Министерство науки и высшего образования Российской Федерации
Ярославский государственный университет
им. П. Г. Демидова
Кафедра вычислительных и программных систем

К. В. Лагутина
А. М. Адрианова

Основы информатики на языке Python

Практикум

Ярославль
ЯрГУ
2023

УДК 004.43(076.5)
ББК 3973.2я73
Л14

Рекомендовано
Редакционно-издательским советом университета
в качестве учебного издания. План 2023 года

Рецензент
кафедра вычислительных и программных систем
Ярославского государственного университета им. П. Г. Демидова

Лагутина, Ксения Владимировна.
Л14 Основы информатики на языке Python : практикум / К. В. Лагутина, А. М. Адрианова ; Яросл. гос. ун-т им. П. Г. Демидова. — Ярославль : ЯрГУ, 2023. — 56 с.

Практикум содержит теорию, примеры программ и задачи по основным механизмам языка программирования Python.

Предназначен для студентов, изучающих дисциплину «Основы программирования».

УДК 004.43(076.5)
ББК 3973.2я73

Введение

Практикум создан для ознакомления студентов с основами языка программирования Python, а также получения практических навыков создания компьютерных программ. Помимо теоретической части, каждая тема содержит множество примеров и задач трёх уровней сложности, которые предназначены для отработки изученного материала. Пособие может использоваться как для самостоятельного изучения Python, так и в качестве дополнительного материала к лекционным занятиям.

В пособии освещается десять тем. В первой теме говорится об особенностях синтаксиса Python, основных типах данных, операторах и операциях языка. Вторая тема содержит информацию об операторах ветвления, а также затрагивает тему логических выражений. Третья тема посвящена управляющим конструкциям, а именно циклам. Здесь объясняется разница между циклами for и while, демонстрируется принцип работы операторов break и continue. В рамках четвёртой темы вводится понятие функции в программировании, подробно разбираются варианты работы с аргументами функции, а также с инструкцией return. Пятая тема содержит краткий обзор популярных библиотек Python и основные правила работы с библиотеками в коде. Следующие четыре темы посвящены основным коллекциям в Python: строкам, спискам, множествам и словарям. Помимо традиционных теоретической и практических тем, каждая из них содержит список методов, необходимых для работы с рассматриваемой коллекцией. В последней теме описывается работа с файлами через Python.

Синтаксис, переменные, операторы Python

Все конструкции языка программирования предназначены для организации процесса обработки данных. Взаимодействие синтаксических и семантических правил определяет основные понятия языка, такие как лексемы, операторы, идентификаторы, константы, переменные, типы данных, функции, процедуры и т. д. Язык программирования имеет ограниченный запас операторов и строгие правила их написания. Правила синтаксиса и семантики определены чётко и однозначно.

Программы на Python обычно пишутся в императивном стиле, каждая строка представляет собой команду для выполнения. Переменные объявляются без указания типа, который определяется во время их инициализации. Важную роль в синтаксисе играют отступы: строки, составляющие блок, должны иметь отступ на четыре пробела больше, чем предыдущая строка.

Основные типы данных в Python:

- *int* — целое число;
- $float$ — действительное число;
- *str* — строка, одинарные и двойные кавычки эквивалентны;
- *bool* — логический тип (True или False).

В Python имеются операторы присваивания, арифметические, логические и т. д. По приоритету они расположены следующим образом:

- {}, (), [] — скобки;

- . — ссылка на атрибут;
- ** — возведение в степень;
- *, /, //, % — умножение, деление, взятие остатка;
- +, - — сложение и вычитание;
- ==, !=, >, >=, <, <=, is, is not, in, not in — сравнение, проверка вхождения;
- not — логическое НЕ;
- and — логическое И;
- or — логическое ИЛИ.

Операция присваивания обозначается =. Для арифметических операций можно использовать сокращённую версию операции: a += b вместо a = a + b.

Пример № 1

В первом примере происходит инициализация двух переменных для роста и веса с помощью присваивания. Далее вычисляется индекс массы тела с помощью арифметических операций. Знак # обозначает комментарий к коду.

Функция *print* служит для вывода данных на консоль. Аргументами для неё могут быть любые типы данных, которые можно преобразовать в строку. Аргументы перечисляются через запятую.

```
m = 108.85 # Заводим переменную для массы тела
h = 1.85 # Заводим переменную для роста
imt = m / h**2 # Вычисляем индекс массы тела
# Возведение в степень - более приоритетная операция, чем деление
print('Индекс массы тела:', imt) # Выводим ответ
```

Пример № 2

Во втором примере с помощью функции *input* считываются строки из консоли. Функции *int* и *float* служат для преобразования строкового типа в числовые, целое и с плавающей точкой соответственно. *str* выполняет обратное действие.

```
print('Введите целое число:')
num = int(input()) # Преобразуем строку к целому числу
print('Введите дробное число')
# Преобразуем строку к числу с плавающей точкой
float_num = float(input())
summ = num + float_num
print('Сумма:',summ)
int_num = int(summ) # Преобразуем дробное число к целому
print('Без дробной части:', int_num)
mult = num * float_num
# Преобразуем число к строке и складываем строки
print('Произведение = ' + str(mult))
```

