу, которая будет способствовать дешифрации текста путем вывода на экран частоты появления разных букв. При этом пробелы, знаки препинания и цифры должны быть проигнорированы. Также не должен учитываться регистр, то есть символы `a` и `A` должны восприниматься как одна буква. Имя файла для анализа пользователь должен передавать программе посредством аргумента командной строки. Если программе не удастся открыть файл для анализа или аргументов командной строки будет слишком много, на экране должно быть отображено соответствующее сообщение об ошибке.

## *Упражнение 155. Частота слов в файле*

(*37 строк*)

Разработайте программу, которая будет показывать слово (или слова), чаще остальных встречающееся в текстовом файле. Сначала пользователь должен ввести имя файла для обработки. После этого вы должны открыть файл и проанализировать его построчно, разделив при этом строки по словам и исключив из них пробелы и знаки препинания. Также при подсчете частоты появления слов в файле вам стоит игнорировать регистры.

**Подсказка.** Возможно, при решении этого задания вам поможет код, реализованный вами в упражнении 117.

## *Упражнение 156. Сумма чисел*

(*Решено. 26 строк*)

Напишите программу, которая будет суммировать все числа, введенные пользователем, игнорируя при этом нечисловой ввод. Выводите на экран текущую сумму чисел после каждого очередного ввода. Ввод пользователем значения, не являющегося числовым, должен приводить к появлению соответствующего предупреждения, после чего пользователю должно быть предложено ввести следующее число. Выход из программы будет осуществляться путем пропуска ввода. Удостоверьтесь, что ваша программа корректно обрабатывает целочисленные значения и числа с плавающей запятой.

**Подсказка.** В данном упражнении вам придется поработать не с файлами, а с исключениями.

## *Упражнение 157. Буквенные и числовые оценки*

(*106 строк*)

Напишите программу, выполняющую перевод из буквенных оценок в числовые и обратно. Программа должна позволять пользователю вводить несколько значений для перевода – по одному в каждой строке. Для начала предпримите попытку сконвертировать введенное пользователем значение из числового в буквенное. Если возникнет исключение, попробуйте выполнить обратное преобразование – из буквенного в числовое. Если и эта попытка окончится неудачей, выведите предупреждение о том, что введенное значение не является допустимым. Пусть ваша программа конвертирует оценки до тех пор, пока пользователь не оставит ввод пустым. При решении данного задания вам могут помочь наработки из упражнений 52 и 53.

## *Упражнение 158. Удаляем комментарии*

(*Решено. 53 строки*)

Как вы знаете, в языке Python для создания комментариев в коде используется символ #. Комментарий начинается с этого символа и продолжается до конца строки – без возможности остановить его раньше.

В данном упражнении вам предстоит написать программу, которая будет удалять все комментарии из исходного файла с кодом на языке Python. Пройдите по всем строкам в файле на предмет поиска символа #. Обнаружив его, программа должна удалить все содержимое, начиная с этого символа и до конца строки. Для простоты не будем рассматривать ситуации, когда знак решетки встречается в середине строки. Сохраните новое содержимое в созданном файле. Имена файла источника и файла назначения должны быть запрошены у пользователя. Удостоверьтесь в том, что программа корректно обрабатывает возможные ошибки при работе с обоими файлами.

## *Упражнение 159. Случайный пароль из двух слов*

(*Решено. 39 строк*)

Создание пароля посредством генерирования случайных символов может обернуться сложностью в запоминании полученной относительно надежной последовательности. Некоторые системы создания паролей рекомендуют сцеплять вместе два слова на английском языке, тем самым упрощая запоминание заветного ряда символов – правда, в ущерб его надежности.

Напишите программу, которая будет открывать файл со списком слов, случайным образом выбирать два из них и сцеплять вместе для получения итогового пароля. При создании пароля исходите из следующего требования: он должен состоять минимум из восьми символов и максимум из

десяти, а каждое из используемых слов должно быть длиной хотя бы в три буквы. Кроме того, сделайте заглавными первые буквы обоих слов, чтобы легко можно было понять, где заканчивается одно и начинается другое. По завершении процесса полученный пароль должен быть отображен на экране.

## *Упражнение 160. Странные слова*

(*67 строк*)

Ученикам, желающим запомнить правила написания слов в английском языке, часто напоминают следующее рифмованное одностишие: «I before E except after C» (I перед E, если не после C). Это правило позволяет запомнить, в какой последовательности писать буквы I и E, идущие в слове одна за другой, а именно: буква I должна предшествовать букве E, если непосредственно перед ними не стоит буква C. Если стоит – порядок гласных будет обратным. Примеры слов, на которые действует это правило: believe, chief, fierce, friend, ceiling и receipt. Но есть и исключения из этого правила, и одним из них является слово weird (странный).

Напишите программу, которая будет построчно обрабатывать текстовый файл. В каждой строке может присутствовать много слов, а может и не быть ни одного. Слова, в которых буквы E и I не соседствуют друг с другом, обработке подвергать не следует. Если же такое соседство присутствует, необходимо проверить, соответствует ли написание анализируемого слова указанному выше правилу. Создайте и выведите на экран два списка. В первом должны располагаться слова, следующие правилу, а во втором – нарушающие его. При этом списки не должны содержать повторяющиеся слова. Также отобразите на экране длину каждого списка, чтобы пользователю было понятно, сколько слов в файле не отвечает правилу.

## *Упражнение 161. Что за химический элемент?*

(*59 строк*)

Напишите программу, которая будет считывать файл, содержащий информацию о химических элементах, и сохранять ее в более подходящей для этого структуре данных. После этого пользователь должен ввести значение. Если введенное значение окажется целочисленным, программа должна вывести на экран обозначение и название химического элемента с введенным количеством протонов. При вводе пользователем строки необходимо отобразить количество протонов элемента с введенным пользователем обозначением или названием. Если введенное пользователем значение не соответствует ни одному из элементов в файле, необходимо вывести соответствующее сообщение об ошибке. Позвольте пользователю вводить значения до тех пор, пока он не оставит ввод пустым.

## *Упражнение 162. Книги без буквы E*

(*Решено. 50 строк*)

Истории литературы известен случай написания романа объемом около 50 тыс. слов, в котором ни разу не была употреблена самая популярная в английском алфавите буква E. Название его – «Gadsby».

Напишите программу, которая будет считывать список слов из файла и собирать статистику о том, в каком проценте слов используется каждая буква алфавита. Выведите результат для всех 26 букв английского алфавита и отдельно отметьте букву, которая встречалась в словах наиболее редко. В вашей программе должны игнорироваться знаки препинания и регистр символов.

**Примечание.** Липограмма представляет собой литературный прием, состоящий в написании текста без использования одной из букв (или группы букв). Часто отвергнутой буквой является одна из распространенных гласных, хотя это условие и не обязательно. Например, в стихотворении Эдгара Аллана По «Ворон» («The Raven»), состоящем из более тысячи слов, ни разу не встречается буква Z, что делает его полноценной липограммой. Еще одним примером липограммы является роман «Исчезание» («La Disparition»). Во французской и английской версиях этого произведения общим объемом примерно в 300 страниц не употребляется буква E, за исключением фамилии автора.

## *Упражнение 163. Популярные детские имена*

(*Решено. 54 строки*)

Набор данных, содержащий детские имена, состоит более чем из 200 файлов, в каждом из которых, помимо сотни имен, указано количество названных тем или иным именем детей. При этом файлы отсортированы по убыванию популярности имен. Для каждого года присутствует по два файла: в одном перечислены мужские имена, в другом – женские. Совокупный набор данных содержит информацию для всех лет, начиная с 1900-го и заканчивая 2012-м.

Напишите программу, которая будет считывать по одному все файлы из набора данных и выделять имена, которые были лидерами по частоте использования как минимум в одном году. На выходе должно получиться два списка: в одном из них будут присутствовать наиболее популярные имена для мальчиков, во втором – для девочек. При этом списки не должны содержать повторяющиеся имена.

## *Упражнение 164. Универсальные имена*

(*56 строк*)

Некоторые имена, такие как Бен, Джонатан и Эндрю, подходят только для мальчиков, другие – Ребекка или Флора – только для девочек. Но есть

и универсальные имена наподобие Крис или Алекс, которые могут носить и мальчики, и девочки.

Напишите программу, которая будет выводить на экран имена, использованные для мальчиков и девочек в указанном пользователем году. Если в этом году универсальных имен не было, нужно известить об этом пользователя. Кроме того, если за указанный пользователем год не было данных по именам, выведите соответствующее сообщение об ошибке. Дополнительные детали по хранению имен в файлах – в упражнении 163.

## *Упражнение 165. Самые популярные имена за период*

(*76 строк*)

Напишите программу, которая будет определять самые популярные имена для детей в выбранном пользователем периоде. Используйте базу данных из упражнения 163. Позвольте пользователю выбрать первый и последний год анализируемого диапазона. В результате программа должна вывести на экран мужское и женское имена, которые были чаще остальных даны детям в заданный период времени.

## *Упражнение 166. Имена без повторов*

(*41 строка*)

Продолжаем использовать базу имен из упражнения 163. Проходя по файлам, выберите имена без дублирования отдельно для мальчиков и для девочек и выведите их на экран. Разумеется, повторяющихся имен в этих списках быть не должно.

## *Упражнение 167. Проверяем правильность написания*

(*Решено. 58 строк*)

Автоматическая проверка орфографии не помешала бы многим из нас. В данном упражнении мы напишем простую программу, сверяющую слова из текстового файла со словарем. Неправильно написанными будем считать все слова, которых не нашлось в словаре.

Имя файла, в котором требуется выполнить орфографическую проверку, пользователь должен передать при помощи аргумента командной строки. В случае отсутствия аргумента должна выдаваться соответствующая ошибка. Сообщение об ошибке также должно появляться, если не удается открыть указанный пользователем файл. Используйте свои наработки из упражнения 117 при решении данной задачи, чтобы знаки препинания не мешали вам проводить орфографический анализ текста. Также вам следует игнорировать регистр символов при выполнении проверки.

**Подсказка.** Конечно, вы могли бы загрузить все слова из текста в список и анализировать их при помощи функции оператора `in`. Но эта операция будет выполняться довольно долго. Гораздо быстрее в Python выполняется проверка на наличие определенного ключа в словаре или значения в наборе. Если вы остановитесь на варианте со словарем, ключами должны стать слова, а значениями – ноль или любое другое число, поскольку их мы в данном случае использовать не будем.

## *Упражнение 168. Повторяющиеся слова*

(*61 строка*)

Проверка орфографии – лишь составная часть расширенного текстового анализа на предмет наличия ошибок. Одной из самых распространенных ошибок в текстах является повторение слов. Например, автор может по ошибке дважды подряд написать одно слово, как в следующем примере:

```
At least one value must be entered
entered in order to compute the average.
```

Некоторые текстовые процессоры умеют распознавать такой вид ошибок при выполнении текстового анализа.

В данном упражнении вам предстоит написать программу для определения наличия дублей слов в тексте. При нахождении повтора на экран должен выводиться номер строки и дублирующееся слово. Удостоверьтесь, что программа корректно обрабатывает случаи, когда повторяющиеся слова находятся на разных строках, как в предыдущем примере. Имя файла для анализа должно быть передано программе в качестве единственного аргумента командной строки. При отсутствии аргумента или невозможности открыть указанный файл на экране должно появляться соответствующее сообщение об ошибке.

## *Упражнение 169. Редактирование текста в файле*

(*Решено. 52 строки*)

Перед публикацией текста или документа обычно принято удалять или изменять в них служебную информацию.

В данном упражнении вам необходимо написать программу, которая будет заменять все служебные слова в тексте на символы звездочек (по количеству символов в словах). Вы должны осуществлять регистрозависимый поиск служебных слов в тексте, даже если эти слова входят в состав других слов. Список служебных слов должен храниться в отдельном файле. Сохраните отредактированную версию исходного файла в новом файле. Имена исходного файла, файла со служебными словами и нового файла должны быть введены пользователем.

**Примечание.** Вам может пригодиться метод `replace` для работы со строками при решении этого задания. Информацию о работе данного метода можно найти в интернете.

В качестве дополнительного задания расширьте свою программу таким образом, чтобы она выполняла замену служебных слов вне зависимости от того, какой регистр символов используется в тексте. Например, если в списке служебных слов будет присутствовать слово `exam`, то все следующие варианты слов должны быть заменены звездочками: exam, Exam, ExaM и EXAM.

## *Упражнение 170. Пропущенные комментарии*

(*Решено. 48 строк*)

При написании функций хорошей практикой считается предварение ее блоком комментариев с описанием назначения функции, ее входных параметров и возвращаемого значения. Но некоторые разработчики просто не пишут комментарии к своим функциям. Другие честно собираются написать их когда-нибудь в будущем, но руки так и не доходят.

Напишите программу, которая будет проходить по файлу с исходным кодом на Python и искать функции, не снабженные блоком комментариев. Можно принять за аксиому, что строка, начинающаяся со слова `def`, следом за которым идет пробел, будет считаться началом функции. И если функция документирована, предшествующая строчка должна начинаться со знака #. Перечислите названия всех функций, не снабженных комментариями, вместе с именем файла и номером строки, с которой начинается объявление функции.

Одно или несколько имен файлов с кодом на языке Python пользователь должен передать в функцию в качестве аргументов командной строки. Для файлов, которые не существуют или не могут быть открыты, должны выдаваться соответствующие предупреждения, после чего должна быть продолжена обработка остальных файлов.

## *Упражнение 171. Строки фиксированной длины*

(*45 строк*)

Ширина окна терминала обычно составляет 80 символов, хотя есть более широкие и узкие терминалы. Это затрудняет отображение текстов, разбитых на абзацы. Бывает, что строки в исходном файле оказываются слишком длинными и автоматически переносятся, тем самым затрудняя чтение, или слишком короткими, что приводит к недостаточному заполнению свободного места на экране.

Напишите программу, которая будет открывать файл и выводить его на экран с постоянной длиной строк. Если в исходном файле строка ока-

зывается слишком длинной, «лишние» слова должны быть перенесены на следующую строку, а если слишком короткой, слова со следующей строки должны перекочевать в конец текущей до полного ее заполнения. Например, следующий отрывок из «Алисы в Стране чудес»:

```
Alice was
beginning to get very tired of sitting by her
sister
on the bank, and of having nothing to do: once
or twice she had peeped into the book her sister
was reading, but it had
no
pictures or conversations in it, "and what is
the use of a book," thought Alice, "without
pictures or conversations?"
```

в отформатированном виде с длиной строки в 50 символов будет выглядеть так:

```
Alice was beginning to get very tired of sitting
by her sister on the bank, and of having nothing
to do: once or twice she had peeped into the book
her sister was reading, but it had no pictures or
conversations in it, "and what is the use of a
book," thought Alice, "without pictures or
conversations?"
```

Убедитесь, что ваша программа корректно обрабатывает тексты, в которых присутствуют абзацы. Идентифицировать конец одного абзаца и начало другого можно по наличию пустой строки в тексте после удаления символа конца строки.

**Подсказка.** Используйте константу для задания максимальной ширины строк. Это позволит вам легко адаптировать программу под любые окна.

## *Упражнение 172. Слова с шестью гласными в ряд*

(*56 строк*)

Существует как минимум одно слово в английском языке, содержащее все гласные буквы в том порядке, в котором они расположены в алфавите, а именно A, E, I, O, U и Y. Напишите программу, которая будет просматривать текстовый файл на предмет поиска и отображения таких слов. Имя исходного файла должно быть запрошено у пользователя. Если имя файла окажется неверным или возникнут иные ошибки при чтении файла, выведите соответствующее сообщение об ошибке.

# Глава 8

# Рекурсия

В главе 4 мы познакомились с функциями и сказали, что они, в свою очередь, также могут вызывать другие функции. Но мы ни разу не обмолвились о том, может ли функция вызывать саму себя. Сейчас пришло время ответить на этот вопрос, и ответить утвердительно, ведь эта специфическая техника помогает решать целый ряд серьезных задач.

Определение, включающее описание чего-то в терминах самого себя, называется *рекурсивным* (recursive). А чтобы быть полезным, рекурсивное определение должно описывать это что-то в контексте другой, обычно уменьшенной или упрощенной версии себя самого. При использовании той же версии рекурсивное определение не принесло бы пользу, поскольку оказалось бы циклически замкнутым. Для нормального функционирования рекурсия должна постепенно продвигаться к версии задачи с известным решением.

Любая функция, вызывающая саму себя, называется рекурсивной по определению, поскольку в ее описании присутствует указание на саму себя. Чтобы прийти к итоговому решению, рекурсивная функция должна реализовывать как минимум один способ достижения результата без вызова самой себя. Такой способ именуется *базовым случаем* (base case). Варианты, при которых функция обращается к себе самой, называются *рекурсивными случаями* (recursive case). В данной главе мы рассмотрим тему рекурсии на трех примерах.

## 8.1. Суммирование целых чисел

Предположим, нам необходимо получить сумму всех целых чисел от нуля до заданного `n`. Это можно сделать при помощи цикла или формулы. Но данную задачу можно решить и при помощи рекурсии. Простейший случай задачи – при `n`, равном нулю. Ответом, разумеется, будет ноль, и он может быть получен без использования другой версии определения. Так что здесь мы имеем дело с базовым случаем рекурсии.

В целом же задача поиска суммы из n положительных чисел сводится к добавлению числа n к сумме значений от нуля до n - 1. Это определение является рекурсивным, поскольку сумма n целых чисел выражена как уменьшенная версия той же самой задачи (вычисления суммы значений от нуля до n - 1) плюс дополнительная работа (добавление n к этой сумме). С каждым очередным применением рекурсивное определение стремится к базовому случаю рекурсии, когда n равно 0. По достижении базового случая рекурсия завершается, и возвращается итоговый результат.

В следующем примере мы разберем рекурсивный алгоритм, описанный ранее. Просуммируем целые числа от нуля до значения, введенного пользователем, включительно. Условное выражение if используется для определения того, с каким случаем мы имеем дело – с базовым или рекурсивным. В первом случае сразу возвращается 0, без запуска рекурсии. Во втором функция вызывается повторно с уменьшенным аргументом (n - 1). Возвращенное из рекурсии значение прибавляется к n. В итоге мы получим сумму всех целых чисел от нуля до n, которая и будет возвращена в виде результата функции.

```
# Вычисляем сумму всех целых чисел от нуля до заданного n с помощью рекурсии
# @param n - максимальное значение для включения в сумму
# @return сумма всех целых чисел от нуля до n включительно
def sum_to(n):
    if n <= 0 :
        return 0 # Базовый случай
    else:
        return n + sum_to(n - 1) # Рекурсивный случай

    # Вычисляем сумму всех целых чисел от нуля до заданного n
num = int(input("Введите положительное целое число: "))
total = sum_to(num)
print("Сумма целых чисел от нуля до", \
        num, "равна", total)
```

Представим, что произойдет, если пользователь введет 2. Для начала введенное значение будет преобразовано в число. Затем вызывается функция sum_to с аргументом 2. Условное выражение if возвращает True, так что происходит повторный рекурсивный вызов функции sum_to – на этот раз с аргументом 1. При этом рекурсивный вызов должен завершиться раньше, чем вызывающая функция сможет высчитать и вернуть результат.

