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.