Пример № 3

В третьем примере вычисляется время падения с заданной высоты. Для того чтобы красиво вывести ответ, используется f-строка. Внутри неё в фигурных скобках можно указать переменные, значения которых нужно вставить в итоговую строку.

```
# Переменная для ускорения свободного падения
g = 9.81
print('Введите высоту как дробное число метров:')
# Преобразуем строку в число с плавающей точкой
h = float(input())
# Вычисление времени по формуле
# Чтобы сделать возведение в степень последним
# действием, ставим скобки
t = (2*h/g)**0.5
# Форматированный вывод значений h и t
print(f'Время падения с высоты {h} метров — {t} секунд')
```

Задачи уровня A

1. Вычислите, какое максимальное число ананасов можно купить на 500 руб. по 85 руб. за штуку.
2. Каждый день во время конференции расходуется 80 пакетиков чая. Конференция длится 4 дня. Чай продаётся в пачках по 100 пакетиков. Сколько пачек чая нужно купить на все дни конференции?
3. Железнодорожный билет для взрослого стоит 720 руб. Стоимость билета для школьника составляет половину от стоимости билета для взрослого. Группа состоит из 15 школьников и 2 взрослых. Выведите общую стоимость билетов на всю группу.

Задачи уровня B

1. Рассчитайте квартальную премию в размере 2/3 от оклада и выведите её значение и сумму, выдаваемую сотруднику «на руки» с учетом 13 % подоходного налога. Размер оклада введите с консоли.
2. Введите имена и размеры оклада для трёх сотрудников и выведите информацию о премиях в виде ведомости с общим итогом.
3. Пользователь вводит цифру n. Посчитайте n + nn + nnn.

Задачи уровня C

1. Для поездки на пикник три человека должны купить мясо по цене 285,5 руб./кг, хлеб по цене 25,4 руб./шт, огурцы по цене 40 руб./кг, помидоры по цене 51 руб./кг, а также потратить деньги на бензин для поездки (43,6 руб./литр). Введите количество покупаемых продуктов и бензина и рассчитайте, сколько рублей должен потратить каждый человек, если расходы делятся поровну.
2. Пусть первый человек купил мясо, второй — хлеб и овощи, а третий — везёт всех на своей машине. Рассчитайте, кто кому сколько денег должен вернуть.
3. Пользователь вводит три стороны треугольника. Рассчитайте периметр и площадь треугольника, радиусы вписанной и описанной окружностей.

Операторы ветвления

В языке Python представлены два вида оператора ветвления: условный оператор и оператор многозначного выбора.

Условный оператор применяется в том случае, если определённая часть программы должна быть выполнена тогда, когда верно некоторое условие. Полный синтаксис условного оператора выглядит таким образом:

```
if условие:
        блок кода № 1
elif условие:
        блок кода № 2

elif условие:
        блок кода № n–1
else:
        блок кода № n
```

После ключевого слова if следует условие – логическое выражение, результатом выполнения которого является одно из двух булевских значений: «истина» (True) или «ложь» (False). Если выражение в ходе работы программы принимает значение «истина», то блок кода № 1 выполняется и работа условного оператора завершается. В противном случае программа переходит к проверке последующих условий, стоящих после ключевого слова elif. Количество блоков elif в рамках одного условного оператора неограниченно, и условие каждого нового блока будет проверяться только в том случае, если ни один из предыдущих блоков не был выполнен. Если же ни одно из перечисленных условий не оказалось верным, то выполняется код, расположенный в блоке else.

Блоки elif и else не являются обязательными. Если они используются, то тогда условный оператор называется «неполным».

Блоки кода в каждой новой ветке условного оператора записываются с отступом в 4 пробела. Перед тем как перейти к примерам, уделим ещё немного времени условиям. Как и в других языках программирования, условия в Python могут быть простыми и составными. Простое условие представляет собой единственное логическое выражение, результатом которого является True или False. Составное условие – это два и более простых условия, соединённых с помощью логических операций «И», «ИЛИ», «НЕ». В Python эти операции обозначаются «and», «or» и «not» соответственно. Для вычисления значения составного условия необходимо опираться на таблицу истинности.

Пример № 1

В данном примере продемонстрирована работа условного оператора в рамках программы для определения чётности числа.

```
# считываем строку и преобразуем её к типу int
number = int(input())

# проверяем число на чётность:
# если число делится на 2 без остатка,
# значит, оно чётное
if number % 2 == 0:
    print("Число чётное")
else:
    print("Число нечётное")
```

Пример № 2

В этом примере представлена полная версия условного оператора в программе, предназначенной для определения оценки студента на основе количества набранных им баллов. Если студент набрал 25 и более баллов, то он получает оценку «отлично». Если количество баллов находится в промежутке от 20 до 25, то его оценка – «хорошо». Если студент набрал от 15 до 20 баллов, то он получает оценку «удовлетворительно», а если меньше 15, то оценку «плохо».

```
# считываем строку, введённую пользователем,
# и преобразуем её к типу int
```

```
score = int(input())

# проверяем условие
if score >= 25:
    # печатаем соответствующую оценку,
    # используя для её определения двойное неравенство
    print("отлично")
elif 20 <= score < 25:
    print("хорошо")
elif 15 <= score < 20:
    print("удовлетворительно")
else:
    print("плохо")
```