Вызов функции sum_to с аргументом 1, в свою очередь, приведет к запуску еще одной ветки рекурсии с вызовом этой же функции с аргументом 0. На этом этапе у нас есть три одновременно запущенные функции sum_to с разными аргументами: 2, 1 и 0. Первая из них ожидает возвращения результата запуска второй, а вторая – третьей. Несмотря на то что эти три функции называются одинаково, каждая отдельная копия никак не связана с остальными.

При вызове функции `sum_to` с аргументом 0 возникает базовый случай, и обратно возвращается 0. Это позволит копии функции с аргументом 1 продвинуться вперед, сложив единицу с нулем, возвращенным рекурсивным вызовом. В результате функция возвращает единицу, и управление передается в первую копию функции с аргументом 2. В ней `n`, равное двум, прибавляется к единице, полученной из рекурсивного вызова, и итоговый ответ 3 сохраняется в переменной `total`. После этого значение переменной выводится на экран, и программа завершает свою работу.

## 8.2. Числа Фибоначчи

Числами Фибоначчи называется последовательность целых чисел, начинающаяся с нуля и единицы, в которой каждое последующее ее значение равно сумме двух предыдущих. Таким образом, первые десять чисел Фибоначчи будут следующие: 0, 1, 1, 2, 3, 5, 8, 13, 21 и 34. Последовательность чисел Фибоначчи обычно обозначается как $F_n$, где $n$ – неотрицательное целое число, определяющее индекс элемента в последовательности (начиная с 0).

Числа Фибоначчи, не считая первых двух, можно легко вычислить по формуле $F_n = F_{n-1} + F_{n-2}$. Данное определение рекурсивно по своей природе, поскольку каждое следующее число ряда Фибоначчи вычисляется на основе предыдущих чисел из этой последовательности. Первые два числа ряда $F_0$ и $F_1$ представляют собой базовые случаи рекурсии, так как имеют постоянные значения, не рассчитываемые на основании предыдущих элементов. Универсальная программа для рекурсивного расчета чисел Фибоначчи представлена ниже. В ней вычисляется и отображается значение $F_n$ для `n`, введенного пользователем.

```
# Рассчитываем n-е число ряда Фибоначчи
# @ragam n - индекс искомого числа Фибоначчи
# @return значение n-го числа Фибоначчи
def fib(n):
    # Базовые случаи
    if n == 0:
        return 0
    if n == 1:
        return 1

    # Рекурсивный случай
    return fib(n-1) + fib(n-2)

# Рассчитываем число Фибоначчи, запрошенное пользователем
n = int(input("Введите неотрицательное число: "))
print("fib(%d) равно %d." % (n, fib(n)))
```

Это очень компактный рекурсивный алгоритм поиска чисел Фибоначчи, но у него есть минус – он не так быстр, особенно когда речь идет о больших значениях. И если `fib(30)` на средненьком по мощности компьютере выполнится довольно быстро, то `fib(70)` потребует в разы больше времени. Так что большие числа Фибоначчи обычно вычисляются при помощи цикла или формулы.

Из этого вы могли бы сделать вывод, что алгоритмы, включающие в себя рекурсию, обязательно будут медленными. И хотя в данном конкретном случае это так, в целом подобное утверждение далеко от истины. Наша предыдущая программа, суммирующая целые числа, работала быстро даже для больших значений, а есть и другие задачи, которые быстро и элегантно решаются при помощи рекурсивных алгоритмов. Среди таких задач – алгоритм Евклида, представляющий собой эффективный способ нахождения наибольшего общего делителя двух целых чисел. Подробнее о нем мы поговорим в упражнении 174.

На рис. 8.1 схематически показано количество вызовов функций при вычислении числа Фибоначчи для `n`, равного 4 и 5, и аналогичное количество при расчете суммы целых чисел. Сравнение алгоритмов при разных `n` наглядно демонстрирует принципиальную разницу между ними.

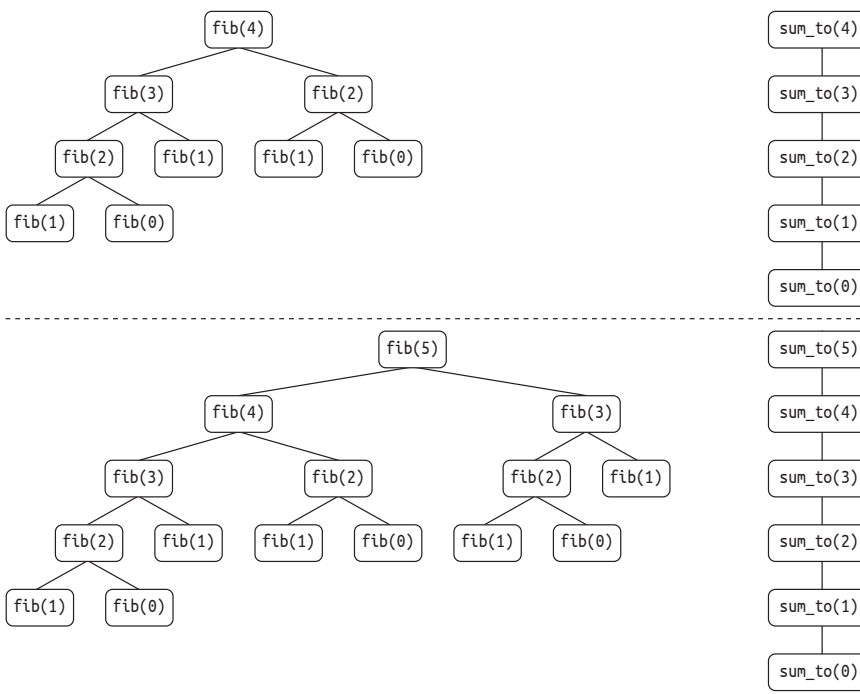



**Рис. 8.1** ❖ Сравнение алгоритмов функций `fib` и `sum_to`)

При увеличении аргумента, передаваемого функции `sum_to`, с 4 до 5 количество вызываемых функций для получения результата также возрастает с 4 до 5. Таким образом, в общем случае увеличение аргумента на единицу для этой функции ведет к вызову одной дополнительной функции при вычислении результата. Здесь наблюдается линейный рост сложности алгоритма в зависимости от величины аргумента, а количество рекурсивных вызовов функции прямо пропорционально величине исходного аргумента.

В то же время при увеличении аргумента, передаваемого в функцию `fib`, с 4 до 5 количество вызовов функций возрастет с 9 до 15. Так что прирост числа `n` на единицу в случае с вычислением чисел Фибоначчи практически удваивает количество рекурсивных вызовов внутри функции. Такой рост сложности алгоритма называется экспоненциальным. Эффективность алгоритмов подобного типа существенно страдает при их запуске для больших значений, поскольку требует двойного прироста времени при каждом увеличении значения аргумента на единицу.

## 8.3. Подсчет символов

Рекурсивные алгоритмы применимы в большом количестве задач, определения которых могут быть выражены через самих себя. И список подобных задач не ограничивается работой с целыми числами. Представьте, что вам необходимо выяснить, сколько раз определенный символ встречается в строке. Для решения этой задачи может быть написана рекурсивная функция, принимающая на вход строку и искомый символ и возвращающая количество появлений этого символа в строке.

Базовый случай для этой функции возникает, если переданная в нее строка является пустой. Логично предположить, что число вхождений любого символа в пустую строку равно нулю, и именно такой результат возвращается из функции – без дополнительных рекурсивных вызовов.

Если в функцию передана непустая строка, количество вхождений в нее определенного символа можно определить при помощи следующего рекурсивного алгоритма. Для упрощения определения рекурсивного случая выделим часть переменной `s` с исходной строкой, но без первой буквы. Таким образом, подобной урезанной частью строки, состоящей из единственного символа, будет пустая строка.

Если первым символом в строке `s` будет искомый символ из переменной `ch`, то количество его вхождений в строку будет равно единице плюс количество вхождений этого символа в урезанную строку (без первой буквы). В противном случае число вхождений `ch` в строку `s` будет равно просто числу его вхождений в урезанную строку. Это определение будет постепенно стремиться к базовому случаю рекурсии (когда строка `s` окажется

пустой), поскольку сокращенная строка всегда короче полной. Программа, в которой реализован данный алгоритм, представлена ниже.

```
# Посчитать, сколько раз символ входит в строку
# @param s – строка, в которой мы будем искать вхождения
# @param ch – искомый символ
# @return количество вхождений символа ch в строку s
def count(s, ch):
    if s == "":
        return 0 # Базовый случай

    # Сокращаем строку, отсекая первый символ
    tail = s[1 : len(s)]

    # Рекурсивные случаи
    if ch == s[0]:
        return 1 + count(tail, ch)
    else:
        return count(tail, ch)

# Считаем количество вхождений символа в строку
s = input("Введите строку: ")
ch = input("Введите искомый символ: ")
print("'%s' появляется %d раз в строке '%s'" % (ch, count(s, ch), s))
```

# 8.4. Упражнения

Функция называется рекурсивной, если в процессе выполнения она вызывает саму себя. Подобные функции обычно включают один или более базовых случаев и один или более рекурсивных случаев. В базовых случаях функции для вычисления результата не требуется обращаться к самой себе повторно, тогда как в рекурсивных планируемый расчет возможен только с попутным обращением к копии этой же функции – зачастую с использованием уменьшенной или упрощенной версии определения. Все упражнения из данного раздела должны быть выполнены при помощи написания одной или нескольких рекурсивных функций. Каждая из этих функций должна вызывать саму себя и может использовать все техники, ранее описанные в этой книге.

## *Упражнение 173. Сумма значений*

(*Решено. 29 строк*)

Напишите программу, которая будет складывать числа, введенные пользователем. Сигналом к окончанию ввода должна служить пустая строка. Отобразите на экране сумму значений (или 0.0, если пользователь сразу

же пропустил ввод). Решите эту задачу с использованием рекурсии. В вашей программе не должны присутствовать циклы.

**Подсказка.** В теле вашей рекурсивной функции должен производиться запрос одного числа у пользователя, после чего должно быть принято решение о том, производить ли еще один рекурсивный вызов. Ваша функция не должна принимать аргументов, а возвращать будет числовое значение.

## *Упражнение 174. Наибольший общий делитель*

(*24 строки*)

Евклид был греческим математиком, жившим около 2300 лет назад. Именно ему приписывается авторство эффективного рекурсивного алгоритма нахождения наибольшего общего делителя двух положительных чисел *a* и *b*. Этот алгоритм описывается так:

```
Если b = 0, тогда
    Возвращаем a
Иначе
    c = остаток от деления a на b
    Возвращаем наибольший общий делитель чисел b и c
```

Напишите программу, реализующую алгоритм Евклида для определения наибольшего общего делителя двух положительных чисел, введенных пользователем. Проверьте программу на работоспособность с очень большими числами. Результат должен высчитываться очень быстро даже для огромных входных значений, состоящих из сотен чисел. Причина заключается в очень высокой эффективности данного алгоритма.

## *Упражнение 175. Рекурсивный перевод числа из десятичного в двоичное*

(*34 строки*)

В упражнении 82 мы уже писали программу, которая посредством цикла переводила значение из десятичной системы счисления в двоичную. Здесь вам придется реализовать этот алгоритм при помощи рекурсии.

Напишите рекурсивную функцию, переводящую неотрицательное целое число в двоичную систему. Воспринимайте 0 и 1 как базовые случаи с возвратом соответствующего строкового значения. Для остальных положительных чисел `n` вам необходимо вычислить следующую цифру при помощи оператора взятия остатка от деления и затем осуществить рекурсивный вызов с вычислением цифр для `n // 2`. Наконец, вам нужно сцепить строковый результат рекурсивного вызова со следующей циф-

рой, которую заранее надо преобразовать в строку, и вернуть полученную строку в качестве результата функции.

Напишите основную программу, которая будет использовать рекурсивную функцию для преобразования неотрицательного числа, введенного пользователем, из десятичной системы счисления в двоичную. Если будет введено отрицательное значение, программа должна вывести соответствующее сообщение об ошибке.

## *Упражнение 176. Фонетический алфавит НАТО*

(*33 строки*)

Фонетический алфавит представляет собой таблицу обозначений букв, каждой из которых соответствует то или иное слово. Широкое распространение такие алфавиты приобретают в условиях повышенной зашумленности каналов передачи информации, когда собеседник может просто не расслышать конкретную букву. В таких случаях вместо букв используются целые слова. Один из наиболее распространенных фонетических алфавитов был разработан в военном блоке НАТО. Соответствие букв и слов в нем приведено в табл. 8.1.

Напишите программу, которая будет запрашивать слово у пользователя и отображать его на экране в виде шифра из соответствующих слов, обозначающих буквы исходного текста. Например, если пользователь введет слово `Hello`, на экране должна быть отображена следующая последовательность слов: `Hotel Echo Lima Lima Oscar`. Для решения этой задачи вам предстоит использовать рекурсивную функцию, а не циклы. При этом все небуквенные символы, введенные пользователем, можно игнорировать.

***Таблица 8.1. Фонетический алфавит НАТО***

| Буква | Слово | Буква | Слово | Буква | Слово |
|---|---|---|---|---|---|
| A | Alpha | J | Juliet | S | Sierra |
| B | Bravo | K | Kilo | T | Tango |
| C | Charlie | L | Lima | U | Uniform |
| D | Delta | M | Mike | V | Victor |
| E | Echo | N | November | W | Whiskey |
| F | Foxtrot | O | Oscar | X | Xray |
| G | Golf | P | Papa | Y | Yankee |
| H | Hotel | Q | Quebec | Z | Zulu |
| I | India | R | Romeo | | |

## *Упражнение 177. Римские цифры*

(*25 строк*)

Как ясно из названия, римские цифры появились еще в Древнем Риме. Но даже после падения Римской империи они продолжали использоваться на территории Европы вплоть до позднего Средневековья, а в определенных случаях применяются и сегодня.

Римские цифры базируются на обозначениях M, D, C, L, X, V и I, соответствующих числам 1000, 500, 100, 50, 10, 5 и 1. В основном римские цифры в составляющих их числах располагаются в порядке убывания – от больших к меньшим. В этом случае итоговое число равно сумме всех составляющих его римских цифр. Если цифры меньшего номинала предшествуют цифрам большего номинала, то первые вычитаются из вторых и итог прибавляется к общей сумме. В такой манере могут использоваться римские цифры C, X и I. При этом вычитание производится только из чисел, максимум в десять раз превосходящих вычитаемое. Таким образом, цифра I может предшествовать V или X, но не может вычитаться из L, C, D или M. А значит, число 99 должно быть написано как XCIX, а не IC.

Создайте рекурсивную функцию, которая будет переводить римскую запись чисел в десятичную. Функция должна обрабатывать один или два символа в начале строки, после чего будет производиться ее рекурсивный вызов для оставшихся символов. Используйте пустую строку с возвращаемым значением 0 в качестве базового случая. Также напишите основную программу, которая будет запрашивать у пользователя число, введенное римскими цифрами, и отображать его десятичный эквивалент. При этом можно сделать допуск о том, что пользователь всегда вводит корректное число, так что обработку ошибок вам реализовывать не нужно.

## *Упражнение 178. Рекурсивные палиндромы*

(*Решено. 30 строк*)

Определение палиндрома мы дали еще в главе 75, а в данном упражнении вам предстоит написать рекурсивную функцию, определяющую, является ли переданная ей в виде аргумента строка палиндромом. Стоит отметить, что пустая строка является палиндромом по определению, как и строка, состоящая из одного символа. Более длинные строки можно считать палиндромами, если их первый и последний символы одинаковые, а подстрока, исключающая эти символы, также является палиндромом.

Напишите основную программу, которая будет запрашивать у пользователя слово и при помощи рекурсивной функции определять, является ли оно палиндромом. В результате на экране должно появиться соответствующее сообщение.

## *Упражнение 179. Рекурсивный квадратный корень*

(*20 строк*)

В упражнении 74 вы узнали, как можно при помощи итераций вычислить квадратный корень из числа. В той задаче с каждой итерацией цикла увеличивалась точность приближения расчета. Здесь мы будем применять похожую тактику приближения, но с использованием рекурсии, а не цикла.

Напишите функцию вычисления квадратного корня с двумя входными параметрами. Первый из них – *n* – будет характеризовать число, из которого вычисляется квадратный корень, а второй – *guess* – текущее приближение при его вычислении. Значение параметра *guess* по умолчанию примем за 1,0. У первого параметра значения по умолчанию быть не должно.

Функция вычисления корня должна быть рекурсивной. Базовый случай будет возникать тогда, когда $guess^2$ будет отличаться от *n* менее чем на $10^{-12}$. В этом случае функция должна возвращать *guess*, считая, что полученное число достаточно близко к квадратному корню от заданного *n*. В противном случае функция должна возвращать результат рекурсивного вызова себя самой с *n* в качестве первого параметра и $\frac{guess + \frac{n}{guess}}{2}$ в качестве второго.

Напишите основную программу, в которой будет демонстрироваться работа вашей функции на примере вычисления квадратного корня из нескольких чисел. При первом вызове функции в основной программе необходимо передать ей лишь один параметр, а значение второго – *guess* – возьмется по умолчанию.

## *Упражнение 180. Редакционное расстояние*

(*Решено. 43 строки*)

Редакционное расстояние между двумя строками представляет собой меру их схожести. Чем меньше между строками редакционное расстояние, тем более похожими они могут считаться. Это означает, что одна строка может быть преобразована в другую с минимальным количеством операций вставки, удаления и замены символов.

Рассмотрим слова `kitten` и `sitting`. Первое слово может быть трансформировано во второе путем выполнения следующих операций: заменить `k` на `s`, заменить `e` на `i` и добавить `g` в конец строки. Это минимально возможное количество операций, необходимое для преобразования слова `kitten` в `sitting`. Таким образом, редакционное расстояние между этими строками составляет 3.

Напишите рекурсивную функцию, вычисляющую редакционное расстояние между двумя строками по следующему алгоритму.

```
Имеем исходные строки s и t
Если длина строки s равна нулю, тогда
    Возвращаем длину строки t
Если длина строки t равна нулю, тогда
    Возвращаем длину строки s
Иначе
    Устанавливаем переменную cost равной 0
    Если последний символ в s не совпадает с последним символом в t, тогда
        Устанавливаем cost равной 1
    Устанавливаем d1 равной редакционному расстоянию между строкой s без последнего
символа и строкой t с прибавлением единицы
    Устанавливаем d2 равной редакционному расстоянию между строкой t без последнего
символа и строкой s с прибавлением единицы
    Устанавливаем d3 равной редакционному расстоянию между строкой s без последнего
символа и строкой t без последнего символа с прибавлением cost
    Возвращаем минимальное значение среди d1, d2 и d3
```

Используйте написанную вами рекурсивную функцию в основной программе, запрашивающей у пользователя две строки и выводящей на экран редакционное расстояние между ними.

## *Упражнение 181. Возможный размен*

(*41 строка*)

Напишите программу, которая будет определять, можно ли составить конкретную сумму из определенного количества монет. Например, можно набрать доллар из четырех монет номиналом в 25 центов. Но при помощи пяти монет доллар никак не собрать. При этом из шести монет это снова возможно, если взять три монеты по 25 центов, две – по 10 и одну номиналом в 5 центов. Также возможно собрать сумму $1,25 из пяти или восьми монет, но не удастся это сделать с четырьмя, шестью или семью монетами.

Ваша основная программа должна запрашивать у пользователя искомую сумму и количество монет. На выходе вы должны получить сообщение о том, можно или нет собрать введенную сумму при помощи заданного количества монет. Представьте, что для решения этой задачи в вашем распоряжении есть монеты номиналом 1, 5, 10 и 25 центов. Также ваша программа должна включать рекурсивный алгоритм. Циклов в ней быть не должно.

## *Упражнение 182. Слова через химические элементы*

(*67 строк*)

Каждый химический элемент из таблицы Менделеева характеризуется своим обозначением, состоящим из одного, двух или трех символов. Есть такая игра – пытаться выразить то или иное слово через химические эле-

менты. Например, слово `silicon` может быть записано при помощи следующих химических элементов: Si, Li, C, O и N. В то же время слово `hydrogen` не может быть составлено из названий элементов.

Напишите рекурсивную функцию, способную определять, можно ли выразить переданное ей слово исключительно через обозначения химических элементов. Ваша функция должна принимать два параметра: слово, которое нужно проверить, и список символов, которые можно при этом использовать. Возвращать функция должна строку, состоящую из использованных символов, если собрать искомое слово можно, и пустую строку в противном случае. При этом регистры символов учитываться не должны.

В основной программе должна быть использована ваша функция для проверки всех элементов таблицы Менделеева на возможность составить их названия из обозначений химических элементов. Отобразите на экране названия элементов вместе с обозначениями, которые были использованы для их написания. Например, одна из строчек может выглядеть так:

```
Silver может быть представлен как SiLvEr
```

Для решения задачи вы можете воспользоваться набором данных с химическими элементами, который можно скачать на сайте автора. Этот набор включает в себя названия и обозначения всех 118 элементов периодической таблицы.

## *Упражнение 183. Последовательность химических элементов*

(*Решено. 81 строка*)

Существует такая игра, в которой химические элементы выстраиваются в длинную цепочку таким образом, чтобы каждое очередное слово начиналось на букву, на которую заканчивается предыдущее. Например, если последовательность начинается с элемента *Hydrogen*, то следующий элемент должен начинаться на *N*, и это может быть *Nickel*. Следом может идти *Lithium*. При этом элементы в ряду не должны повторяться. При игре в одиночку целью является составление списка элементов максимально возможной длины. Если в игре принимают участие двое, то целью становится назвать такой химический элемент, чтобы оппонент не смог продолжить последовательность.

Напишите программу, которая будет запрашивать у пользователя химический элемент и при помощи рекурсивной функции определять максимально возможную последовательность слов. Выведите на экран полученный ряд. Удостоверьтесь, что программа выводит соответствующее сообщение об ошибке, если пользователь укажет несуществующий химический элемент.

**Подсказка.** Для некоторых элементов вашей программе может понадобиться до двух минут, чтобы найти последовательность максимально возможной длины. Так что вам лучше начать проверку своей программы с элементов Molybdenum или Magnesium, для которых длина последовательности составляет всего восемь слов. На выполнение этой задачи уйдет всего несколько секунд.

## *Упражнение 184. Выравниваем список*

(*Решено. 33 строки*)

Списки в языке Python могут содержать в себе вложенные списки. В то же время вложенные списки могут также состоять из списков, тем самым увеличивая глубину общей иерархии. В результате списки могут обретать структуру, похожую на следующую: `[1, [2, 3], [4, [5, [6, 7]]], [[[8], 9], [10]]]`.

Списки со множеством уровней вложенности могут пригодиться при описании сложных отношений между элементами, но при этом такая структура значительно усложняет выполнение операций над элементами.

*Процесс выравнивания* (flattening) списка заключается в избавлении от вложенной структуры и простом перечислении входящих в него элементов. Допустим, для приведенного выше примера выровненный список будет выглядеть так: `[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`. Для выполнения операции выравнивания списка *data* можно последовать следующему алгоритму:

```
Если список data пуст, тогда
    Возвращаем пустой список
Если первый элемент списка data также является списком, тогда
    Выравниваем первый элемент списка data и присваиваем результат переменной l1
    Выравниваем все элементы первого элемента списка data, за исключением первого,
и присваиваем результат переменной l2
    Возвращаем результат конкатенации списков l1 и l2
Если первый элемент списка data не является списком, тогда
    Присваиваем переменной l1 список, состоящий из первого элемента списка data
    Выравниваем все элементы первого элемента списка data, за исключением первого,
и присваиваем результат переменной l2
    Возвращаем результат конкатенации списков l1 и l2
```

Напишите функцию, реализующую рекурсивный алгоритм выравнивания списка, описанный выше. Функция должна принимать на вход список и возвращать выровненную версию списка. В основной программе продемонстрируйте работу функции на примере приведенного выше списка, а также нескольких других.

**Подсказка.** В Python есть функция `type`, возвращающая тип данных своего единственного аргумента. Информацию о том, как узнать, является ли передаваемый ей аргумент списком, можно найти в интернете.

## *Упражнение 185. Декодирование на основе длин серий*

(*33 строки*)

Кодирование на основе длин серий представляет собой простую технику сжатия информации, которая демонстрирует свою эффективность при наличии множества соседствующих друг с другом повторяющихся значений. Сжатие достигается за счет замены целой группы повторяющихся значений на однократное его упоминание с отдельно хранящимся счетчиком повторов. Например, список `["A", "A", "A", "A", "A", "A", "A", "A", "A", "A", "A", "A", "B", "B", "B", "B", "A", "A", "A", "A", "A", "A", "B"]` может быть закодирован в следующем виде: `["A", 12, "B", 4, "A", 6, "B", 1]`. Процесс декодирования заключается в размножении каждого элемента в соответствии со счетчиком.

Напишите рекурсивную функцию для декодирования списка, кодированного на основе длин серий. В качестве единственного аргумента функция должна принимать закодированный соответствующим образом список. На выходе должен получиться расшифрованный список элементов. В основной программе продемонстрируйте работу алгоритма декодирования на примере представленного здесь списка.

## *Упражнение 186. Кодирование на основе длин серий*

(*Решено. 38 строк*)

Напишите рекурсивную функцию, реализующую алгоритм кодирования на основе длин серий, описанный в упражнении 185. На вход функции должен поступать список или строка, а на выходе будет закодированный список. В основной программе запросите у пользователя строку, сожмите ее при помощи своей функции и отобразите на экране кодированный список.

**Подсказка.** Возможно, вам придется поместить цикл внутрь тела своей рекурсивной функции.

# Часть II

# РЕШЕНИЯ

# Глава 9

# Введение в программирование

## *Упражнение 1. Почтовый адрес*

```
##
# Отобразить почтовый адрес пользователя
#
print("Бен Стивенсон")
print("Департамент теории вычислительных систем")
print("Университет Калгари")
print("2500 Университетское шоссе NW")
print("Калгари, Альберта T2N 1N4")
print("Канада")
```

## *Упражнение 3. Площадь комнаты*

```
##
# Вычислить площадь комнаты
#

# Считываем ввод пользователя
length = float(input("Введите длину комнаты (м): "))
width = float(input("Введите ширину комнаты (м): "))

# Вычислим площадь комнаты
area = length * width

# Отобразим результат
print("Площадь комнаты равна", area, "кв.м.")
```

Функция `float` используется для преобразования пользовательского ввода в нужный тип данных.

В Python умножение выполняется при помощи оператора *.

## *Упражнение 4. Площадь садового участка*

```
##
# Вычисляем площадь садового участка в акрах
#
```

```
SQFT_PER_ACRE = 43560

# Запрашиваем информацию у пользователя
length = float(input("Введите длину участка (футы): "))
width = float(input("Введите ширину участка (футы): "))

# Вычислим площадь в акрах
acres = length * width / SQFT_PER_ACRE

# Отобразим результат
print("Площадь садового участка равна", acres, "акров")
```

## *Упражнение 5. Сдаем бутылки*

```
##
# Вычисляем доход от сданной тары
#
LESS_DEPOSIT = 0.10
MORE_DEPOSIT = 0.25

# Запрашиваем у пользователя количество бутылок каждого вида
less = int(input("Сколько у вас бутылок объемом 1 литр и меньше? "))
more = int(input("Сколько у вас бутылок объемом больше 1 литра? "))

# Вычисляем сумму
refund = less * LESS_DEPOSIT + more * MORE_DEPOSIT

# Отобразим результат
print("Ваша выручка составит $%.2f." % refund)
```

> Спецификатор формата %.2f указывает Python на то, что необходимо форматировать значение в виде числа с плавающей запятой с двумя десятичными знаками.

## *Упражнение 6. Налоги и чаевые*

```
##
# Вычисляем налог и сумму чаевых в ресторане
#
TAX_RATE = 0.05
TIP_RATE = 0.18
```

> В моем регионе налог составляет 5 %. В языке Python 5 % и 18 % представляются как 0.05 и 0.18 соответственно.

```
# Запрашиваем сумму счета у пользователя
cost = float(input("Введите сумму счета: "))
```

```
# Вычисляем сумму налога и чаевых
tax = cost * TAX_RATE
tip = cost * TIP_RATE
total = cost + tax + tip

# Отобразим результат
print("Налог составил %.2f, чаевые - %.2f, общая сумма", \
        "заказа: %.2f" % (tax, tip, total))
```

Знак обратной косой черты (\) называется символом продолжения строки. Он сообщает Python о том, что инструкция будет продолжена на следующей строке. Не вводите пробелы и иные символы, включая символ табуляции, после косой черты.

## *Упражнение 7. Сумма первых n положительных чисел*

```
##
# Рассчитываем сумму первых n положительных чисел
#
# Запрашиваем значение числа n у пользователя
n = int(input("Введите положительное число: "))

# Рассчитываем сумму
sm = n * (n + 1) / 2
```

В Python есть встроенная функция с именем `sum`, поэтому для нашей переменной мы выбрали другое имя.

```
# Отобразим результат
print("Сумма первых", n, "положительных чисел равна", sm)
```

## *Упражнение 10. Арифметика*

```
##
# Демонстрируем математические операции и использование модуля math
#
from math import log10
```

Необходимо импортировать функцию `log10` из модуля `math`, прежде чем она будет использована. Обычно все инструкции импорта располагаются в начале кода.

```
# Запрашиваем два целых числа у пользователя
a = int(input("Введите число a: "))
```

```
b = int(input("Введите число b: "))

# Рассчитываем и отображаем сумму, разницу, произведение, частное и остаток от деления
print(a, "+", b, "=", a + b)
print(a, "-", b, "=", a - b)
print(a, "*", b, "=", a * b)
print(a, "/", b, "=", a / b)
print(a, "%", b, "=", a % b)
```

Остаток от деления двух чисел вычисляется при помощи функции %, для возведения в степень есть оператор **.

```
# Рассчитываем десятичный логарифм и степень
print("Десятичный логарифм числа", a, "равен", log10(a))
print(a, "в степени", b, "равно", a ** b)
```

## *Упражнение 13. Размен*

```
##
# Рассчитываем минимальное количество монет для представления указанной суммы
#
CENTS_PER_TOONIE = 200
CENTS_PER_LOONIE = 100
CENTS_PER_QUARTER = 25
CENTS_PER_DIME =10
CENTS_PER_NICKEL =5

# Запрашиваем у пользователя сумму в центах
cents = int(input("Введите сумму в центах: "))

# Определим количество двухдолларовых монет путем деления суммы на 200. Затем вычислим
# оставшуюся сумму для размена, рассчитав остаток от деления
print(" ", cents // CENTS_PER_TOONIE, "двухдолларовых монет")
cents = cents % CENTS_PER_TOONIE
```

Деление без остатка в Python выполняется при помощи оператора //. При этом результат всегда будет округлен в нижнюю сторону, что нам и требуется.

```
# Повторяем эти действия для остальных монет
print(" ", cents // CENTS_PER_LOONIE, "однодолларовых монет")
cents = cents % CENTS_PER_LOONIE
print(" ", cents // CENTS_PER_QUARTER, "25-центовых монет")
cents = cents % CENTS_PER_QUARTER
print(" ", cents // CENTS_PER_DIME, "10-центовых монет")
cents = cents % CENTS_PER_DIME
print(" ", cents // CENTS_PER_NICKEL, "5-центовых монет")
cents = cents % CENTS_PER_NICKEL

# Отобразим остаток в центах
print(" ", cents, "центов")
```

## *Упражнение 14. Рост*

```
##
# Преобразуем рост в футах и дюймах в сантиметры
#
IN_PER_FT = 12
CM_PER_IN = 2.54

# Запрашиваем рост у пользователя
print("Введите рост:")
feet = int(input("Количество футов: "))
inches = int(input("Количество дюймов: "))

# Переводим в сантиметры
cm = (feet * IN_PER_FT + inches) * CM_PER_IN

# Отобразим результат
print("Ваш рост в сантиметрах:", cm)
```

## *Упражнение 17. Теплоемкость*

```
##
# Вычислить количество энергии, требуемое для нагрева воды, а также стоимость нагрева
#
# Определим константы для удельной теплоемкости воды и стоимости электричества
WATER_HEAT_CAPACITY = 4.186
ELECTRICITY_PRICE = 8.9
J_TO_KWH = 2.777e-7
```

> Python позволяет записывать числа в научной нотации, когда коэффициент располагается слева от буквы e, а порядок – справа. Таким образом, число $2{,}777*10^{-7}$ может быть записано как 2,777e–7.

```
# Запрашиваем у пользователя объем воды и требуемое изменение температуры
volume = float(input("Объем воды в миллилитрах: "))
d_temp = float(input("Повышение температуры (в градусах Цельсия): "))
```

> Поскольку вода обладает плотностью 1 грамм на миллилитр, в данном упражнении можно взаимозаменять граммы и миллилитры. Решение запрашивать у пользователя объем жидкости в миллилитрах было принято из соображений удобства, поскольку люди привыкли представлять себе объем кофейной кружки, а не массу воды в ней.

```
# Вычисляем количество энергии в джоулях
q = volume * d_temp * WATER_HEAT_CAPACITY
```

```
# Отображаем результат в джоулях
print("Потребуется %d Дж энергии." % q)

# Вычисляем стоимость
kwh = q * J_TO_KWH
cost = kwh * ELECTRICITY_PRICE

# Отображаем стоимость
print("Стоимость энергии: %.2f центов." % cost)
```

## *Упражнение 19. Свободное падение*

```
##
# Рассчитываем скорость объекта, отпущенного с определенной высоты,
# в момент столкновения с землей
#
from math import sqrt

# Определяем константу ускорения свободного падения
GRAVITY = 9.8

# Запрашиваем высоту, с которой объект был отпущен
d = float(input("Высота отпускания объекта (в метрах): "))

# Рассчитываем финальную скорость
vf = sqrt(2 * GRAVITY * d)
```

$v_i^2$ не была включена в формулу расчета $v_f$, поскольку $v_i = 0$.

```
# Отобразим результат
print("Объект достигнет земли на скорости %.2f м/с." % vf)
```

## *Упражнение 23. Площадь правильного многоугольника*

```
##
# Вычисляем площадь правильного многоугольника
#
from math import tan, pi

# Запрашиваем информацию у пользователя
s = float(input("Введите длину сторон: "))
n = int(input("Введите число сторон: "))
```

Сразу конвертируем *n* в целочисленное значение, а не в число с плавающей запятой, поскольку у многоугольника не может быть дробного количества сторон.

```
# Вычисляем площадь многоугольника
area = (n * s ** 2) / (4* tan(pi / n))

# Отобразим результат
print("Площадь многоугольника равна", area)
```

## Упражнение 25. Единицы времени (снова)

```
##
# Переводим секунды в дни, часы, минуты и секунды
#
SECONDS_PER_DAY = 86400
SECONDS_PER_HOUR = 3600
SECONDS_PER_MINUTE = 60

# Запрашиваем у пользователя длительность в секундах
seconds = int(input("Введите количество секунд: "))

# Переводим введенное значение в дни, часы, минуты и секунды
days = seconds / SECONDS_PER_DAY
seconds = seconds % SECONDS_PER_DAY
hours = seconds / SECONDS_PER_HOUR
seconds = seconds % SECONDS_PER_HOUR
minutes = seconds / SECONDS_PER_MINUTE
seconds = seconds % SECONDS_PER_MINUTE

# Отобразим результат в требуемом формате
print("Длительность:", \
        "%d:%02d:%02d:%02d." % (days, hours, minutes, seconds))
```

Спецификатор формата `%02d` указывает Python на то, что необходимо форматировать целочисленное значение в виде двух цифр путем добавления ведущего нуля при необходимости.

## Упражнение 29. Температура с учетом ветра

```
##
# Вычисляем коэффициент охлаждения ветром
#
WC_OFFSET = 13.12
WC_FACTOR1 = 0.6215
WC_FACTOR2 = -11.37
WC_FACTOR3 = 0.3965
WC_EXPONENT = 0.16

# Запрашиваем у пользователя температуру воздуха и скорость ветра
temp = float(input("Температура воздуха (градусы Цельсия): "))
```

Вычисление коэффициента охлаждения ветром потребовало ввода нескольких констант, которые были определены учеными и медиками.

```
speed = float(input("Скорость ветра (км/ч): "))

# Определяем коэффициент охлаждения ветром
wci = WC_OFFSET + \
WC_FACTOR1 * temp + \
WC_FACTOR2 * speed ** WC_EXPONENT + \
WC_FACTOR3 * temp * speed ** WC_EXPONENT

# Отобразим результат, округленный до ближайшего целого
print("Коэффициент охлаждения ветром равен", round(wci))
```

## *Упражнение 33. Сортировка трех чисел*

```
##
# Сортируем три числа по возрастанию
#
# Запрашиваем числа у пользователя и записываем их в переменные a, b и c
a = int(input("Введите первое число: "))
b = int(input("Введите второе число: "))
c = int(input("Введите третье число: "))
mn = min(a, b, c) # Минимальное значение
mx = max(a, b, c) # Максимальное значение
md = a + b + c - mn - mx # Среднее значение

# Отобразим результат
print("Числа в порядке возрастания:")
print(" ", mn)
print(" ", md)
print(" ", mx)
```

Поскольку слова `min` и `max` являются в Python зарезервированными, мы не можем использовать их для именования переменных. Вместо этого мы будем хранить значения минимума и максимума в переменных с именами `mn` и `mx`.

## *Упражнение 34. Вчерашний хлеб*

```
##
# Вычисляем стоимость вчерашнего хлеба
#
BREAD_PRICE = 3.49
DISCOUNT_RATE = 0.60

# Запрашиваем данные у пользователя
num_loaves = int(input("Введите количество вчерашних буханок хлеба: "))

# Вычисляем скидку и общую стоимость
regular_price = num_loaves * BREAD_PRICE
discount = regular_price * DISCOUNT_RATE
total = regular_price - discount

# Отобразим результат в нужном формате
print("Номинальная цена: %5.2f" % regular_price)
```

```
print("Сумма скидки: %5.2f" % discount)
print("Итого: %5.2f" % total)
```

Формат %5.2f предполагает использование пяти знакомест для отображения чисел, при этом под десятичные знаки должно быть отведено два места. Это поможет внешне выровнять столбцы в таблице при разном количестве цифр в значениях.