Пример № 3

Этот пример представляет собой программу, которая определяет, попадает ли число N в промежуток [1, 10].

```
# считываем строку и преобразуем её к типу int
N = int(input())

# с помощью составного условия реализуем нужную проверку
if (N >= 1) and (N <= 10):
    print("Число N попадает в промежуток [1, 10]")
else:
    print("Число N не попадает в промежуток [1, 10]")
```

В Python 3.10 был реализован ещё один оператор ветвления — конструкция match-case, которая реализует сопоставление значения некоторого условия с шаблоном. Ниже приведён полный синтаксис этой конструкции:

```
match условие:
        case шаблон № 1:
                блок кода № 1
        case шаблон № 2:
                блок кода № 2
    case шаблон № n:
                блок кода № n–1
```

```
    case _:
                блок кода № n
```

После ключевого слова match указывается условие — выражение или переменная, чьё значение впоследствии сопоставляется с одним из шаблонов. Шаблоном называется выражение или переменная, идущие после ключевого слова case. Количество шаблонов может быть любым. В случае совпадения условия с шаблоном выполняется соответствующий блок кода. Если же ни один из шаблонов не соответствует условию, то выполняется опциональный блок, шаблон для которого указывается как символ нижнего подчёркивания.

Пример № 4

В этой программе реализовано распределение дней недели по категориям «будний день» и «выходной».

```
# получаем введённое пользователем значение
day = input()

# сопоставляем полученное значение с шаблонами
match day:
    case "понедельник":
        print("будний день")
    case "вторник":
        print("будний день")
    case "среда":
        print("будний день")
    case "четверг":
        print("будний день")
    case "пятница":
        print("будний день")
    case "суббота":
        print("выходной")
    case "воскресенье":
        print("выходной")
    # указываем, что нужно делать,
    # если ни один из шаблонов не подошёл
    case _:
        print("этот день недели мы не знаем")
```

Задачи уровня A

1. Пользователь вводит название месяца. Напишите программу, которая выводит название сезона, к которому относится этот метод. Учтите возможные ошибки пользователя.

2. Пользователь вводит два числа. Напишите программу, которая выводит наибольшее из них. Учтите ситуацию, когда эти числа будут равны.

Задачи уровня B

1. Напишите программу, которая считывает с консоли букву и определяет, гласная она или согласная.

2. Вводятся три целых числа m, n и k. Выведите на консоль наибольшее и наименьшее из них.

Задачи уровня C

1. Пользователь вводит 2 числа и одну из пяти математических операций (+ - * / %). Составьте программу, которая бы вычисляла полученное выражение. Учтите возможные ошибки пользователя.

2. Пользователь вводит три числа. Напишите программу, которая проверяет их делимость на 3. В конце программа выводит те числа, которые делятся на 3, а если ни одно из них не делится на 3, то программа выводит сообщение «Ни одно из введённых чисел не делится на 3».

Операторы циклов

Ещё одной важной управляющей конструкцией в Python являются циклы. Эта конструкция используется в случае, когда необходимо многократно выполнить какой-либо блок кода.

В Python реализованы два вида циклов: цикл while и цикл for.

Цикл while – цикл с предусловием – выполняет указанные в теле цикла инструкции, пока условие, стоящее после ключевого слова while, истинно. Условием может быть любое логическое выражение.

Синтаксис цикла while выглядит следующим образом:

```
while условие:
        тело цикла
```

Пример № 1

На примере этой программы показано использование цикла while для вычисления чисел, введённых пользователем. Пользователь может ввести любое количество чисел. Сигналом остановки ввода является число 0.

```
# считываем число, введённое пользователем,
# и преобразуем его к типу int
number = int(input())
# создаём переменную, в которой будем хранить результат
summ = 0

# задаём цикл, в условии которого проверяем,
# равно ли значение переменной number 0 или нет
while number != 0:
    # добавляем введённое пользователем значение к сумме
```

```
    summ += number
    # считываем новое число
    number = int(input())

# выводим результат
print(summ)
```

Для управления работой цикла существуют операторы break и continue. Оператор break полностью прерывает выполнение цикла, а оператор continue позволяет перейти к следующей итерации, не завершая предыдущую.