# Глава 10

# Принятие решений

## *Упражнение 35. Чет или нечет?*

```
##
# Определяем и выводим на экран информацию о том, четное введенное число или нечетное
#
# Запрашиваем целое число у пользователя
num = int(input("Введите целое число: "))

# Используем оператор взятия остатка от деления
if num % 2 == 1:
    print(num, "нечетное.")
else:
    print(num, "четное.")
```

Остаток от деления четного числа на 2 всегда будет давать 0, а нечетного – 1.

## *Упражнение 37. Гласные и согласные*

```
##
# Определяем, является буква гласной или согласной
#
# Запрашиваем ввод буквы с клавиатуры
letter = input("Введите букву латинского алфавита: ")

# Классифицируем букву и выводим результат
if letter == "a" or letter == "e" or \
        letter == "i" or letter == "o" or \
        letter == "u":
    print("Это гласная буква.")
elif letter == "y":
    print("Иногда буква гласная, иногда согласная.")
else:
    print("Это согласная буква.")
```

Эта версия программы работает только для ввода букв в нижнем регистре. Вы можете добавить проверку на заглавные буквы, если есть такая необходимость.

## *Упражнение 38. Угадайте фигуру*

```
##
# Определяем вид фигуры по количеству сторон
```

```
#
# Запрашиваем у пользователя количество сторон
nsides = int(input("Введите количество сторон фигуры: "))

# Определяем вид фигуры, оставляя его пустым, если введено некорректное число
name = ""
if nsides == 3:
    name = "треугольник"
elif nsides == 4:
    name = "прямоугольник"
elif nsides == 5:
    name = "пятиугольник"
elif nsides == 6:
    name = "шестиугольник"
elif nsides == 7:
    name = "семиугольник"
elif nsides == 8:
    name = "восьмиугольник"
elif nsides == 9:
    name = "девятиугольник"
elif nsides == 10:
    name = "десятиугольник"

# Выводим ошибку ввода
if name == "":
    print("Введенное количество сторон не поддерживается программой.")
else:
    print("Эта фигура:", name)
```

Пустую строку мы используем как индикатор. Если введенное пользователем значение окажется за границами обрабатываемого нами диапазона, значение переменной `name` останется пустым, что позволит нам позже вывести сообщение об ошибке.

## *Упражнение 39. Сколько дней в месяце?*

```
##
# Определяем количество дней в месяце
#
# Запрашиваем у пользователя название месяца
month = input("Введите название месяца: ")

# Вычисляем количество дней в месяце
days = 31
```

Изначально считаем, что в месяце 31 день, после чего постепенно корректируем это предположение.

```
if month == "Апрель" or month == "Июнь" or \
        month == "Сентябрь" or month == "Ноябрь":
    days = 30
```

```
elif month == "Февраль":
    days = "28 или 29"
```

Для февраля присваиваем переменной days строковое значение. Это позволит нам вывести информацию о том, что в этом месяце количество дней может варьироваться.

```
# Выводим результат
print("Количество дней в месяце", month, "равно", days)
```

## *Упражнение 41. Классификация треугольников*

```
##
# Классифицируем треугольники на основании длин их сторон
#
# Запрашиваем у пользователя длины сторон треугольника
side1 = float(input("Введите длину первой стороны: "))
side2 = float(input("Введите длину второй стороны: "))
side3 = float(input("Введите длину третьей стороны: "))

# Определяем вид треугольника
if side1 == side2 and side2 == side3:
    tri_type = "равносторонний"
elif side1 == side2 or side2 == side3 or \
        side3 == side1:
    tri_type = "равнобедренный"
else:
    tri_type = "разносторонний"

# Отображаем вид треугольника
print("Это", tri_type, "треугольник")
```

При проверке того, является ли треугольник равносторонним, можно было добавить условие равенства side1 и side3, но в этом нет необходимости, поскольку оператор равенства (==) является транзитивным.

## *Упражнение 42. Узнать частоту по ноте*

```
##
# Преобразуем название ноты в частоту
#
C4_FREQ = 261.63
D4_FREQ = 293.66
E4_FREQ = 329.63
F4_FREQ = 349.23
G4_FREQ = 392.00
A4_FREQ = 440.00
B4_FREQ = 493.88

# Запрашиваем у пользователя название ноты
name = input("Введите название ноты в виде буквы и цифры, например C4: ")
```

```
# Сохраняем название ноты и номер октавы в разных переменных
note = name[0]
octave = int(name[1])

# Получаем частоту ноты четвертой октавы
if note == "C":
    freq = C4_FREQ
elif note == "D":
    freq = D4_FREQ
elif note == "E":
    freq = E4_FREQ
elif note == "F":
    freq = F4_FREQ
elif note == "G":
    freq = G4_FREQ
elif note == "A":
    freq = A4_FREQ
elif note == "B":
    freq = B4_FREQ

# Адаптируем частоту к конкретной октаве
freq = freq / 2 ** (4 - octave)

# Выводим результат
print("Частота ноты", name, "равна", freq)
```

## *Упражнение 43. Узнать ноту по частоте*

```
##
# Запрашиваем у пользователя частоту ноты и определяем ее название
#
C4_FREQ = 261.63
D4_FREQ = 293.66
E4_FREQ = 329.63
F4_FREQ = 349.23
G4_FREQ = 392.00
A4_FREQ = 440.00
B4_FREQ = 493.88
LIMIT = 1

# Запрашиваем у пользователя частоту ноты
freq = float(input("Введите частоту ноты (Гц): "))

# Определяем ноту по частоте. Если нота не найдена, присваиваем переменной
# пустую строку
if freq >= C4_FREQ - LIMIT and freq <= C4_FREQ + LIMIT:
    note = "C4"
elif freq >= D4_FREQ - LIMIT and freq <= D4_FREQ + LIMIT:
    note = "D4"
```

```
elif freq >= E4_FREQ - LIMIT and freq <= E4_FREQ + LIMIT:
    note = "E4"
elif freq >= F4_FREQ - LIMIT and freq <= F4_FREQ + LIMIT:
    note = "F4"
elif freq >= G4_FREQ - LIMIT and freq <= G4_FREQ + LIMIT:
    note = "G4"
elif freq >= A4_FREQ - LIMIT and freq <= A4_FREQ + LIMIT:
    note = "A4"
elif freq >= B4_FREQ - LIMIT and freq <= B4_FREQ + LIMIT:
    note = "B4"
else:
    note = ""

# Отображаем результат или сообщение об ошибке
if note == "":
    print("Ноты с заданной частотой не существует.")
else:
    print("Введенная частота примерно соответствует ноте", note)
```

## *Упражнение 47. Определение времени года*

```
##
# Определяем и выводим название сезона по дате
#
# Запрашиваем у пользователя дату
month = input("Введите название месяца: ")
day = int(input("Введите день: "))
```

> Представленное решение содержит множество блоков `elif`, чтобы максимально упростить сценарий. Можно уменьшить количество блоков `elif` за счет усложнения общей условной конструкции.

```
# Определяем сезон
if month == "Январь" or month == "Февраль":
    season = "Зима"
elif month == "Март":
    if day < 20:
        season = "Зима"
    else:
        season = "Весна"
elif month == "Апрель" or month == "Май":
    season = "Весна"
elif month == "Июнь":
    if day < 21:
        season = "Весна"
    else:
```

```
        season = "Лето"
elif month == "Июль" or month == "Август":
    season = "Лето"
elif month == "Сентябрь":
    if day < 22:
        season = "Лето"
    else:
        season = "Осень"
elif month == "Октябрь" or month == "Ноябрь":
    season = "Осень"
elif month == "Декабрь":
    if day < 21:
        season = "Осень"
    else:
        season = "Зима"

# Выводим результат
print(month, day, "соответствует сезону", season)
```

## *Упражнение 49. Китайский гороскоп*

```
##
# Определяем животное, ассоциированное с введенным годом в китайском гороскопе
#
# Запрашиваем у пользователя год
year = int(input("Введите год: "))

# Определяем ассоциированное с годом животное
if year % 12 == 8:
    animal = "Дракон"
elif year % 12 == 9:
    animal = "Змея"
elif year % 12 == 10:
    animal = "Лошадь"
elif year % 12 == 11:
    animal = "Коза"
elif year % 12 == 0:
    animal = "Обезьяна"
elif year % 12 == 1:
    animal = "Петух"
elif year % 12 == 2:
    animal = "Собака"
elif year % 12 == 3:
    animal = "Свинья"
elif year % 12 == 4:
    animal = "Крыса"
elif year % 12 == 5:
    animal = "Бык"
```

```
elif year % 12 == 6:
    animal = "Тигр"
elif year % 12 == 7:
    animal = "Кролик"

# Выводим результат
print("Год %d ассоциирован с животным: %s." % (year, animal))
```

При форматировании нескольких элементов в одной строке они перечисляются через запятую в круглых скобках справа от оператора %.

## *Упражнение 52. Буквенные оценки – в числовые*

```
##
# Преобразуем буквенные оценки в числовые
#
A = 4.0
A_MINUS = 3.7
B_PLUS = 3.3
B = 3.0
B_MINUS = 2.7
C_PLUS = 2.3
C = 2.0
C_MINUS = 1.7
D_PLUS = 1.3
D = 1.0
F = 0
INVALID = -1

# Запрашиваем буквенную оценку у пользователя
letter = input("Введите буквенную оценку: ")
letter = letter.upper()
```

Инструкция `letter = letter.upper()` переводит все символы нижнего регистра в верхний, сохраняя результат в первоначальную переменную. Это позволит работать с оценками, введенными в нижнем регистре, без включения в программу дополнительных условных конструкций.

```
# Преобразуем оценку из буквенной в числовую, используя значение -1 как индикатор,
# означающий ошибочный ввод
if letter == "A+" or letter == "A":
    gp = A
elif letter == "A-":
    gp = A_MINUS
```

```
elif letter == "B+":
    gp = B_PLUS
elif letter == "B":
    gp = B
elif letter == "B-":
    gp = B_MINUS
elif letter == "C+":
    gp = C_PLUS
elif letter == "C":
    gp = C
elif letter == "C-":
    gp = C_MINUS
elif letter == "D+":
    gp = D_PLUS
elif letter == "D":
    gp = D
elif letter == "F":
    gp = F
else:
    gp = INVALID

# Выводим результат
if gp == INVALID:
    print("Введена некорректная оценка.")
else:
    print("Буквенная оценка", letter, "соответствует", gp, "баллам.")
```

## *Упражнение 54. Оценка работы*

```
##
# Определение оценки работы сотрудников при помощи рейтингов от пользователя
#
RAISE_FACTOR = 2400.00
UNACCEPTABLE = 0
ACCEPTABLE = 0.4
MERITORIOUS = 0.6

# Запрашиваем у пользователя рейтинг
rating = float(input("Введите рейтинг: "))

# Классифицируем сотрудников
if rating == UNACCEPTABLE:
    performance = "низкий"
elif rating == ACCEPTABLE:
    performance = "удовлетворительный"
elif rating >= MERITORIOUS:
    performance = "высокий"
else:
    performance = ""
```

```
#Выводим результат
if performance == "":
    print("Введен ошибочный рейтинг.")
else:
    print("Основываясь на введенном рейтинге, ваш уровень: %s." % \
        performance)
print("Прибавка к зарплате составит: $%.2f." % \
    (rating * RAISE_FACTOR))
```

Необходимость заключать выражение `rating * RAISE_FACTOR` в последней строке кода в скобки объясняется тем, что операторы `%` и `*` имеют равный приоритет. Добавление скобок позволило сообщить Python, что сначала нужно выполнить математическую операцию, а затем – операцию форматирования.

## *Упражнение 58. Високосный год?*

```
##
# Определяем, високосный заданный год или нет
#
# Запрашиваем у пользователя год
year = int(input("Введите год: "))

# Определяем, високосный или нет
if year % 400 == 0:
    isLeapYear = True
elif year % 100 == 0:
    isLeapYear = False
elif year % 4 == 0:
    isLeapYear = True
else:
    isLeapYear = False

# Отображаем результат
if isLeapYear:
    print(year, " - високосный год.")
else:
    print(year, " - невисокосный год.")
```

## *Упражнение 61. Действительный номерной знак машины?*

```
## Определяем формат номерного знака. Всего допустимых формата два:
# 1) 3 буквы и 3 цифры
# 2) 4 цифры 3 буквы
# Запрашиваем номер у пользователя
plate = input("Введите номерной знак машины: ")
```

```
# Проверяем номерной знак. Необходимо проверить все 6 знаков для номера старого образца
# и 7 знаков – для нового
if len(plate) == 6 and \
  plate[0] >= "A" and plate[0] <= "Z" and \
  plate[1] >= "A" and plate[1] <= "Z" and \
  plate[2] >= "A" and plate[2] <= "Z" and \
  plate[3] >= "0" and plate[3] <= "9" and \
  plate[4] >= "0" and plate[4] <= "9" and \
  plate[5] >= "0" and plate[5] <= "9":
    print("Это номерной знак старого образца.")
elif len(plate) == 7 and \
  plate[0] >= "0" and plate[0] <= "9" and \
  plate[1] >= "0" and plate[1] <= "9" and \
  plate[2] >= "0" and plate[2] <= "9" and \
  plate[3] >= "0" and plate[3] <= "9" and \
  plate[4] >= "A" and plate[4] <= "Z" and \
  plate[5] >= "A" and plate[5] <= "Z" and \
  plate[6] >= "A" and plate[6] <= "Z":
    print("Это номерной знак нового образца.")
else:
    print("Неверный номерной знак.")
```

## *Упражнение 62. Играем в рулетку*

```
##
# Определяем выпавший номер на рулетке и выигрыш
#
from random import randrange

# Симулируем запуск рулетки, используя число 37 для представления номера 00
value = randrange(0, 38)
if value == 37:
    print("Выпавший номер: 00...")
else:
    print("Выпавший номер: %d..." % value)

# Отображаем выигрыш для одного числа
if value == 37:
    print("Выигравшая ставка: 00")
else:
    print("Pay", value)

# Отображаем выигрыш по цветам
# В первой строке проверяем число на вхождение в ряд 1, 3, 5, 7 и 9
# Во второй строке проверяем число на вхождение в ряд 12, 14, 16 и 18
# В третьей строке проверяем число на вхождение в ряд 19, 21, 23, 25 и 27
# В четвертой строке проверяем число на вхождение в ряд 30, 32, 34 и 36
if value % 2 == 1 and value >= 1 and value <= 9 or \
  value % 2 == 0 and value >= 12 and value <= 18 or \
```

```
  value % 2 == 1 and value >= 19 and value <= 27 or \
  value % 2 == 0 and value >= 30 and value <= 36:
    print("Выигравшая ставка: красное")
elif value == 0 or value == 37:
    pass
else:
    print("Выигравшая ставка: черное")

# Отображаем выигрыш по чет/нечет
if value >= 1 and value <= 36:
    if value % 2 == 1:
        print("Выигравшая ставка: нечетное")
    else:
        print("Выигравшая ставка: четное")

# Отображаем выигрыш по низ/верх
if value >= 1 and value <= 18:
    print("Выигравшая ставка: от 1 до 18")
elif value >= 19 and value <= 36:
    print("Выигравшая ставка: от 19 до 36")
```

Тело блоков `if`, `elif` или `else` должно содержать по крайней мере одно выражение. В языке Python есть ключевое слово `pass`, которое можно использовать, когда требуется выражение, но никаких операций выполнять не нужно.

# Глава 11

# Повторения

## *Упражнение 66. Никаких центов*

```
##
# Вычислить полную сумму покупки. Сумму для уплаты наличными округлить до
# ближайших пяти центов, поскольку одноцентовые монеты выведены из обращения
#
PENNIES_PER_NICKEL = 5
NICKEL = 0.05
```

> Хотя очень маловероятно, что в будущем в пятицентовой монете может оказаться больше пяти центов, мы все равно используем в программе переменные на случай, если когда-то нужно будет адаптировать ее для использования с другими номиналами монет.

```
# Собираем общую сумму
total = 0.00

# Запрашиваем цену первого товара как строку
line = input("Введите цену товара (пустая строка для выхода): ")

# Продолжаем запрашивать цены товаров, пока строка не будет оставлена пустой
while line != "":
    # Добавляем цену в общей сумме (после перевода ее в число с плавающей запятой)
    total = total + float(line)
    # Запрашиваем цену следующего товара
    line = input("Введите цену товара (пустая строка для выхода): ")

# Показываем полную сумму к оплате
print("Полная сумма к оплате: %.02f" % total)

# Считаем, сколько центов осталось бы, если бы мы оплатили всю покупку 5-центовыми
# монетами
rounding_indicator = total * 100 % PENNIES_PER_NICKEL
if rounding_indicator < PENNIES_PER_NICKEL / 2:
```

```
    # Если оставшаяся сумма центов меньше 2,5, округляем значение путем вычитания
    # полученного количества центов из общей суммы
    cash_total = total - rounding_indicator / 100
else:
    # Иначе добавляем 5 центов и затем вычитаем нужное количество центов
    cash_total = total + NICKEL - rounding_indicator / 100

# Выводим итоговую сумму для оплаты
print("Сумма для оплаты наличными: %.02f" % cash_total)
```

## *Упражнение 67. Найти периметр многоугольника*

```
##
# Рассчитаем периметр многоугольника, построенного на основании координат точек,
# введенных пользователем. Пустая строка завершает ввод данных
#
from math import sqrt

# Переменная для хранения периметра многоугольника
perimeter = 0

# Запрашиваем координаты первой точки
first_x = float(input("Введите первую координату X: "))
first_y = float(input("Введите первую координату Y: "))

# Инициализируем координаты предыдущей точки начальными значениями
prev_x = first_x
prev_y = first_y

# Запрашиваем остальные координаты
line = input("Введите следующую координату X (Enter для окончания ввода): ")
while line != "":
    # Преобразуем координату X в число и запрашиваем координату Y
    x = float(line)
    y = float(input("Введите следующую координату Y: "))

    # Рассчитываем расстояние до предыдущей точки и добавляем к периметру
    dist = sqrt((prev_x - x) ** 2 + (prev_y - y) ** 2)
    perimeter = perimeter + dist

    # Устанавливаем значения предыдущих координат
    # для следующей итерации
    prev_x = x
    prev_y = y

    # Запрашиваем следующую координату X
    line = input("Введите следующую координату X (Enter для окончания ввода): ")

# Рассчитываем расстояние от последней точки до первой и добавляем к периметру
dist = sqrt((first_x - x) ** 2 + (first_y - y) ** 2)
```

Расстояние между точками на плоскости можно рассчитать по теореме Пифагора.

```
perimeter = perimeter + dist

# Отображаем результат
print("Периметр многоугольника равен", perimeter)
```

## Упражнение 69. Билеты в зоопарк

```
##
# Рассчитать стоимость посещения зоопарка для группы
#
# Константы для хранения цен на разные категории билетов
BABY_PRICE = 0.00
CHILD_PRICE = 14.00
ADULT_PRICE = 23.00
SENIOR_PRICE = 18.00

# Сохраним как константы возрастные ограничения
BABY_LIMIT = 2
CHILD_LIMIT = 12
ADULT_LIMIT = 64

# Переменная для хранения общей суммы посещения
total = 0

# Читаем ввод пользователя до пустой строки
line = input("Введите возраст посетителя (пустая строка для окончания ввода): ")
while line != "":
    age = int(line)

    # Добавляем цену билета к общей сумме
    if age <= BABY_LIMIT:
        total = total + BABY_PRICE
    elif age <= CHILD_LIMIT:
        total = total + CHILD_PRICE
    elif age <= ADULT_LIMIT:
        total = total + ADULT_PRICE
    else:
        total = total + SENIOR_PRICE

    # Считываем возраст следующего посетителя
    line = input("Введите возраст посетителя (пустая строка для окончания ввода): ")

# Отображаем сумму группового посещения с правильным форматированием
print("Сумма посещения зоопарка для этой группы составит $%.2f" % total)
```

С нынешними правилами первый блок `if-elif-else` в нашей программе можно и не писать. Но он пригодится, если власти отменят бесплатное посещение зоопарка для маленьких деток.

## Упражнение 70. Биты четности

```
##
# Рассчитать значение бита четности для набора из 8 бит, введенного пользователем
```

```
#
# Запрашиваем первые 8 бит
line = input("Введите 8 бит информации: ")
# Продолжаем цикл, пока пользователь не введет пустую строку
while line != "":
    # Убеждаемся в правильности ввода пользователя
    if line.count("0") + line.count("1") != 8 or len(line) != 8:
        # Выводим сообщение об ошибке
        print("Это не 8 бит… Попробуйте еще.")
    else:
        # Считаем единички
        ones = line.count("1")
```

Метод count возвращает количество вхождений указанной подстроки в строке, к которой применен.

```
        # Отображаем значение бита четности
        if ones % 2 == 0:
            print("Бит четности должен иметь значение 0.")
        else:
            print("Бит четности должен иметь значение 1.")

    # Считываем следующий ввод пользователя
    line = input("Введите 8 бит информации: ")
```

## *Упражнение 73. Код Цезаря*

```
##
# Реализовать код Цезаря, в котором используются символы, сдвинутые
# на определенное количество позиций.
# Отрицательные значения сдвига можно использовать для декодирования.
#
# Запрашиваем у пользователя сообщение и сдвиг
message = input("Введите сообщение: ")
shift = int(input("Введите сдвиг: "))

# Обрабатываем каждый символ для создания зашифрованного сообщения
new_message = ""
for ch in message:
    if ch >= "a" and ch <= "z":
        # Обрабатываем букву в нижнем регистре, определяя ее позицию
        # в алфавите (0-25), вычисляя новую позицию и добавляя букву в сообщение
        pos = ord(ch) - ord("a")
        pos = (pos + shift) % 26
        new_char = chr(pos + ord("a"))
        new_message = new_message + new_char
```

Функция *ord* преобразует символ в целочисленную позицию в таблице ASCII. Функция *chr* возвращает символ в таблице ASCII по позиции, переданной в качестве аргумента.

```
    elif ch >= "A" and ch <= "Z":
        # Обрабатываем букву в верхнем регистре, определяя ее позицию
        # в алфавите (0-25), вычисляя новую позицию и добавляя букву в сообщение
        pos = ord(ch) - ord("A")
        pos = (pos + shift) % 26
        new_char = chr(pos + ord("A"))
        new_message = new_message + new_char
    else:
        # Если это не буква, просто сохраняем ее в сообщении
        new_message = new_message + ch
# Отображаем полученное сообщение
print("Новое сообщение", new_message)
```

## *Упражнение 75. Палиндром или нет?*

```
##
# Определить, является ли введенная пользователем строка палиндромом
#

# Запрашиваем строку у пользователя
line = input("Введите строку: ")

# Предполагаем, что это палиндром, пока не доказано обратное
is_palindrome = True

# Сравниваем символы, начиная с двух сторон. Продолжаем, пока не достигнем середины или
# не поймем, что это не палиндром
i = 0
while i < len(line) / 2 and is_palindrome:
    # Если символы не равны, сразу понимаем, что это не палиндром
    if line[i] != line[len(line) - i - 1]:
        is_palindrome = False

    # Переходим к следующему символу
    i = i + 1

# Вывод результата
if is_palindrome:
    print(line, " - это палиндром")
else:
    print(line, " - это не палиндром")
```

## Упражнение 77. Таблица умножения

```
##
# Вывести таблицу умножения от 1 до 10
#
MIN = 1
MAX = 10

# Строка заголовков
print("    ", end = "")
for i in range(MIN, MAX + 1):
    print("%4d" % i, end = "")
print()

# Выводим таблицу
for i in range(MIN, MAX + 1):
    print("%4d" % i, end = "")
    for j in range(MIN, MAX + 1):
        print("%4d" % (i * j), end = "")
    print()
```

Указание в качестве последнего аргумента функции print выражения end = “” позволяет избежать принудительного перевода строки.

## Упражнение 79. Наибольший общий делитель

```
##
# Рассчитаем наибольший общий делитель для двух целых чисел с использованием цикла
#
# Запрашиваем у пользователя два целых числа
n = int(input("Введите первое целое число: "))
m = int(input("Введите второе целое число: "))

# Присваиваем d наименьшее из n и m
d = min(n, m)

# В цикле находим наибольший общий делитель для двух чисел
while n % d != 0 or m % d != 0:
    d = d - 1

# Выводим результат
print("Наибольший общий делитель для", n, "и", m, "равен", d)
```

## Упражнение 82. Десятичное число в двоичное

```
##
# Перевести десятичное число в двоичное
#
NEW_BASE = 2

# Запрашиваем число для перевода у пользователя
num = int(input("Введите неотрицательное целое число: "))
```

```
# Будем сохранять результат в переменной result
result = ""
q = num
```

Алгоритм, предложенный здесь, выражен при помощи цикла *repeat-until* (повторяй-пока), но за неимением в Python такого типа циклов нам пришлось адаптировать алгоритм для работы с циклом `while`. Как пример, этого можно достигнуть путем дублирования тела цикла и размещения его непосредственно перед циклом.

```
# Пишем копию тела цикла вне самого цикла
r = q % NEW_BASE
result = str(r) + result
q = q // NEW_BASE

# Выполняем цикл, пока q не станет равен нулю
while q > 0:
    r = q % NEW_BASE
    result = str(r) + result
    q = q // NEW_BASE

# Отображаем результат
print(num, "в десятичной системе равно", result, "в двоичной.")
```

## *Упражнение 83. Максимальное число в последовательности*

```
##
# Находим максимумы в случайном ряду из 100 целых чисел
# и считаем количество обновлений максимального значения
#
from random import randrange

NUM_ITEMS = 100

# Генерируем и выводим первое число
mx_value = randrange(1, NUM_ITEMS + 1)
print(mx_value)

# В этой переменной будем накапливать количество обновлений максимума
num_updates = 0

# Проходим по числам
for i in range(1, NUM_ITEMS):
    # Генерируем новое случайное число
    current = randrange(1, NUM_ITEMS + 1)

    # Если оно превышает текущий максимум...
```

```
    if current > mx_value:
        # Обновляем максимум и увеличиваем счетчик на единицу
        mx_value = current
        num_updates = num_updates + 1
        # Отображаем значение с пометкой
        print(current, "<== Обновление")
    else:
        # Отображаем значение
        print(current)

# Отображаем результаты
print("Максимальное значение в ряду:", mx_value)
print("Количество смен максимального значения:", num_updates)
```

# Глава 12

# Функции

## *Упражнение 88. Медиана трех значений*

```
##
# Рассчитываем и выводим на экран медиану трех чисел, введенных пользователем
# В этой программе реализованы две техники вычисления медианы для демонстрации
# разных подходов к решению одной и той же задачи
#

## Рассчитываем медиану трех чисел при помощи блока if
# @param a – первое значение
# @param b – второе значение
# @param c – третье значение
# @return медиана чисел a, b и c
#
def median(a, b, c):
    if a < b and b < c or a > b and b > c:
        return b
    if b < a and a < c or b > a and a > c:
        return a
    if c < a and b < c or c > a and b > c:
        return c

## Рассчитываем медиану трех чисел при помощи функций min и max и капельки арифметики
# @param a – первое значение
# @param b – второе значение
# @param c – третье значение
# @return медиана чисел a, b и c
#
def alternateMedian(a, b, c):
    return a + b + c - min(a, b, c) - max(a, b, c)

# Выводим медиану чисел, введенных пользователем
def main():
    x = float(input("Введите первое число: "))
    y = float(input("Введите второе число: "))
```

Каждая функция, которую вы пишете, должна начинаться с комментария. Строки, начинающиеся с конструкции @param, используются для описания параметров. Строка, начинающаяся с @return, описывает возвращаемое значение.

Медиана трех чисел равна их сумме за вычетом минимального и максимального значений.

```
    z = float(input("Введите третье число: "))
    print("Медиана равна:", median(x, y, z))
    print("С помощью альтернативного метода:", \
        alternateMedian(x, y, z))

# Вызываем основную функцию
main()
```

## *Упражнение 90. Двенадцать дней Рождества*

```
##
# Отображаем полный текст песни The Twelve Days of Christmas.
#
from int_ordinal import intToOrdinal
```

> Функция `intToOrdinal`, написанная вами для упражнения 89, импортируется здесь, чтобы не нужно было ее дублировать.

```
## Отображаем один куплет песни The Twelve Days of Christmas
# @param n - куплет для отображения
# @return (None)
def displayVerse(n):
    print("On the", intToOrdinal(n), "day of Christmas")
    print("my true love sent to me:")

    if n >= 12:
        print("Twelve drummers drumming,")
    if n >= 11:
        print("Eleven pipers piping,")
    if n >= 10:
        print("Ten lords a-leaping,")
    if n >= 9:
        print("Nine ladies dancing,")
    if n >= 8:
        print("Eight maids a-milking,")
    if n >= 7:
        print("Seven swans a-swimming,")
    if n >= 6:
        print("Six geese a-laying,")
    if n >= 5:
        print("Five golden rings,")
    if n >= 4:
        print("Four calling birds,")
    if n >= 3:
        print("Three French hens,")
    if n >= 2:
```

```
        print("Two turtle doves,")
    if n == 1:
        print("A", end=" ")
    else:
        print("And a", end=" ")
    print("partridge in a pear tree.")
    print()

# Отображаем все 12 куплетов песни
def main():
    for verse in range(1, 13):
        displayVerse(verse)

# Вызываем основную функцию
main()
```

## *Упражнение 93. Центрируем строку*

```
##
# Центрируем строку в рамках заданного окна
#
WIDTH = 80

## Создаем новую строку, которая будет центрирована в окне заданной ширины
# @param s – строка для центрирования
# @param width – ширина окна, в котором будет центрирована строка
# @return копия строки s с ведущими пробелами для центрирования
def center(s, width):
# Если строка слишком длинная, возвращаем оригинал
    if width < len(s):
        return s

    # Вычисляем количество пробелов, необходимое для центрирования строки
    spaces = (width – len(s)) // 2
    result = " " * spaces + s
```

Оператор // используется, чтобы округлить результат деления до целого числа. Оператор / не может быть использован, поскольку возвращает число с плавающей точкой, а символ пробела можно поставить только целое количество раз.

```
    return result

# Демонстрируем центрирование строки
def main():
    print(center("Известная история", WIDTH))
    print(center("от:", WIDTH))
```

```
    print(center("Кого-то известного", WIDTH))
    print()
    print("Жили-были...")

# Call the main function
main()
```

## *Упражнение 95. Озаглавим буквы*

```
##
# Заглавные буквы в строке
#

## Озаглавливаем нужные буквы в строке
# @param s - исходная строка для обработки
# @return новая строка
def capitalize(s):
    # Создаем копию исходной строки, в которой будем собирать итоговую строку
    result = s

    # Делаем заглавной первый непробельный символ в строке
    pos = 0
    while pos < len(s) and result[pos] == ' ':
        pos = pos + 1

    if pos < len(s):
        # Заменяем символ на заглавный, не затрагивая остальные символы в строке
        result = result[0 : pos] + result[pos].upper() + \
            result[pos + 1 : len(result)]
```

Использование двоеточия внутри квадратных скобок позволяет обратиться к подстроке, которая начинается с индекса, соответствующего числу до двоеточия, и заканчивается индексом, равным числу справа от двоеточия, не включая его.

```
    # Делаем заглавной первую букву, которая следует за точкой и
    # восклицательным или вопросительным знаком
    pos = 0
    while pos < len(s):
        if result[pos] == "." or result[pos] == "!" or \
            result[pos] == "?":
            # Переходим за знак ''.'', ''!'' or ''?''
            pos = pos + 1

            # Пропускаем пробелы
            while pos < len(s) and result[pos] == " ":
                pos = pos + 1
```

```
            # Если не достигли конца строки, меняем текущий символ на заглавную букву
            if pos < len(s):
                result = result[0 : pos] + \
                    result[pos].upper() + \
                    result[pos + 1 : len(result)]

        # Идем к следующему символу
        pos = pos + 1

    # Делаем заглавными буквы i, когда им предшествует пробел, а следом идет пробел,
    # точка, восклицательный или вопросительный знак либо апостроф
    pos = 1
    while pos < len(s) - 1:
        if result[pos - 1] == " " and result[pos] == "i" and \
            (result[pos + 1] == " " or result[pos + 1] == "." or \
                result[pos + 1] == "!" or result[pos + 1] == "?" or \
                result[pos + 1] == "'"):
            # Заменяем i на I, не затрагивая другие символы
            result = result[0 : pos] + "I" + \
                result[pos + 1 : len(result)]
        pos = pos + 1

    return result

# Демонстрируем работу функции
def main():
    s = input("Введите текст: ")
    capitalized = capitalize(s)
    print("Новая версия:", capitalized)

# Вызываем основную функцию
main()
```

## *Упражнение 96. Является ли строка целым числом?*

```
##
# Определяем, представляет ли строка, введенная пользователем, целое число
#

## Посмотрим, включает ли строка целочисленное значение
# @param s - строка для проверки
# @return True, если это целое число. Иначе False.
#
def isInteger(s):
    # Удаляем пробелы в начале и конце строки
    s = s.strip()

    # Определяем, являются ли оставшиеся символы целыми числами
    if (s[0] == "+" or s[0] == "-") and s[1:].isdigit():
```

```
        return True
    if s.isdigit():
        return True
    return False
```

Метод isdigit возвращает True, если строка состоит как минимум из одного символа и все символы в ней являются цифрами.

```
# Демонстрируем работу функции isInteger
def main():
    s = input("Введите строку: ")
    if isInteger(s):
        print("Строка является целым числом.")
    else:
        print("Строка не является целым числом.")

# Вызываем основную функцию, только если файл не импортирован
if __name__ == "__main__":
    main()
```

Переменной __name__ автоматически присваивается значение при запуске программы на языке Python. При этом если файл запущен напрямую из Python, значение этой переменной будет равняться строке "__main__", а если импортирован в другую программу – имени модуля.

## *Упражнение 98. Простое число?*

```
## Определяем, является ли число простым
# @param n - целое число для проверки
# @return True, если число простое, иначе False
def isPrime(n):
    if n <= 1:
        return False

    # Проверяем все числа от двух до n, не включая его, на деление n на них без остатка
    for i in range(2, n):
        if n % i == 0:
            return False
    return True
```

Если n % i == 0, значит, n без остатка делится на i, а следовательно, оно не простое.

```
# Определяем, является ли простым введенное пользователем число
def main():
    value = int(input("Введите целое число: "))
    if isPrime(value):
        print(value, "- простое число.")
    else:
        print(value, "не является простым числом.")

# Вызываем основную функцию, только если файл не импортирован
```

```
if __name__ == "__main__":
    main()
```

## Упражнение 100. Случайный пароль

```
##
# Генерируем и отображаем случайный пароль, содержащий от 7 до 10 символов
#
from random import randint

SHORTEST = 7
LONGEST = 10
MIN_ASCII = 33
MAX_ASCII = 126

## Генерируем случайный пароль
# @return строка, содержащая случайный пароль
def randomPassword():
    # Выбираем случайную длину пароля
    randomLength = randint(SHORTEST, LONGEST)
    # Генерируем соответствующее количество случайных символов, добавляя их
к результату
    result = ""
    for i in range(randomLength):
        randomChar = chr(randint(MIN_ASCII, MAX_ASCII))
        result = result + randomChar
```

Функция chr принимает код ASCII в качестве единственного параметра и возвращает символ, соответствующий этому коду.

```
    # Возвращаем случайный пароль
    return result

# Генерируем и отображаем случайный пароль
def main():
    print("Ваш случайный пароль:", randomPassword())

# Вызываем основную функцию, только если файл не импортирован
if __name__ == "__main__":
    main()
```

## Упражнение 102. Проверка пароля на надежность

```
##
# Проверка пароля на надежность
#
```

```
## Проверяем, является ли пароль надежным. Надежным будем считать пароль, в котором
# как минимум 8 символов, есть большие и маленькие буквы, а также цифры
# @param password - пароль для проверки
# @return True, если пароль надежен, иначе False
def checkPassword(password):
    has_upper = False
    has_lower = False
    has_num = False

    # Проверяем каждый символ в пароле
    for ch in password:
        if ch >= "A" and ch <= "Z":
            has_upper = True
        elif ch >= "a" and ch <= "z":
            has_lower = True
        elif ch >= "0" and ch <= "9":
            has_num = True

    # Если пароль отвечает всем четырем требованиям
    if len(password) >= 8 and has_upper and has_lower and has_num:
        return True

    # Если как минимум одно требование не соблюдено
    return False

# Демонстрируем работу функции
def main():
    p=input("Введите пароль: ")
    if checkPassword(p):
        print("Это надежный пароль.")
    else:
        print("Это ненадежный пароль.")

# Вызываем основную функцию, только если файл не импортирован
if __name__ == "__main__":
    main()
```

## *Упражнение 105. Произвольные системы счисления*

```
##
# Переводим значение из одной системы счисления в другую. Диапазон систем - от 2 до 16
#
from hex_digit import *
```

Модуль `hex_digit` содержит функции `hex2int` и `int2hex`, которые мы написали, решая упражнение 104. Инструкция `import *` позволит загрузить все функции из модуля.

```
## Переводим число из десятичной системы в произвольную
# @param num - число в десятичной системе для преобразования
# @param new_base - основание для выходного результата
# @return строка в новой системе счисления
def dec2n(num, new_base):
    # Формируем представление числа в новой системе счисления, сохраняя в result
    result = ""
    q = num

    # Первый запуск тела будущего цикла
    r = q % new_base
    result = int2hex(r) + result
    q=q // new_base

    # Продолжаем цикл, пока q не станет равен нулю
    while q > 0:
        r = q % new_base
        result = int2hex(r) + result
        q = q // new_base

    # Возвращаем результат
    return result

## Переводим число из произвольной системы
# в десятичную
# @param num - число в системе по основанию b,
# сохраненное в виде строки
# @param b - основание преобразуемого числа
# @return число в десятичной системе счисления
def n2dec(num, b):
    decimal = 0

    # Обрабатываем каждую цифру по основанию b
    for i in range(len(num)):
        decimal = decimal * b
        decimal = decimal + hex2int(num[i])

    # Возвращаем результат
    return decimal

# Преобразуем число между произвольными системами счисления
def main():
    # Запрашиваем у пользователя исходную систему счисления и число
    from_base = int(input("Исходная система счисления (2-16): "))
    if from_base < 2 or from_base > 16:
        print("Допустимый диапазон систем счисления: от 2 до 16.")
        print("Выходим...")
        quit()

    from_num = input("Введите число по этому основанию: ")
```