Пример № 2

В этом примере продемонстрировано решение задачи с использованием бесконечного цикла и оператора break. Условие задачи: пользователь с консоли вводит числа. Как только пользователь введёт первое нечётное число, необходимо прервать цикл и вывести сообщение «Было введено нечётное число!»

```
# запускаем выполнение бесконечного цикла,
# используя вместо условия ключевое слово True,
# которое соответствует логической истине
# (таким образом мы обеспечиваем постоянную истинность
# условия на протяжении всего выполнения программы)
while True:
    # считываем и преобразуем введённое
    # пользователем число
    number = int(input())
    # проверяем число на чётность
    if number % 2 != 0:
        print("Было введено нечётное число!")
        # прерываем работу цикла с помощью
        # оператора break
        break
```

Пример № 3

Этот пример иллюстрирует работу оператора continue. Условие задачи: пользователь вводит с консоли 10 чисел. Необходимо просуммировать

каждое третье число, введённое пользователем.

```
# задаём переменную, которая будет хранить
# номер введённого числа
count = 0
# в переменной res будем хранить сумму
# каждого третьего введённого числа
res = 0

# запускаем цикл, который будет выполняться до тех пор,
# пока переменная count строго меньше 10
while count < 10:
    # считываем и преобразуем введённое пользователем число
    number = int(input())
    # проверяем число на чётность
    count += 1
    # добавляем число к итоговой сумме,
    # если его номер делится на 3
    if count % 3 == 0:
        res += number
    else:
        continue

# выводим итоговый результат
print(res)
```

Кроме того, у цикла while существует модификация, называемая while-else. Блок else является необязательным и выполняется только в том случае, если основная часть цикла завершилась в штатном режиме (то есть выполнение цикла не было прервано с использованием оператора break).

Пример № 4

В данном примере пользователь последовательно вводит числа. В случае если все вводимые пользователем числа строго меньше 10, то цикл выполняется 5 раз и в конце программа выводит сообщение: «Вы ввели 5 чисел, которые строго меньше 10». Если же пользователь вводит хоть одно число больше или равное 10, то выполнение программы сразу же прерывается.

```
# задаём переменную, которая будет
```

```
# хранить номер введённого числа
count = 0
# в переменной res будем хранить сумму
# каждого третьего введённого числа
res = 0

# запускаем цикл, который будет выполняться
# до тех пор, пока переменная count
# строго меньше 10
while count < 5:
    # считываем и преобразуем введённое пользователем число
    number = int(input())
    # в случае, если число больше или равно 10,
    # но прерываем выполнение цикла
    if number >= 10:
        break
    # увеличиваем переменную count,
    # чтобы не допустить зацикливания
    count += 1
else:
    print("Вы ввели 5 чисел, которые строго меньше 10")
```

Цикл for — цикл со счётчиком — представляет собой цикл, который выполняется, пока некоторая переменная — итератор цикла — изменяет своё значение в пределах какого-либо итерируемого объекта. Синтаксис цикла for выглядит следующим образом:

```
for итератор in итерируемый объект:
        тело цикла
```

Чаще всего в роли итерируемого объекта выступает диапазон, сгенерированный функцией range. Такой подход даёт программисту возможность изменять значение диапазона итератора цикла от заранее заданного начального значения до конечного значения с определённым шагом. На каждой итерации тело цикла выполняется строго один раз и значение итератора увеличивается на значение, равное шагу цикла.

Функция range получает на вход один, два или три аргумента и на их основе генерирует набор чисел. Если на вход функции range был подан только один аргумент, то функция принимает его за конец нужного диапазона и генерирует числа от 0 до переданного аргумента, не включая его. Если же на вход было передано два аргумента, то функция считает

эти аргументы как начало и конец диапазона и генерирует числа, начиная с первого аргумента и до второго, не включая его. В том случае, если функция range принимает три аргумента, то она распознаёт их как начало диапазона, его конец и шаг, с которым будут генерироваться числа.

Пример № 5

В этом примере продемонстрирован вывод всех натуральных чисел из диапазона от 0 до N. Значение N вводит пользователь.

```
# считываем введённое пользователем число
# и преобразуем его к типу int
N = int(input())

# распечатываем в цикле необходимые значения
for i in range(N + 1):
    print(i)
```

Как и цикл while, цикл for имеет необязательный блок else, который выполняется тогда, когда основная часть цикла завершила работу в штатном режиме.

Циклы while и for являются взаимозаменяемыми.

Пример № 6

В данном примере продемонстрировано сложение чисел из диапазона от 1 до 9 с помощью цикла while и цикла for.

```
# реализация с помощью цикла while
summ = 0
i = 1

while i < 10:
    summ += i
    i += 1

print(summ)


# реализация с помощью цикла for
```

```
summ = 0

for i in range(1, 10):
    summ += i

print(summ)
```

Циклы и условные операторы могут быть вложены друг в друга произвольным образом.

Задачи уровня A

1. Пользователь вводит строки одну за другой до тех пор, пока не введёт пустую. Программа должна выводить введенные строки, пока не встретилась пустая.
2. Пользователь вводит с клавиатуры целые числа N и M. Просуммируйте все числа из заданного диапазона и выведите результат на консоль. Предусмотрите ситуацию, когда могут быть введены некорректные N и M.

Задачи уровня B

1. Для введённого пользователем с клавиатуры натурального числа посчитайте сумму всех его цифр (заранее неизвестно сколько цифр будет в числе).
2. Пользователь вводит с клавиатуры числа a и b (a, b $>$ 0, целые, a $<$ b). Напишите программу, которая бы последовательно выводила все числа из диапазона от a до b, пока не встретит первое круглое число. Если такого числа не найдётся, то программа должна вывести все числа из диапазона.

Задачи уровня C

1. С помощью цикла while реализуйте возведение числа N в степень M. N, M вводит пользователь. N, M являются натуральными числами.
2. Выведите на экран все положительные делители натурального числа, введённого пользователем с клавиатуры.

Функции

Функцией называется блок кода, выполняющий какие-либо операции. Этот блок выносится отдельно от основного кода и определяется с помощью ключевого слова def. Ниже приведён синтаксис функции в Python.

```
def имя функции ( позиционные аргументы ,
                  именованные аргументы ):
        тело функции
```

Для вызова функции достаточно указать имя функции в коде и передать ей необходимые аргументы.

Пример № 1

Простейшая функция, суммирующая значения двух аргументов и выводящая результат суммирования на консоль.

```
# объявляем функцию
def my_sum(a, b):
    # суммируем переданные аргументы
    c = a + b
    # выводим результат на консоль
    print(c)

# вызываем функцию
my_sum(3, 5)
```

Для возвращения значения из функции используется ключевое слово return.

Пример № 2

Этот пример является минимальным усложнением примера № 1. Теперь значение, полученное в результате суммирования аргументов, не выводится на консоль, а возвращается в программу.

```
# объявляем функцию
def my_sum(a, b):
    # суммируем переданные аргументы
    c = a + b
    # возвращаем результат
    return c


# вызываем функцию
res = my_sum(3, 5)
```

Обратите внимание на появившуюся в этом примере переменную res. Она сохраняет значение, возвращённое функцией my_sum, для последующего использования его в программе. Отсутствие переменной, принимающей значение, не является ошибкой (в этом случае код тоже будет работать), но результат выполнения функции будет утерян.

Одна функция может возвращать сразу несколько значений. В таком случае результатом её работы будет являться особая неизменяемая коллекция кортеж. Возвращённый кортеж может быть сохранён в одну переменную, а может быть сразу же разобран в несколько переменных. Такое возможно благодаря особенностям языка Python.

Пример № 3

Рассмотрим функцию, которая принимает два аргумента, а возвращает их сумму и произведение.

```
# объявляем функцию
def sum_mul(a, b):
    # суммируем переданные аргументы
    c = a + b
    # вычисляем произведение переданных аргументов
    d = a * b
    # возвращаем результат
    return c, d
```

```
# раскладываем значения полученного кортежа в
# заранее подготовленные переменные
res_sum, res_mul = sum_mul(3, 5)
```

Функция может иметь сразу несколько точек выхода. Каждая из них обозначается собственной командой return.

Пример № 4

Этот пример демонстрирует работу функции с несколькими точками выхода. Эта функция получает на вход число и определяет её чётность.

```
# объявляем функцию
def odd_even(num):
    # проверяем переданное число на чётность
    # и в зависимости от этого возвращаем результат.
    # Обратите внимание, что в данном примере
    # в качестве условия для условного оператора
    # может быть передан либо 0, либо 1. В этом случае
    # 0 будет эквивалентен False, а 1 – True
    if num % 2:
        return "нечётное"
    else:
        return "чётное"

res = odd_even(3)
```

Принимаемых значений – аргументов – у функции может быть сколь угодно много, а может не быть вовсе. Аргументы функции делятся на две категории: позиционные (обязательные) и именованные (необязательные). В отличие от позиционных аргументов, именованным аргументам может быть заранее присвоено значение по умолчанию, которое будет использоваться в том случае, если при вызове функции именованному аргументу не будет присвоено другое значение. В сигнатуре функции все позиционные аргументы обязательно должны быть перечислены перед именованными. При указании значения именованных аргументов следует помнить синтаксический нюанс: в данном случае пробелы вокруг оператора присваивания ставить не нужно.

Пример № 5

Этот пример демонстрирует использование позиционных и именованных аргументов в рамках решения задачи по поиску периметра и площади прямоугольника. В аргументы функции передаются два обязательных аргумента – длины сторон прямоугольника, а также один необязательный – метод, регулирующий работу функции. Если значения метода равняется 1, то функция возвращает периметр, а если 2, то площадь.

```
# объявляем функцию
def rectangle(a, b, method=1):
    if method == 1:
        return 2 * (a + b)
    elif method == 2:
        return a * b


# вызываем функцию с разным количеством аргументов
res = rectangle(3, 4)
res = rectangle(3, 4, method=2)
```

Если заранее неизвестно, сколько аргументов будет передано функции, то в таком случае можно воспользоваться механизмом Python, который позволяет принимать переменное количество аргументов. Для реализации этой возможности надо поставить перед именем аргумента одну звёздочку (*) в том случае, если неизвестно точное число передаваемых позиционных аргументов, и две звёздочки (**), если планируется работа с именованными аргументами. Переданные таким образом позиционные аргументы воспринимаются программой как кортеж, а именованные — как словарь. Как следствие, с полученными коллекциями можно работать так же, как и с обычным кортежем или словарём.