Значение по основанию b должно быть представлено в виде строки, поскольку оно может содержать буквы, если основание счисления превышает 10.

```
    # Преобразуем в десятичное число и отображаем результат
    dec = n2dec(from_num, from_base)
    print("Результат: %d по основанию 10." % dec)

    # Преобразуем в число с новым основанием и отображаем результат
    to_base = int(input("Введите требуемую систему счисления (2-16): "))
    if to_base < 2 or to_base > 16:
        print("Допустимый диапазон систем счисления: от 2 до 16.")
        print("Выходим...")
        quit()

    to_num = dec2n(dec, to_base)
    print("Результат: %s по основанию %d." % (to_num, to_base))

# Вызов основной функции
main()
```

## *Упражнение 107. Максимальное сокращение дробей*

```
##
# Максимальное сокращение дробей
#

## Вычислить наибольший общий делитель для двух целых чисел
# @param n - первое число (должно быть ненулевым)
# @param m - второе число (должно быть ненулевым)
# @return наибольший общий делитель двух целых чисел
def gcd(n, m):
    # Инициализируем d как меньшее значение из n и m
    d = min(n, m)
    # Используем цикл while для поиска наибольшего общего делителя n и m
    while n % d != 0 or m % d != 0:
        d = d - 1
    return d
```

Для достижения цели функция gcd использует цикл. Также существует простой и элегантный способ определения наибольшего общего делителя двух чисел при помощи рекурсии. Эта концепция будет описана в упражнении 174.

```
## Сокращаем дробь до предела
# @param num - числитель дроби
# @param den - знаменатель дроби (должен быть ненулевым)
# @return числитель и знаменатель сокращенной дроби
def reduce(num, den):
    # Если числитель равен нулю, сокращенная дробь будет равна 0/1
    if num == 0:
```

```
        return (0, 1)

    # Находим наибольший общий делитель числителя и знаменателя
    g = gcd(num, den)

    # Делим числитель и знаменатель на НОД и возвращаем результат
    return (num // g, den // g)
```

В функции reduce мы использовали оператор //, чтобы вернуть целочисленные значения числителя и знаменателя.

```
# Запрашиваем дробь у пользователя и отображаем ее максимально сокращенный вариант
def main():
    # Запрашиваем числитель и знаменатель у пользователя
    num = int(input("Введите числитель дроби: "))
    den = int(input("Введите знаменатель дроби: "))

    # Вычисляем сокращенную дробь
    (n, d) = reduce(num, den)

    # Выводим результат
    print("Дробь %d/%d может быть сокращена до %d/%d." % (num, den, n, d))

# Вызов основной функции
main()
```

## *Упражнение 108. Переводим меры*

```
##
# Преобразуем меры объема ингредиентов в рецептах с целью их более лаконичного
# выражения
# Например, 59 чайных ложек можно сократить до 1 стакана, 3 столовых ложек
# и 2 чайных ложек.
#
TSP_PER_TBSP = 3
TSP_PER_CUP = 48

##
# Преобразуем меры объема ингредиентов в рецептах с целью их более лаконичного
# выражения
# @param num - количество единиц объема для преобразования
# @param unit - единица измерения (''cup'', ''tablespoon'' или ''teaspoon'')
# @return строка, представляющая меры в сокращенной форме
def reduceMeasure(num, unit):
    # Приводим единицу измерения к нижнему регистру
    unit = unit.lower()
```

Единицы измерения приводятся к нижнему регистру путем вызова метода lower и сохранения результата в ту же переменную. Это позволит пользователю вводить меру в любом регистре.

```
# Вычислим объем в чайных ложках
if unit == "teaspoon" or unit == "teaspoons":
    teaspoons = num
elif unit == "tablespoon" or unit == "tablespoons":
    teaspoons = num * TSP_PER_TBSP
elif unit == "cup" or unit == "cups":
    teaspoons = num * TSP_PER_CUP

# Преобразуем объем в чайных ложках в другие единицы измерения
cups = teaspoons // TSP_PER_CUP
teaspoons = teaspoons - cups * TSP_PER_CUP
tablespoons = teaspoons // TSP_PER_TBSP
teaspoons = teaspoons - tablespoons * TSP_PER_TBSP

# Создаем строковую переменную для хранения результата
result = ""

# Добавляем количество стаканов к результату (если надо)
if cups > 0:
    result = result + str(cups) + " cup"
    # Множественное число
    if cups > 1:
        result = result + "s"

# Добавляем количество столовых ложек к результату (если надо)
if tablespoons > 0:
    # Добавляем запятую, если нужно
    if result != "":
        result = result + ", "

    result = result + str(tablespoons) + " tablespoon"
    # Множественное число
    if tablespoons > 1:
        result = result + "s"

# Добавляем количество чайных ложек к результату (если надо)
if teaspoons > 0:
    # Добавляем запятую, если нужно
    if result != "":
        result = result + ", "
        result = result + str(teaspoons) + " teaspoon"
        # Множественное число
        if teaspoons > 1:
```

```
                result = result + "s"

    # Обрабатываем ноль
    if result == "":
        result = "0 teaspoons"

    return result
```

В эту программу мы включили сразу несколько вызовов функции для охвата большого числа разнообразных мер.

```
# Демонстрируем работу функции reduceMeasure путем нескольких обращений
def main():
    print("59 teaspoons is %s." % reduceMeasure(59, "teaspoons"))
    print("59 tablespoons is %s." % \
    reduceMeasure(59, "tablespoons"))
    print("1 teaspoon is %s." % reduceMeasure(1, "teaspoon"))
    print("1 tablespoon is %s." % reduceMeasure(1, "tablespoon"))
    print("1 cup is %s." % reduceMeasure(1, "cup"))
    print("4 cups is %s." % reduceMeasure(4, "cups"))
    print("3 teaspoons is %s." % reduceMeasure(3, "teaspoons"))
    print("6 teaspoons is %s." % reduceMeasure(6, "teaspoons"))
    print("95 teaspoons is %s." % reduceMeasure(95, "teaspoons"))
    print("96 teaspoons is %s." % reduceMeasure(96, "teaspoons"))
    print("97 teaspoons is %s." % reduceMeasure(97, "teaspoons"))
    print("98 teaspoons is %s." % reduceMeasure(98, "teaspoons"))
    print("99 teaspoons is %s." % reduceMeasure(99, "teaspoons"))

# Вызов основной функции
main()
```

## *Упражнение 109. Магические даты*

```
##
# Определяем все магические даты в XX веке
#
from days_in_month import daysInMonth

## Определяем, является ли дата магической
# @param day - день в дате
# @param month - месяц в дате
# @param year - год в дате
# @return True, если дата является магической, иначе False
def isMagicDate(day, month, year):
    if day * month == year % 100:
        return True
```

Выражение `year % 100` позволяет перейти к представлению года из двух цифр.

```
    return False

# Находим и отображаем все магические даты XX века
def main():
    for year in range(1900, 2000):
        for month in range(1, 13):
            for day in range(1, daysInMonth(month, year) + 1):
                if isMagicDate(day, month, year):
                    print("%02d/%02d/%04d - магическая дата." % (day, month, year))

# Вызов основной функции
main()
```

# Глава 13

# Списки

## *Упражнение 110. Порядок сортировки*

```
##
# Выводим числа, введенные пользователем, в порядке возрастания
#
# Начинаем с пустого списка
data = []

# Считываем значения и добавляем их в список, пока пользователь не введет ноль
num = int(input("Введите целое число (0 для окончания ввода): "))
while num != 0:
    data.append(num)
    num = int(input("Введите целое число (0 для окончания ввода): "))

# Сортируем значения
data.sort()
```

Вызов метода `sort` применительно к списку переставляет значения в нем, располагая их в нужном нам порядке. В данном случае этот способ подходит, поскольку нам нет необходимости сохранять копию исходного списка. Для создания копии исходного списка с отсортированными элементами можно воспользоваться функцией `sorted`. Вызов этой функции не оказывает влияния на порядок следования элементов в исходном списке, а значит, ее стоит использовать, когда в дальнейшем вам пригодятся обе версии списка: исходный и отсортированный в нужном вам порядке.

```
# Выводим числа в порядке возрастания
print("Введенные числа в порядке возрастания:")
for num in data:
    print(num)
```

## *Упражнение 112. Удаляем выбросы*

```
##
# Удаляем выбросы из набора данных
#
```

```
## Удаляем выбросы из списка значений
# @param data – список значений для обработки
# @param num_outliers – количество наименьших и наибольших элементов для удаления
# @return копия исходного списка с отсортированными значениями и
# удаленными наименьшими и наибольшими элементами
def removeOutliers(data, num_outliers):
    # Создаем копию списка с отсортированными значениями
    retval = sorted(data)

    # Удаляем num_outliers наибольших значений
    for i in range(num_outliers):
        retval.pop()

    # Удаляем num_outliers наименьших значений
    for i in range(num_outliers):
        retval.pop(0)

    # Возвращаем результат
    return retval

# Запрашиваем данные у пользователя и удаляем по два наибольших и наименьших значения
def main():
    # Запрашиваем данные у пользователя, пока он не оставит ввод пустым
    values = []
    s=input("Введите значение (Enter для окончания ввода): ")
    while s != "":
        num = float(s)
        values.append(num)
        s = input("Введите значение (Enter для окончания ввода): ")

    # Отображаем результат или соответствующее сообщение об ошибке
    if len(values) < 4:
        print("Вы ввели недостаточное количество чисел.")
    else:
        print("Список с удаленными выбросами: ", \
                removeOutliers(values, 2))
        print("Исходный список: ", values)

# Вызов основной функции
main()
```

Наибольшие и наименьшие значения в списке можно было удалить и в одном цикле. В данном случае используется два цикла, чтобы решение было более понятным.

## *Упражнение 113. Избавляемся от дубликатов*

```
##
# Считываем ряд слов, введенных пользователем, и выводим их без дубликатов
# в том же порядке, в котором они были введены
#

# Запрашиваем слова у пользователя и сохраняем их в список
words = []
```

```
word = input("Введите слово (Enter для окончания ввода): ")
while word != "":
    # Добавляем слово в список, только если
    # оно уже не присутствует в нем
    if word not in words:
        words.append(word)

    # Запрашиваем следующее слово у пользователя
    word = input("Введите слово (Enter для окончания ввода): ")

# Отображаем уникальные слова
for word in words:
    print(word)
```

Выражения `word not in words` и `not (word in words)` эквивалентны.

## *Упражнение 114. Отрицательные, положительные и нули*

```
##
# Запрашиваем коллекцию целых чисел у пользователя. Отображаем сначала отрицательные,
# затем нули и после этого положительные
#

# Создаем три списка для хранения отрицательных, нулевых и положительных значений
negatives = []
zeros = []
positives = []
```

В данном решении используется отдельный список для хранения введенных пользователем нулей. Но в этом нет особой необходимости, поскольку нули все одинаковые. Достаточно было бы хранить количество введенных пользователем нулей и при выводе отображать их столько, сколько надо.

```
# Запрашиваем числа у пользователя, помещая их в соответствующие списки
line = input("Введите целое число (Enter для окончания ввода): ")
while line != "":
    num = int(line)
    if num < 0:
        negatives.append(num)
    elif num > 0:
        positives.append(num)
    else:
        zeros.append(num)

    # Запрашиваем следующее число у пользователя
    line = input("Введите целое число (Enter для окончания ввода): ")

# Выводим сначала отрицательные числа, затем нули и после этого положительные
print("Введенные числа: ")
for n in negatives:
```

```
    print(n)
for n in zeros:
    print(n)
for n in positives:
    print(n)
```

## *Упражнение 116. Совершенные числа*

```
##
# Целое число n называется совершенным, если сумма всех его собственных делителей
# равна самому числу n
# Покажем все совершенные числа от 1 до LIMIT.
#
from proper_divisors import properDivisors

LIMIT = 10000

## Определяем, является ли число совершенным
# @param n - число, которое необходимо проверить на совершенство
# @return True, если число совершенно, иначе False
def isPerfect(n):
    # Получим список собственных
    # делителей числа
    divisors = properDivisors(n)

    # Рассчитываем их сумму
    total = 0
    for d in divisors:
        total = total + d

    # Определяем, является ли число совершенным, и возвращаем результат
    if total == n:
        return True
    return False

# Отображаем все совершенные числа от 1 до LIMIT.
def main():
    print("Совершенные числа от 1 до", LIMIT, ":")
    for i in range(1, LIMIT + 1):
        if isPerfect(i):
            print(" ", i)

#Call the main function
main()
```

Сумма элементов списка также может быть вычислена при помощи функции `sum` языка Python. Это позволит избежать написания цикла и сократить операцию до одной строки.

## *Упражнение 120. Форматирование списка*

```
##
# Отображаем перечень введенных слов через запятую и с союзом "и" между
# последними двумя словами
#
```

```
## Форматируем список с запятыми и союзом "и"
#@param items - список элементов для форматирования
#@return строка с установленными правилами форматирования
def formatList(items):
    # Отдельно рассматриваем пустой список и список из одного элемента
    if len(items) == 0:
        return "<пусто>"
    if len(items) == 1:
        return str(items[0])

    # Идем по всем элементам списка, за исключением двух последних
    result = ""
    for i in range(0, len(items) - 2):
        result = result + str(items[i]) + ", "
```

Каждый введенный элемент мы явным образом преобразуем в строку путем вызова функции `str` перед выполнением форматирования. Это позволит функции `formatList` корректно обрабатывать не только строковые элементы, но и числовые.

```
    # Добавляем к строке два последних элемента, разделенных союзом "и"
    result = result + str(items[len(items) - 2]) + " и "
    result = result + str(items[len(items) - 1])
    # Возвращаем результат
    return result

# Запрашиваем у пользователя слова и форматируем их
def main():
    # Запрашиваем у пользователя слова, пока не будет пропущена строка ввода
    items = []
    line = input("Введите слово (Enter для окончания ввода): ")

    while line != "":
        items.append(line)
        line = input("Введите слово (Enter для окончания ввода): ")

    # Форматируем и отображаем результат
    print("Введенные элементы: %s." % formatList(items))

# Вызов основной функции
main()
```

## *Упражнение 121. Случайные лотерейные номера*

```
##
# Собираем уникальные случайные номера для лотерейного билета
#
from random import randrange
```

```
MIN_NUM = 1
MAX_NUM = 49
NUM_NUMS = 6
```

Использование констант поможет быстро адаптировать программу для любой лотереи.

```
# Используем список для хранения номеров лотерейного билета
ticket_nums = []

# Генерируем NUM_NUMS случайных, но уникальных значений
for i in range(NUM_NUMS):
    # Генерируем номер, которого еще нет в списке
    rand = randrange(MIN_NUM, MAX_NUM + 1)
    while rand in ticket_nums:
        rand = randrange(MIN_NUM, MAX_NUM + 1)

    # Добавляем номер к билету
    ticket_nums.append(rand)

# Сортируем номера по возрастанию и отображаем их
ticket_nums.sort()
print("Номера вашего билета: ", end="")
for n in ticket_nums:
    print(n, end=" ")
print()
```

## *Упражнение 125. Тасуем колоду карт*

```
##
# Создаем колоду карт и перетасовываем ее
#
from random import randrange

## Генерируем стандартную колоду карт с четырьмя мастями и 13 номиналами в каждой
# @return список карт с каждой картой, представленной двумя символами
def createDeck():
    # Создаем список для хранения карт
    cards = []

    # Проходим по всем мастям и номиналам
    for suit in ["s", "h", "d", "c"]:
        for value in ["2", "3", "4", "5", "6", "7", "8", "9", \
                      "T", "J", "Q", "K", "A"]:

            # Генерируем карту и добавляем ее в колоду
            cards.append(value + suit)

    # Возвращаем целую колоду
    return cards

## Тасуем колоду, переданную в функцию в качестве параметра
# @param cards - список карт для тасования
```

```
# @return (None)
def shuffle(cards):
    # Проходим по картам
    for i in range(0, len(cards)):
        # Выбираем случайный индекс между текущим индексом и концом списка
        other_pos = randrange(i, len(cards))

        # Меняем местами текущую карту со случайно выбранной
        temp = cards[i]
        cards[i] = cards[other_pos]
        cards[other_pos] = temp

# Отображаем колоду до и после тасования
def main():
    cards = createDeck()
    print("Исходная колода карт: ")
    print(cards)
    print()

    shuffle(cards)
    print("Перетасованная колода карт: ")
    print(cards)

# Вызываем основную функцию, только если программа не была импортирована как модуль
if __name__ == "__main__":
    main()
```

## *Упражнение 128. Подсчитать элементы в списке*

```
##
# Подсчитываем количество элементов в списке, больших или равных
# заданному минимуму и меньших заданного максимума
#

## Определяем, сколько элементов в списке больше или равны
# заданному минимуму и меньше заданного максимума
# @param data - список значений для обработки
# @param mn - минимальная граница
# @param mx - максимальная граница
# @return количество элементов e, отвечающее условию mn <= e < mx
def countRange(data, mn, mx):
    # Подсчитываем количество элементов в списке из указанного диапазона
    count = 0
    for e in data:
        # Проверяем каждый элемент
        if mn <= e and e < mx:
            count = count + 1

    # Возвращаем результат
    return count
```

```
# Демонстрируем работу функции countRange
def main():
    data = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

    # Случай, когда несколько элементов входят в диапазон
    print("Подсчитываем количество элементов в списке [1..10] между 5 и 7...")
    print("Результат: %d Ожидание: 2" % countRange(data, 5, 7))

    # Случай, когда все элементы входят в диапазон
    print("Подсчитываем количество элементов в списке [1..10] между -5 и 77...")
    print("Результат: %d Ожидание: 10" % countRange(data, -5, 77))

    # Случай, когда ни один из элементов не входит в диапазон
    print("Подсчитываем количество элементов в списке [1..10] между 12 и 17...")
    print("Результат: %d Ожидание: 0" % countRange(data, 12, 17))

    # Случай, когда список пуст
    print("Подсчитываем количество элементов в списке [] между 0 и 100...")
    print("Результат: %d Ожидание: 0" % countRange([], 0, 100))

    # Случай, когда в списке есть дубликаты
    data = [1, 2, 3, 4, 1, 2, 3, 4]
    print("Подсчитываем количество элементов в списке", data, "между 2 и 4...")
    print("Результат: %d Ожидание: 4" % countRange(data, 2, 4))

# Вызов основной программы
main()
```