Пример № 6

Данный пример показывает решение задачи по суммированию некоторого заранее неизвестного количества чисел.

```
# объявляем функцию
def my_sum(*numbers):
    # инициализируем переменную, где потом будем
    # хранить результат сложения
```

```
    res = 0
    # с помощью цикла перебираем все элементы полученного
    # кортежа numbers и суммируем их
    for number in numbers:
        res += number
    # возвращаем результат
    return res


# вызываем функцию с разным количеством аргументов
res = my_sum()
res = my_sum(3)
res = my_sum(3, 4, 5)
```

Задачи уровня A

1. Напишите функцию, которая принимает на вход два числа, а возвращает их сумму, разность, произведение и частное. Предусмотрите ситуации, когда какое-либо из указанных действий выполнить невозможно.
2. На вход функции подаётся три числа – стороны треугольника. Допишите функцию так, чтобы она вычисляла площадь треугольника.
3. Напишите функцию, которая вычисляет факториал заданного числа. Предусмотрите ситуацию, когда передано отрицательное число.

Задачи уровня B

1. Напишите функцию, которая по трём переданным сторонам треугольника определяет его тип (остроугольный, прямоугольный или тупоугольный).
2. Напишите функцию, которая принимает на вход натуральное число, а возвращает все простые числа из диапазона от 2 до переданного числа (включая его).
3. Напишите функцию, которая находит решение линейного уравнения ax = b. Предусмотрите случаи, когда у уравнения решений нет и когда решений бесконечно много.

Задачи уровня C

1. Напишите функцию, которая получает на вход некоторое количество целых чисел, а возвращает сумму тех из них, которые делятся на 7.

2. Функция получает на вход кортеж целых чисел, а также значение для необязательного аргумента method, по умолчанию равного 1. Допишите функцию так, чтобы в зависимости от значения необязательного аргумента функция возвращала бы либо произведение всех чётных чисел из ранее переданного кортежа, либо сумму всех нечётных. Аргумент method регулирует работу функции следующим образом: если его значение является нечётным числом, то функция должна вернуть сумму нечётных чисел, а если чётным, то произведение чётных чисел.

3. Напишите функцию, которая находит решение квадратного уравнения $ax^2 + bx + c = 0$. Укажите аргументу a значение по умолчанию 1, аргументу b — 2 и аргументу c — 1. Предусмотрите случаи, когда у уравнения решений нет и когда оно является линейным.

Модули

Модуль/пакет/библиотека — это совокупность программных ресурсов, предназначенных для использования другими программами. Он содержит атрибуты: классы, функции, константы, переменные, подмодули.

Для того чтобы импортировать себе в программу сторонний модуль, нужно воспользоваться оператором *import*, например:

```
import math
import pandas as pd
from numpy import array
```

В первом случае модуль *math* импортируется по собственному имени, и к его атрибутам можно обращаться через это имя: *math.sin*.

as позволяет переименовать модуль и обращаться к его атрибутам по псевдониму: *pd.Dataframe*, а конструкция *from ... import ...* импортирует конкретные атрибуты по их именам, к которым впоследствии можно обращаться без префиксов.

Пример № 1

В первом примере вычисляется наибольший общий делитель с помощью функции *gcd* из модуля *math*.

```
import math
# вводим исходные данные
a = int(input("Введите первое число: "))
b = int(input("Введите второе число: "))
# Считаем НОД с помощью функции из модуля math
print("НОД: ", math.gcd(a, b))
```

Пример № 2

Во втором примере из модуля *datetime* импортируются конкретный атрибут *datetime*. С его помощью парсятся даты и время отправления и прибытия поезда по заданному формату, а также вычисляется время в пути.

```
from datetime import datetime
print('Введите дату отправления поезда')
start_date = datetime.strptime(input(), "%d/%m/%y %H:%M")
print('Введите дату прибытия поезда')
finish_date = datetime.strptime(input(), "%d/%m/%y %H:%M")
# Вычисляем время в пути
travel_time = finish_date - start_date
# Проверяем, задана ли была дата прибытия правильно
if travel_time.total_seconds() < 0:
    print('Поезд прибывает раньше, чем отправляется')
else:
    print('Время в пути:', travel_time)
```

Пример № 3

В третьем примере генерируется случайный идентификатор из букв и цифр. Для выбора случайного элемента используются функции из модуля *random*.

```
# Импортируем модуль с атрибутами для случайного выбора
import random
# Функция генерирует случайное число от min_num
# до max_num включительно.
# Генерация происходит с помощью random.randint.
# Если диапазон задан неверно, возвращается min_num
def generate_random_int_number(min_num, max_num):
    if min_num < max_num:
        return random.randint(min_num, max_num)
    else:
        return min_num
# Функция генерирует случайную строку длины str_len.
# Если str_len не задано, оно будет 5 по умолчанию.
# Строка генерируется из заглавных и строчных
# латинских букв.
# Выбор случайной буквы из строки происходит
```

```
# с помощью random.choice
def generate_random_letters(str_len=5):
    letters = 'abcdefghijklmnopqrstuvwxyz' \
              'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
    random_str = ''
    for _ in range(str_len):
        random_str += random.choice(letters)
    return random_str
# Функция генерирует случайный идентификатор,
# вызывая предыдущие функции.
# Идентификатор будет состоять из 2 букв, 3 цифр
#  и 5 букв последовательно
def get_id():
    num = generate_random_int_number(100, 999)
    return generate_random_letters(str_len=2) \
        + str(num) + generate_random_letters()
# Инициализируем генератор случайного выбора
random.seed(42)
# Генерируем и печатаем идентификатор
print(get_id())
```

Задачи уровня A

1. Вводится угол в градусах. Нужно перевести его в радианы и определить синус, косинус и тангенс угла.

2. Вводится день рождения пользователя. Выведите, какого числа ему исполнится 10 000 дней, какого числа — 1 000 000 минут, какого числа — 1 000 000 000 секунд.

3. Сгенерируйте строку из русских заглавных и строчных букв длиной в 100 символов.

4. Вычислите факториал числа, заданного пользователем, с помощью функции из модуля math.

Задачи уровня B

1. Имеется несколько вариантов ввода дат:

2005-08

20050809 18:31:12

2005-08-09 18:31

09.08.2005 18:31:42

2005-08-09T18:31:42

Для каждого варианта ввода даты напишите строку кода с *strptime* для парсинга.

2. Задано время отправления поезда и время в пути до конечной станции. Требуется написать программу, которая найдет время прибытия этого поезда (возможно, в другие сутки). Время в пути — это два числа, часы и минуты.

3. Сгенерируйте 100 случайных чисел с плавающей точкой в диапазоне, заданном пользователем. Для этого воспользуйтесь, например, функцией $random.uniform()$. Выведите каждое из чисел, огруглив его до одного знака после запятой.

4. Создайте программу, которая выводит количество дней, часов и минут до даты, заданной пользователем. Если это дата из прошлого, программа должна сообщить, сколько дней и часов уже прошло.

Задачи уровня C

1. Имеется кучка из N камней. N вычисляется случайно как число от 4 до 30. Каждый игрок по очереди берёт 1, 2 или 3 камня. Выигрывает тот, кто оставляет последний камень сопернику. Запрограммируйте игру двух пользователей между собой. Для вычисления случайного целого числа можно использовать функцию *randint* из модуля *random*.

2. Найдите приблизительное значение $cos(x)$ с помощью ряда Маклорена. Запросите у пользователя точку x и n — количество элементов ряда. Вычислите все n слагаемых ряда Маклорена с помощью функций модуля *math*.

Строки

Для удобного представления текстовой информации в Python используются строки, которые представляют собой неизменяемую коллекцию проиндексированных упорядоченных элементов.

Для создания строки необходимо обернуть нужное значение в одинарные или двойные кавычки. Принципиальной разницы между разными типами кавычек нет — в нижеприведённом примере значения переменных string1 и string2 эквивалентны между собой.

```
string1 = "строка"
string2 = 'строка'
```

Возможность использования различных кавычек при создании строковых литералов позволяет использовать кавычки внутри строки как обыкновенные кавычки, а не как особый символ для обозначения строкового типа данных.

```
string = "Картина ' Девушка с жемчужной серёжкой '"
```

К каждому отдельному элементу строки можно получить доступ по индексу. Для этого необходимо обратиться к переменной и после этого указать нужный индекс в квадратных скобках. Как и в любой другой коллекции, индексация элементов строк начинается с 0.

Пример № 1

Эта программа считывает строку string и натуральное число N, а затем выводит на экран элемент строки string, расположенный по индексу N.

```
# считываем строку
string = input()
```

```
# считываем целое число - индекс
N = int(input())
# выводим элемент строки по индексу
print(string[N])
```

Одной из наиболее полезных операций над строками является операция взятия среза. Она позволяет получить некоторую подстроку изначальной строки. Общий синтаксис операции среза выглядит следующим образом:

имя_строки[начало_среза:конец_среза:шаг]

Операция среза может получить на вход 0, 1, 2 или 3 аргумента. Если операция среза была вызвана только с одним аргументом n, то результатом выполнения операции будет подстрока, взятая с 0 по n-1 элемент. Если же на вход были поданы два аргумента n и m, то в таком случае будет возвращена подстрока с n по m-1 элемент. А в случае, если операции были поданы все три аргумента n, m и k, то тогда будет возвращена подстрока, полученная на основе исходной в диапазоне с n по m-1 элемент с шагом k. Случай, когда операции среза не было подано вообще никаких аргументов является простейшим. При таких обстоятельствах данная операция никак не преобразует исходную строку и та вернётся в изначальном виде. В качестве аргументов операции среза могут выступать как положительные, так и отрицательные числа.

Пример № 2

Данный пример наглядно иллюстрирует поведение операции среза при различных аргументах. В комментариях после каждой команды print() отражён результат работы команды и пояснение.

```
# считываем строку изучаем_Python
string = input()

print(string[::])    # изучаем_Python
print(string[:3])    # изу
print(string[:1000])    # изучаем_Python.
print(string[2:5])    # уча
print(string[1:6:2])    # зче
print(string[0:6:2])    # иуа
```

```
print(string[-6:-1])   # Pytho
print(string[::-1])    # nohtyP_ меачузи
print(string[1:6:-1])  # В данном случае была
                       возвращена пустая строка,
                       так началом среза
                       является элемент с индексом 1,
                       который при обратном обходе
                       будет меньше элемента с индексом 6,
                       следовательно, обход будет невозможен.
```

К простейшим операциям со строками относится дублирование и конкатенация строк.