## *Упражнение 129. Разбиение строки на лексемы*

```
##
# Разбиение строки, содержащей математическое выражение, на лексемы
#

## Преобразуем математическое выражение в список лексем
# @param s - строка для разбора
# @return список лексем строки s или пустой список, если возникла ошибка
def tokenList(s) :
    # Удаляем все пробелы из строки s
    s = s.replace(" ", "")

    # Проходим по символам в строке, определяя лексемы и добавляя их к списку
    tokens = []
    i = 0
    while i < len(s):
        # Обрабатываем односимвольные лексемы: *, /, ^, ( и )
        if s[i] == "*" or s[i] == "/" or s[i] == "^" or \
                s[i] == "(" or s[i] == ")" or s[i] == "+" or s[i] == "-":
            tokens.append(s[i])
            i = i + 1
```

```
        # Обрабатываем числа без лидирующих + или -
        elif s[i] >= "0" and s[i] <= "9":
            num = ""
            # Добавляем символы в лексему, пока они представляют собой цифры
            while i < len(s) and s[i] >= "0" and s[i] <= "9":
                num = num + s[i]
                i = i + 1
            tokens.append(num)

        # Наличие других символов означает недействительность выражения.
        # Возвращаем пустой список для сигнализации возникновения ошибки
        else:
            return []

    return tokens

# Запрашиваем выражение у пользователя, разбиваем на лексемы и отображаем результат
def main():
    exp = input("Введите математическое выражение: ")
    tokens = tokenList(exp)
    print("Лексемы:", tokens)

# Вызываем основную функцию, только если программа не была импортирована как модуль
if __name__ == "__main__":
    main()
```

## *Упражнение 130. Унарные и бинарные операторы*

```
##
# Устанавливаем разницу между унарными и бинарными операторами + и -
#
from token_list import tokenList

## Определяем появление унарных операторов + и - в списке лексем и
# меняем их на u+ и u- соответственно
# @param tokens - список лексем, который может содержать унарные операторы + и -
# @return список лексем с заменой унарных операторов + и - на u+ и u-
def identifyUnary(tokens):
    retval = []

    # Обрабатываем каждую лексему в списке
    for i in range(len(tokens)):
        # Если первая лексема - это + или -, то это унарный оператор
        if i == 0 and (tokens[i] == "+" or tokens[i] == "-"):
            retval.append("u" + tokens[i])
        # Если это лексема + или -, а предыдущая лексема являлась
        # оператором или открывающей скобкой, то это унарный оператор
        elif i>0and (tokens[i] == "+" or tokens[i] == "-") and \
                (tokens[i-1] == "+" or tokens[i-1] == "-" or
                tokens[i-1] == "*" or tokens[i-1] == "/" or
```

```
                tokens[i-1] == "("):
            retval.append("u" + tokens[i])
        # Любая другая лексема свидетельствует о том, что это бинарный оператор,
        # так что он добавляется к списку без изменений
        else:
            retval.append(tokens[i])

    # Возвращаем новый список лексем с измененными унарными операторами
    return retval

# Демонстрируем выполнение пометки унарных операторов
def main():
    # Запрашиваем выражение у пользователя, разбиваем на лексемы и отображаем результат
    exp = input("Введите математическое выражение: ")
    tokens = tokenList(exp)
    print("Лексемы:", tokens)
    # Идентифицируем список унарных операторов
    marked = identifyUnary(tokens)
    print("С помеченными унарными операторами: ", marked)

# Вызываем основную функцию, только если программа не была импортирована как модуль
if __name__ == "__main__":
    main()
```

## *Упражнение 134. Все подсписки заданного списка*

```
##
# Находим все подсписки в заданном списке
#

## Создаем список из всех подсписков заданного списка
# @param data – список, в котором выполняется поиск подсписков
# @return список из всех подсписков исходного списка
def allSublists(data):
    # Начинаем с добавления пустого списка
    sublists = [[]]

    # Генерируем подсписки длиной от 1 до len(data)
    for length in range(1, len(data) + 1):
        # Генерируем подсписки начиная с каждого индекса
        for i in range(0, len(data) – length + 1):
            # Добавляем найденный подсписок к общему списку
            sublists.append(data[i : i + length])

    # Возвращаем результат
    return sublists

# Демонстрируем работу функции allSublists
def main():
    print("Подсписки []: ")
```

Список с пустым списком внутри обозначается как [[]].

```
    print(allSublists([]))

    print("Подсписки [1]: ")
    print(allSublists([1]))

    print("Подсписки [1, 2]: ")
    print(allSublists([1, 2]))

    print("Подсписки [1, 2, 3]: ")
    print(allSublists([1, 2, 3]))

    print("Подсписки [1, 2, 3, 4]: ")
    print(allSublists([1, 2, 3, 4]))

# Вызов основной программы
main()
```

## *Упражнение 135. Решето Эратосфена*

```
##
# Определяем все простые числа от 2 до значения, введенного пользователем,
# при помощи алгоритма "Решето Эратосфена"
#
# Запрашиваем у пользователя конечное значение
limit = int(input("Вывести простые числа вплоть до какого значения? "))

# Создаем список для чисел от 0 до limit
nums = []
for i in range(0, limit + 1):
    nums.append(i)

# "Вычеркиваем" единицу, заменяя ее на ноль
nums[1] = 0

# Вычеркиваем числа, кратные всем найденным простым числам
p = 2
while p < limit:
    # Вычеркиваем все числа, кратные p, но не его само
    for i in range(p * 2, limit + 1, p):
        nums[i] = 0

    # Находим следующее "невычеркнутое" число
    p = p + 1
    while p < limit and nums[p] == 0:
        p = p + 1

# Отображаем результат
print("Простые числа вплоть до", limit, ":")
for i in nums:
    if nums[i] != 0:
        print(i)
```

# Глава 14

# Словари

## *Упражнение 136. Поиск по значению*

```
##
# Проводим поиск всех ключей в словаре по заданному значению
#

## Поиск в словаре по значению
# @param data - словарь для поиска
# @param value - искомое значение
# @return список (возможно, пустой) ключей, соответствующих искомому значению
def reverseLookup(data, value):
    # Создаем список ключей для заданного значения
    keys = []

    # Проверяем каждый ключ и добавляем в список,
    # если он соответствует искомому значению
    for key in data:
        if data[key] == value:
            keys.append(key)

    # Возвращаем список ключей
    return keys

# Демонстрируем работу reverseLookup
def main():
    # Словарь соответствий французских слов английским
    frEn = {"le" : "the", "la" : "the", "livre" : "book", \
            "pomme" : "apple"}

    # Показываем работу функции reverseLookup для трех случаев:
    # для множества ключей, одного ключа и отсутствия ключей
    print("Перевод английского слова 'the' на французский: ", \
            reverseLookup(frEn, "the"))
    print("Ожидаемый результат: ['le', 'la']")
    print()
    print("Перевод английского слова 'apple' на французский: ", \
```

Каждый ключ в словаре должен быть уникальным. При этом значения могут повторяться. Таким образом, поиск в словаре по значению может привести к результату, содержащему от нуля до множества ключей.

```
            reverseLookup(frEn, "apple"))
    print("Ожидаемый результат: ['pomme']")
    print()
    print("Перевод английского слова 'asdf' на французский: ", \
            reverseLookup(frEn, "asdf"))
    print("Ожидаемый результат: []")

# Вызываем основную функцию, только если файл не был импортирован в качестве модуля
if __name__ == "__main__":
    main()
```

## *Упражнение 137. Две игральные кости*

```
##
# Симуляция многократного выбрасывания двух игральных костей и сравнение
# полученных результатов с ожидаемыми
#
from random import randrange

NUM_RUNS = 1000
D_MAX = 6

## Симуляция выбрасывания двух шестигранных игральных костей
# @return общее количество очков, выпавших на двух костях
def twoDice():
    # Две кости
    d1 = randrange(1, D_MAX + 1)
    d2 = randrange(1, D_MAX + 1)

    # Возвращаем сумму
    return d1 + d2

# Симулируем многократное выбрасывание костей и отображаем результат
def main():
    # Составим словарь для ожидаемых значений
    expected = {2: 1/36, 3: 2/36, 4: 3/36, 5: 4/36, 6: 5/36, \
            7: 6/36, 8: 5/36, 9: 4/36, 10: 3/36, \
            11: 2/36, 12: 1/36}

    # Составим словарь для хранения результатов выбрасывания костей
    counts = {2: 0, 3: 0, 4: 0, 5: 0, 6: 0, 7: 0, \
            8: 0, 9: 0, 10: 0, 11: 0, 12: 0}
```

Словари изначально инициализированы ключами от 2 до 12. В словаре `expected` значения заполнены в соответствии с теорией вероятностей, тогда как в словаре `counts` изначально стоят нули, которые будут меняться в процессе симуляции выбрасывания костей.

```
    # Проведение NUM_RUNS симуляций и подсчет результатов
    for i in range(NUM_RUNS):
        t = twoDice()
        counts[t] = counts[t] + 1

    # Сравниваем ожидаемые результаты с реальными
    print("Всего  Реальный  Ожидаемый")
    print("        процент    процент")
    for i in sorted(counts.keys()):
        print("%5d %9.2f %10.2f" % \
            (i, counts[i] / NUM_RUNS * 100, expected[i] * 100))

# Вызов основной функции
main()
```

## *Упражнение 142. Уникальные символы*

```
##
# Считаем уникальные символы в строке при помощи словаря
#

# Запрашиваем строку у пользователя
s = input("Введите строку: ")

# Добавляем каждый символ в словарь со значением True. После окончания процедуры
# количество ключей в словаре и будет отражать число уникальных символов
characters = {}
for ch in s:
    characters[ch] = True
```

Каждому ключу в словаре должно соответствовать значение. Но в нашем примере эти значения никогда не будут использованы, так что мы решили применить `True`. Вместо него мы могли использовать любое другое значение.

```
# Отображаем результат
print("Строка содержит", len(characters), \
        "уникальных символов.")
```

Функция `len` возвращает количество ключей в словаре.

## *Упражнение 143. Анаграммы*

```
##
# Определяем, являются ли два введенных пользователем слова анаграммами
#
```

```
## Рассчитываем распределение частоты появления символов в строке
# @param s – строка для обработки
# @return словарь с количеством символов в строке
def characterCounts(s):
    # Создаем пустой словарь
    counts = {}

    # Обновляем словарь для каждого символа в строке
    for ch in s:
        if ch in counts:
            counts[ch] = counts[ch] + 1
        else:
            counts[ch] = 1

    # Возвращаем результат
    return counts

# Определяем, являются ли строки, введенные пользователем, анаграммами
def main():
    # Запрашиваем строки у пользователя
    s1 = input("Введите первую строку: ")
    s2 = input("Введите вторую строку: ")

    # Вызываем функцию для двух введенных строк
    counts1 = characterCounts(s1)
    counts2 = characterCounts(s2)

    # Выводим результат
    if counts1 == counts2:
        print("Введенные строки являются анаграммами.")
    else:
        print("Введенные строки не являются анаграммами.")
```

Два словаря считаются равными, если содержат одинаковое количество ключей и те же самые ассоциированные с ними значения.

```
# Вызов основной функции
main()
```

## *Упражнение 145. Эрудит*

```
##
# Используем словарь для подсчета количества очков за собранное слово в Эрудите
#

# Создаем словарь с соответствием букв и очков за них
points = {"A": 1, "B": 3, "C": 3, "D": 2, "E": 1, "F": 4, \
```

```
            "G": 2, "H": 4, "I": 1, "J": 2, "K": 5, "L": 1, \
            "M": 3, "N": 1, "O": 1, "P": 3, "Q": 10, "R": 1, \
            "S": 1, "T": 1, "U": 1, "V": 4, "W": 4, "X": 8,
            "Y": 4, "Z": 10}

# Запрашиваем у пользователя слово
word = input("Введите слово: ")

# Считаем количество очков
uppercase = word.upper()
score = 0
for ch in uppercase:
    score = score + points[ch]

# Выводим результат
print(word, "оценивается в", score, "очков.")
```

Введенное слово мы переводим в верхний регистр, чтобы не ограничивать пользователя в выборе регистра при вводе слова. Этого результата можно было добиться и путем добавления строчных букв к словарю.

## *Упражнение 146. Карточка лото*

```
##
# Создадим и отобразим случайную карточку лото
#
from random import randrange

NUMS_PER_LETTER = 15

## Создание случайной карточки для игры в лото
# @return словарь с ключами, представляющими буквы B, I, N, G и O,
# и списком номеров под каждой буквой
def createCard():
    card = {}

    # Диапазон целых чисел для букв
    lower = 1
    upper = 1 + NUMS_PER_LETTER

    # Для каждой из пяти букв
    for letter in ["B", "I", "N", "G", "O"]:
        # Создаем пустой список для буквы
        card[letter] = []

        # Генерируем случайные номера, пока не наберется пять уникальных
        while len(card[letter]) != 5:
            next_num = randrange(lower, upper)
            # Убеждаемся, что не храним дубликаты номеров
            if next_num not in card[letter]:
                card[letter].append(next_num)

        # Обновляем диапазон номеров для следующей буквы
        lower = lower + NUMS_PER_LETTER
```

```
        upper = upper + NUMS_PER_LETTER

    # Возвращаем сгенерированную карточку
    return card

## Выводим отформатированную карточку лото
# @param card – карточка лото для отображения
# @return (None)
def displayCard(card):
    # Заголовки
    print("B  I  N  G  O")

    # Отображаем номера
    for i in range(5):
        for letter in ["B", "I", "N", "G", "O"]:
            print("%2d " % card[letter][i], end="")
        print()

# Создаем случайную карточку для игры в лото и отображаем ее
def main():
    card = createCard()
    displayCard(card)

# Вызываем основную функцию, только если файл не импортирован
if __name__ == "__main__":
    main()
```

# Глава 15

# Файлы и исключения

## *Упражнение 149. Отображаем заголовок файла*

```
##
# Показываем первые 10 строк файла, имя которого передано в качестве аргумента
командной строки
#
import sys

NUM_LINES = 10

# Проверяем, что программе был передан только один аргумент командной строки
if len(sys.argv) != 2:
    print("Передайте имя файла в качестве аргумента командной строки.")
    quit()

try:
    # Открываем файл на чтение
    inf = open(sys.argv[1], "r")

    # Читаем первую строку из файла
    line = inf.readline()

    # Продолжаем цикл, пока не прочитаем 10 строк или не дойдем до конца файла
    count = 0
    while count < NUM_LINES and line != "":
        # Удаляем символ конца строки и увеличиваем счетчик
        line = line.rstrip()
        count = count + 1

        # Отображаем строку
        print(line)

        # Читаем следующую строку из файла
        line = inf.readline()

        # Закрываем файл
        inf.close()
```

По вызову функции `quit` программа мгновенно завершается.

```
except IOError:
    # Отображаем ошибку, если с чтением из файла возникли проблемы
    print("Ошибка при доступе к файлу.")
```

## Упражнение 150. Отображаем конец файла

```
##
# Показываем первые 10 строк файла, имя которого передано в качестве аргумента
командной строки
#
import sys

NUM_LINES = 10

# Проверяем, что программе был передан только один аргумент командной строки
if len(sys.argv) != 2:
    print("Передайте имя файла в качестве аргумента командной строки.")
    quit()

try:
    # Открываем файл на чтение
    inf = open(sys.argv[1], "r")

    # Читаем файл, сохраняя NUM_LINES последних строк
    lines = []
    for line in inf:
        # Добавляем последнюю прочитанную строку к концу списка
        lines.append(line)
        # Если у нас накопилось больше NUM_LINES строк, удаляем самую старую
        if len(lines) > NUM_LINES:
            lines.pop(0)

    # Закрываем файл
    inf.close()

except:
    print("Ошибка при доступе к файлу.")
    quit()

# Отображаем последние строки из файла
for line in lines:
    print(line, end="")
```

## Упражнение 151. Сцепляем файлы

```
##
# Сцепляем два или более файлов и отображаем результат
#
import sys
```

```
# Убедимся, что хотя бы один параметр был передан в качестве аргумента командной строки
if len(sys.argv) == 1:
    print("Нужно передать программе хотя бы один аргумент.")
    quit()

# Обрабатываем все файлы, имена которых были переданы в качестве аргументов
for i in range(1, len(sys.argv)):
    fname = sys.argv[i]
    try:
        # Открываем текущий файл на чтение
        inf = open(fname, "r")

        # Отображаем файл
        for line in inf:
            print(line, end="")

        # Закрываем файл
        inf.close()

    except:
        # Отображаем предупреждение, но не завершаем выполнение программы
        print("Невозможно открыть/отобразить файл", fname)
```

Элемент с индексом 0 в списке sys.argv является ссылкой на исполняемый файл Python. Поэтому наш цикл for начинает анализировать параметры с индекса 1.

## *Упражнение 156. Сумма чисел*

```
##
# Подсчитаем сумму переданных пользователем чисел, не считая нечислового ввода
#

# Запрашиваем у пользователя первое число
line = input("Введите число: ")
total = 0

# Продолжаем запрашивать числа, пока пользователь не оставит ввод пустым
while line != "":
    try:
        # Пытаемся конвертировать значение в число
        num = float(line)
        # Если все прошло успешно, прибавляем введенное число к общей сумме
        total = total + num
        print("Сумма сейчас составляет", total)
    except ValueError:
        # Отображаем предупреждение и переходим к следующему вводу
        print("Это было не число")

    # Запрашиваем следующее число
    line = input("Введите число: ")

# Отображаем сумму
print("Общая сумма:", total)
```

## *Упражнение 158. Удаляем комментарии*

```
##
# Удаляем все комментарии из файла Python (за исключением ситуаций, когда
# символ комментария указан в середине строки)
#

# Запрашиваем у пользователя имя исходного файла и открываем его
try:
    in_name = input("Введите имя файла Python: ")
    inf = open(in_name, "r")

except:
    # Выводим сообщение и завершаем работу программы в случае возникновения ошибки
    print("При открытии файла возникла проблема.")
    print("Завершение работы программы...")
    quit()

# Запрашиваем у пользователя имя итогового файла и открываем его
try:
    out_name = input("Введите имя нового файла: ")
    outf = open(out_name, "w")

except:
    # Закрываем исходный файл, показываем сообщение об ошибке и завершаем программу
    inf.close()
    print("С создаваемым файлом возникла проблема.")
    print("Завершение работы программы...")
    quit()

try:
    # Читаем строки из исходного файла, избавляем их от комментариев и
    # сохраняем в новом файле
    for line in inf:
        # Находим позицию символа комментария (-1, если его нет)
        pos = line.find("#")
        # Если комментарий есть, создаем
        # строковый срез со всеми знаками до него
        # и сохраняем обратно в переменную line
        if pos > -1:
            line = line[0 : pos]
            line = line + "\n"

        # Записываем потенциально измененную
        # строку в новый файл
        outf.write(line)

    # Закрываем файлы
    inf.close()
    outf.close()
```

Позиция символа комментария сохраняется в переменной `pos`. Так что выражение `line[0 : pos]` указывает на все символы в строке до знака комментария, не включая его.

```
except:
    # Выводим сообщение об ошибке, если что-то пошло не так
    print("При обработке файла возникла проблема.")
    print("Завершаем программу...")
```

## *Упражнение 159. Случайный пароль из двух слов*

```
##
# Генерируем пароль путем сцепления двух случайных слов. Длина пароля должна составлять
# от 8 до 10 символов, а каждое слово должно быть длиной минимум 3 символа
#
from random import randrange

WORD_FILE = "../Data/words.txt"
```

Пароль, который мы создаем, должен содержать от 8 до 10 символов. Поскольку минимальная длина слова составляет 3 символа, а в пароле должно быть два слова, получается, что в пароле не может присутствовать слово длиной более 7 символов.

```
# Читаем все слова из файла, оставляя только те из них, длина которых варьируется
# от 3 до 7 символов, и сохраняем их в список
words = []
inf = open(WORD_FILE, "r")
for line in inf:
    # Удаляем символ новой строки
    line = line.rstrip()

    # Оставляем слова длиной от 3 до 7 символов
    if len(line) >= 3 and len(line) <= 7:
        words.append(line)

# Закрываем файл
inf.close()

# Случайным образом выбираем первое слово для пароля
first = words[randrange(0, len(words))]
first = first.capitalize()

# Выбираем второе слово для пароля до тех пор, пока оно не будет подходить по размеру
password = first
while len(password) < 8 or len(password) > 10:
    second = words[randrange(0, len(words))]
    second = second.capitalize()
    password = first + second

# Отображаем случайный пароль
print("Случайный пароль:", password)
```

## *Упражнение 162. Книги без буквы E*

```
##
# Выводим процент слов, использующих все буквы в алфавите
# Также найдем букву, реже остальных встречающуюся в словах
#
WORD_FILE = "../Data/words.txt"

# Создадим словарь со счетчиком слов, содержащих каждую букву.
# Для каждой буквы инициализируем счетчик нулем
counts = {}
for ch in "ABCDEFGHIJKLMNOPQRSTUVWXYZ":
    counts[ch] = 0

# Открываем файл, обрабатываем каждое слово и обновляем словарь со счетчиками
num_words = 0
inf = open(WORD_FILE, "r")
for word in inf:
    # Переводим слово в верхний регистр и избавляемся от символа новой строки
    word = word.upper().rstrip()
    # Перед обновлением словаря нужно создать список уникальных букв в слове.
    # Иначе мы будем увеличивать счетчик для повторяющихся букв в слове.
    # Также будем игнорировать небуквенные символы
    unique = []
    for ch in word:
        if ch not in unique and ch >= "A" and ch <= "Z":
            unique.append(ch)

    # Увеличим счетчики для всех букв в списке уникальных символов
    for ch in unique:
        counts[ch] = counts[ch] + 1

    # Увеличиваем количество обработанных слов
    num_words = num_words + 1

# Закрываем файл
inf.close()

# Выводим результат для каждой буквы. Параллельно определяем, какая буква
# встречается в словах наиболее редко, чтобы вывести ее отдельно.
smallest_count = min(counts.values())
for ch in sorted(counts):
    if counts[ch] == smallest_count:
        smallest_letter = ch
    percentage = counts[ch] / num_words * 100
    print(ch, "встречается в %.2f процентах слов" % percentage)

# Отображаем самую редко используемую букву в словах
print()
print("Буква, от которой легче всего будет избавиться:", smallest_letter)
```

## *Упражнение 163. Популярные детские имена*

```
##
# Отображаем мужские и женские имена, бывшие самыми популярными как минимум
# в одном году с 1900-го по 2012-й
#

FIRST_YEAR = 1900
LAST_YEAR = 2012

## Загружаем первую строку из файла, извлекаем имя и добавляем его к списку имен,
# если его там еще нет
# @param fname - имя файла, из которого будут считываться данные
# @param names - список, к которому будем добавлять данные (если они отсутствуют)
# @return (None)
def LoadAndAdd(fname, names):
    # Открываем файл, читаем первую строку и извлекаем имя
    inf = open(fname, "r")
    line = inf.readline()
    inf.close()
    parts = line.split()
    name = parts[0]

    # Добавляем имя в список, если оно там еще не присутствует
    if name not in names:
        names.append(name)

# Отображаем мужские и женские имена, бывшие самыми популярными как минимум
# в одном году с 1900-го по 2012-й
def main():
    # Создаем списки для хранения самых популярных имен
    girls = []
    boys = []

    # Обрабатываем все годы из диапазона, читая первые строки из файлов с мужскими
и женскими именами
    for year in range(FIRST_YEAR, LAST_YEAR + 1):
        girl_fname = "../Data/BabyNames/" + str(year) + \
                     "_GirlsNames.txt"
        boy_fname = "../Data/BabyNames/" + str(year) + \
                     "_BoysNames.txt"
```

В данном решении я предположил, что файл с именами лежит не в той же папке, где файл Python. Если ваши файлы находятся в той же директории, часть пути к файлу `../Data/BabyNames/` нужно пропустить.

```
        LoadAndAdd(girl_fname, girls)
```

```
        LoadAndAdd(boy_fname, boys)

    # Выводим списки
    print("Самые популярные имена девочек:")
    for name in girls:
        print(" ", name)
    print()

    print("Самые популярные имена мальчиков: ")
    for name in boys:
        print(" ", name)

# Вызов основной функции
main()
```

## *Упражнение 167. Проверяем правильность написания*

```
##
# Находим и отображаем все слова в файле, написанные неправильно
#
from only_words import onlyTheWords
import sys

WORDS_FILE = "../Data/words.txt"

# Убедимся, что программе передано допустимое количество аргументов командной строки
if len(sys.argv) != 2:
    print("При вызове программы должен быть указан один аргумент.")
    print("Завершаем программу...")
    quit()

# Открываем файл. Выходим, если возникла ошибка
try:
    inf = open(sys.argv[1], "r")
except:
    print("Ошибка при открытии файла '%s' на чтение. Завершаем программу..." % \
            sys.argv[1])
    quit()

# Загружаем все слова в словарь. В значения ставим 0, но использовать его не будем
valid = {}
words_file = open(WORDS_FILE, "r")

for word in words_file:
    # Переводим слово в нижний регистр и удаляем символ новой строки
    word = word.lower().rstrip()

    # Добавляем слово в словарь
    valid[word] = 0

words_file.close()
```

В данном решении используются словари со словами в виде ключей и неиспользуемыми значениями. В целом эффективнее в этой ситуации будет использовать наборы (set), если вы знакомы с подобной структурой данных. Списки использовать не рекомендуется из-за их медлительности при поиске элементов.

```
# Читаем все строки из файла, добавляя слова с ошибками в соответствующий список
misspelled = []
for line in inf:
    # Избавляемся от знаков препинания при помощи функции из упражнения 117
    words = onlyTheWords(line)
    for word in words:
        # Если слово написано неправильно и отсутствует в списке, добавляем его туда
        if word.lower() not in valid and word not in misspelled:
            misspelled.append(word)

# Закрываем анализируемый файл
inf.close()

# Отображаем слова с ошибками или сообщение об их отсутствии
if len(misspelled) == 0:
    print("Все слова написаны правильно.")
else:
    print("Следующие слова написаны неправильно:")
    for word in misspelled:
        print(" ", word)
```

## *Упражнение 169. Редактирование текста в файле*

```
##
# Редактируем файл, удаляя в нем служебные слова. Отредактированная версия
# записывается в новый файл
#
# Заметьте, что в этой программе не выполняются проверки на ошибки
# и она регистрозависимая
#

# Запрашиваем у пользователя имя файла для редактирования и открываем его
inf_name = input("Введите имя файла для редактирования: ")
inf = open(inf_name, "r")

# Запрашиваем у пользователя имя файла со служебными словами и открываем его
sen_name = input("Введите имя файла со служебными словами: ")
sen = open(sen_name, "r")

# Загружаем все служебные слова в список
words = []
line = sen.readline()
```

```
while line != "":
    line = line.rstrip()
    words.append(line)
    line = sen.readline()

# Закрываем файл со служебными словами
sen.close()
```

Файл со служебными словами может быть закрыт в середине программы, поскольку все слова из него уже считаны в список.

```
# Запрашиваем у пользователя имя нового файла и открываем его
outf_name = input("Введите имя нового файла: ")
outf = open(outf_name, "w")

# Считываем все строки из исходного файла. Заменяем все служебные слова на звездочки.
# Пишем строки в новый файл.
line = inf.readline()
while line != "":
    # Ищем и заменяем служебные слова. Количество звездочек соответствует
    # длине исходного слова
    for word in words:
        line = line.replace(word, "*" * len(word))

    # Пишем измененную строку в новый файл
    outf.write(line)

    # Читаем следующую строку из исходного файла
    line = inf.readline()

# Закрываем исходный и новый файлы
inf.close()
outf.close()
```

## *Упражнение 170. Пропущенные комментарии*

```
##
# Находим и отображаем имена функций Python, которым не предшествует строка
с комментарием
#
from sys import argv

# Проверяем аргументы командной строки
if len(argv) == 1:
    print("По крайней мере одно имя файла должно присутствовать в качестве", \
            "аргумента командной строки.")
    print("Завершаем программу...")
```

```
    quit()

# Обрабатываем все файлы, имена которых переданы в виде аргументов командной строки
for fname in argv[1 : len(argv)]:
    # Попытка обработать файл
    try:
        inf = open(fname, "r")
        # Двигаясь по файлу, нам важно хранить копию предыдущей строки, чтобы
        # можно было проверить, начинается ли она с символа комментария.
        # Также нам необходимо считать строки в файле
        prev = " "
        lnum = 1
```

Переменная `prev` должна быть инициализирована строкой длиной как минимум в один символ. Иначе программа закроется аварийно, если в первой строке анализируемого файла будет начинаться определение функции.

```
        # Считываем все строки из файла
        for line in inf:
            # Если нашли функцию без комментария
            if line.startswith("def ") and prev[0] != "#":
                # Ищем первую открывающую скобку в строке, чтобы считать имя функции
                bracket_pos = line.index("(")
                name = line[4 : bracket_pos]

                # Отображаем информацию о недокументированной функции
                print("%s строка %d: %s" % (fname, lnum, name))
            # Сохраняем текущую строку и обновляем счетчик
            prev = line
            lnum = lnum + 1
        # Закрываем текущий файл
        inf.close()
    except:
        print("Возникла проблема с файлом '%s'." % fname)
        print("Идем к следующему файлу...")
```

# Глава 16

# Рекурсия

## *Упражнение 173. Сумма значений*

```
##
# Вычисление суммы значений, введенных пользователем. Для окончания ввода
# необходимо оставить ввод пустым
#

## Вычисление суммы значений, введенных пользователем
# @return сумма введенных значений
def readAndTotal():
    # Запрашиваем значение у пользователя
    line = input("Введите число (пропустите ввод для завершения): ")

    # Базовый случай: пользователь пропустил ввод, возвращаем 0
    if line == "":
        return 0
    else:
        # Рекурсивный случай: преобразуем line в число и используем рекурсию для
        # чтения следующих строк
        return float(line) + readAndTotal()

# Запрашиваем у пользователя ряд чисел и отображаем их сумму
def main():
    total = readAndTotal()
    print("Сумма введенных чисел:", total)

# Вызов основной функции
main()
```

## *Упражнение 178. Рекурсивные палиндромы*

```
##
# Определяем, является ли введенная строка палиндромом, с использованием рекурсии
#

## Определяем, является ли введенная строка палиндромом
```

```
# @param s - строка для проверки
# @return True, если палиндром, False в противном случае
def isPalindrome(s):
    # Базовый случай: пустая строка является палиндромом, так же, как и строка длиной
в один символ
    if len(s) <= 1:
        return True

    # Рекурсивный случай: строку можно считать палиндромом, если первый и последний
    # символы одинаковые, а подстрока, исключающая эти символы, также является
палиндромом
    return s[0] == s[len(s) - 1] and \
           isPalindrome(s[1 : len(s) - 1])

# Определяем, является ли введенная строка палиндромом
def main():
    # Запрашиваем строку у пользователя
    line = input("Введите строку: ")

    # Проверяем и выводим результат
    if isPalindrome(line):
        print("Это палиндром!")
    else:
        print("Это не палиндром.")

# Вызов основной функции
main()
```

## *Упражнение 180. Редакционное расстояние*

```
##
# Вычисляем и отображаем редакционное расстояние между строками
#

## Вычисляем и отображаем редакционное расстояние между строками
# @param s - первая строка
# @param t - вторая строка
# @return редакционное расстояние между этими строками
def editDistance(s, t):
    # Если одна из строк пустая, то в редакционное расстояние включается по одному
пункту
    # для вставки каждого символа из второй строки
    if len(s) == 0:
        return len(t)
    elif len(t) == 0:
        return len(s)
    else:
        cost = 0

        # Если последние символы не совпадают, устанавливаем cost в 1
```

```
        if s[len(s) - 1] != t[len(t) - 1]:
            cost = 1

        # Рассчитываем три расстояния
        d1 = editDistance(s[0 : len(s) - 1], t) + 1
        d2 = editDistance(s, t[0 : len(t) - 1]) + 1
        d3 = editDistance(s[0 : len(s) - 1] , t[0 : len(t) - 1]) + \
                        cost

        # Возвращаем минимальное
        return min(d1, d2, d3)

# Рассчитываем редакционное расстояние между двумя строками, введенными пользователем
def main():
    # Запрашиваем у пользователя две строки
    s1 = input("Введите строку: ")
    s2 = input("Введите другую строку: ")

    # Рассчитываем и отображаем редакционное расстояние
    print("Редакционное расстояние между %s и %s равно %d." % \
        (s1, s2, editDistance(s1, s2)))

# Вызов основной функции
main()
```

## *Упражнение 183. Последовательность химических элементов*

```
##
# Определяем максимально продолжительную последовательность химических элементов,
# в которой следующий элемент начинается на последнюю букву предыдущего
#
ELEMENTS_FNAME = "../Data/Elements.csv"

## Определяем максимально продолжительную последовательность химических элементов,
# в которой следующий элемент начинается на последнюю букву предыдущего
# @param start - первое слово в последовательности
# @param words - список слов, которые могут появляться в последовательности
# @return максимальная по длине последовательность слов
def longestSequence(start, words):
    # Базовый случай: Если start пустая, то и список слов пустой
    if start == "":
        return []

    # Находим самый длинный список слов, проверяя все слова, которые могут
    # появиться в последовательности
    best = []
    last_letter = start[len(start) - 1].lower()
    for i in range(0, len(words)):
```

```
        first_letter = words[i][0].lower()

        # Если первая буква следующего слова совпадает с последней буквой предыдущего
        if first_letter == last_letter:
            # Используем рекурсию для поиска последовательности-кандидата
            candidate = longestSequence(words[i], \
                    words[0 : i] + words[i + 1 : len(words)])

            # Сохраняем последовательность, если она лучшая на данный момент
            if len(candidate) > len(best):
                best = candidate

    # Возвращаем лучшую последовательность с первым словом в начале
    return [start] + best

## Загружаем все элементы из файла
# @return список всех элементов
def loadNames():
    names = []

    # Открываем файл с набором данных
    inf = open(ELEMENTS_FNAME, "r")

    # Загружаем построчно в список элементов
    for line in inf:
        line = line.rstrip()
        parts = line.split(",")
        names.append(parts[2])

    # Закрываем файл и возвращаем список
    inf.close()
    return names

# Отображаем самую длинную последовательность, начинающуюся со введенного
# пользователем элемента
def main():
    # Загружаем имена элементов в список
    names = loadNames()

    # Считываем первый элемент и делаем первую букву заглавной
    start = input("Введите название элемента: ")
    start = start.capitalize()

    # Проверяем, что пользователь ввел правильный элемент
    if start in names:
        # Удаляем первый элемент из списка
        names.remove(start)

        # Находим самую длинную последовательность
        sequence = longestSequence(start, names)

        # Отображаем последовательность элементов
```

```
        print("Самая длинная последовательность, начинающаяся с", start, ", это:")
        for element in sequence:
            print(" ", element)
    else:
        print("Извините, вы ввели неправильный элемент.")

# Вызов основной функции
main()
```

## *Упражнение 184. Выравниваем список*

```
##
# Используя рекурсию, выравниваем список с вложенными элементами
#

## Удаляем все иерархии внутри списка
# @param data - список для выравнивания
# @return выровненная версия списка
def flatten(data):
    # Если список пустой, делать нечего
    if data == []:
        return []

    # Если первый элемент списка - список
    if type(data[0]) == list:
        # Выравниваем первый элемент и остальные
        return flatten(data[0]) + flatten(data[1:])
    else:
        # Первый элемент - не список, так что только остальные элементы нужно
        # выравнивать
        return [data[0]] + flatten(data[1:])

# Демонстрируем работу функции
def main():
    print(flatten([1, [2, 3], [4, [5, [6, 7]]], [[[8], 9], [10]]]))
    print(flatten([1, [2, [3, [4, [5, [6, [7, [8, [9, \
            [10]]]]]]]]]]))
    print(flatten([[[[[[[[[[1], 2], 3], 4], 5], 6], 7], 8], 9], \
            10]))
    print(flatten([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]))
    print(flatten([]))

# Вызов основной функции
main()
```

## *Упражнение 186. Кодирование на основе длин серий*

```
##
# Выполняем кодирование на основе длин серий для строк и списков
#
```

```
## Кодирование на основе длин серий для строк и списков
# @param data – строка или список для кодирования
# @return список, в котором на четных позициях располагаются значения,
# а на нечетных – количество раз, которое должно повториться предшествующее значение
def encode(data):
    # Если данных нет, нечего и кодировать
    if len(data) == 0:
        return []
```

Если бы мы задали условие `data == ""`, то эта функция работала бы только для строк. Если `data == []`, то только для списков. Проверка длины списка `len(data) == 0` позволяет обеспечить правильную работу функции как для строк, так и для списков.

```
    # Находим индекс первого элемента, отличного от элемента с индексом 0
    index = 1
    while index < len(data) and data[index] == data[index - 1]:
        index = index + 1

    # Кодируем группу символов
    current = [data[0], index]

    # Используем рекурсию для обработки символов от index до конца строки
    return current + encode(data[index : len(data)])

# Демонстрируем работу функции кодирования
def main():
    # Запрашиваем строку у пользователя
    s = input("Введите строку символов: ")

    # Отображаем результат
    print("При использовании кодирования на основе длин серий " \
        "результат будет следующим:", encode(s))

# Вызов основной функции
main()
```

# Предметный указатель

## Ф



## Ц

Книги издательства «ДМК ПРЕСС»
можно купить оптом и в розницу
в книготорговой компании «Галактика»
(представляет интересы издательств
«ДМК ПРЕСС», «СОЛОН ПРЕСС», «КТК Галактика»).
Адрес: г. Москва, пр. Андропова, 38;
тел.: **(499) 782-38-89**, электронная почта: **books@alians-kniga.ru**.
При оформлении заказа следует указать адрес (полностью),
по которому должны быть высланы книги;
фамилию, имя и отчество получателя.
Желательно также указать свой телефон и электронный адрес.
Эти книги вы можете заказать и в интернет-магазине: **www.a-planeta.ru**.

Бен Стивенсон

**Python. Сборник упражнений**

Главный редактор *Мовчан Д. А.*
Зам. главного редактора *Сенченкова Е. А.*
dmkpress@gmail.com
Перевод *Гинько А. Ю.*
Корректор *Синяева Г. И.*
Верстка *Чаннова А. А.*
Дизайн обложки *Мовчан А. Г.*

Гарнитура PT Serif. Печать цифровая.
Усл. печ. л. 17,4. Тираж 200 экз.

Веб-сайт издательства: **www.dmkpress.com**