Дублирование позволяет создать на основе исходной строки новую, которая будет состоять из n-кратного повтора исходной строки. Операция дублирования в Python обозначается символом звёздочки.

Операция конкатенации буквально склеивает две строки в одну. В Python она обозначается символом плюса.

Пример № 3

Этот пример показывает работу операций дублирования и конкатенации.

```
# считываем целое число
n = int(input())
# считываем первую строку
string1 = input()
# дублируем строку n раз
print(string1 * n)

# считываем вторую строку
string2 = input()
# конкатенируем первую строку со второй
print(string1 + string2)
```

Строки, как и другие коллекции в Python, располагают широким набором встроенных функций и методов. Ниже перечислены наиболее важные из них.

- len(string) – возвращает количество элементов в строке string;
- x in string – проверяет, содержится ли элемент x в строке string или нет;

- string.find(s, start, end) – находит подстроку s в строке string в диапазоне от start до end-1. Если искомая подстрока найдена, то метод вернёт номер первого вхождения подстроки в строку, иначе -1. Аргументы start и end являются необязательными и при их отсутствии поиск будет происходить в пределах всей строки;

- string.replace(s1, s2, count) – заменяет в строке string подстроку s1 на подстроку s2 столько раз, сколько указано в аргументе count. Аргумент count является необязательным, и при его отсутствии будет заменено каждое вхождение подстроки s1.

- string.count(s, start, end) – возвращает количество непересекающихся вхождений подстроки s в диапазоне от start до end. Аргументы start и end являются необязательными и по умолчанию равняются 0 и длине строки;

- string.split(x) — превращает строку string в список подстрок, разделённых по выбранному разделителю x;

- x.join(lst) — соединяет строки из списка lst в одну с помощью выбранного соединителя x;

- string.strip(s) — убирает символы s с обеих сторон строки string;

- string.startswith(s) — проверяет, начинается ли строка string с подстроки s;

- string.endswith(s) — проверяет, заканчивается ли строка string подстрокой s;

- string.islower– проверяет, состоит ли строка из символов в нижнем регистре;

- string.isupper – проверяет, состоит ли строка из символов в верхнем регистре;

- string.lower() — перевод всех символов строки string в нижний регистр;

- string.upper() — перевод всех символов строки string в верхний регистр;

- string.capitalize() — перевод первой буквы строки string в верхний регистр, остальных — в нижний.

Пример № 4

Пользователь вводит 5 слов, разделённых запятой и пробелом. Программа выводит на консоль самое короткое и самое длинное из них.

```
# считываем слова и разделяем их
# по заданному разделителю
words = input().split(",␣")

# задаём начальные значения для самого
# короткого и самого длинного слова
shortest = words[0]
longest = words[0]

# находим самое короткое и самое длинное слово
for word in words:
    if len(word) < len(shortest):
        shortest = word
    if len(word) > len(longest):
        longest = word

# выводим результат
print(shortest)
print(longest)
```

Пример № 5

Пользователь последовательно вводит две строки. Программа определяет, является ли первая строка подстрокой второй, учитывая при этом регистр обеих строк.

```
# считываем введённые строки
# и приводим их к одному регистру
string1 = input().lower()
string2 = input().lower()

# проверяем, является ли вторая строка
# подстрокой первой
if string1.find(string2) != -1:
    print("является")
```

```
else:
    print("не является")
```

Пример № 6

Пользователь последовательно вводит две строки. Программа удаляет из первой строки первые два вхождения второй подстроки и выводит на консоль получившийся результат.

```
# считываем введённые строки
string1 = input()
string2 = input()

# производим удаление подстроки
# путём замены её на пустую строку
res = string1.replace(string2, "", 2)

# выводим результат
print(res)
```

Задачи уровня A

1. Считайте строку и выведите все её элементы в формате «номер элемента в строке: элемент».

2. Напишите программу, которая считывает строку и выводит все её элементы, имеющие

   - чётные индексы
   - нечётные индексы

Задачи уровня B

1. Пользователь вводит 10 слов, разделённых точкой с запятой. Выведите на консоль все слова, количество букв в которых превышает значение среднего арифметического для количества букв всех введённых слов.

2. Пользователь вводит 10 слов, разделённых точкой с запятой. Определите, сколько из них начинаются с большой буквы и заканчиваются восклицательным знаком, вопросительным знаком или точкой.

Задачи уровня C

1. Пользователь вводит 10 слов, разделённых точкой с запятой, а затем контрольную строку на отдельной строке. Выведите только те слова, которые начинаются с контрольной строки.

2. Введите строку из консоли. Удалите лишние пробелы и табуляции из начала и конца строки. Приведите строку к формату, когда только первое слово начинается с заглавной буквы, остальной текст находится в нижнем регистре, а строка заканчивается одной точкой без многоточия, восклицательных и вопросительных знаков.

Списки

Для типа данных *list* в Python можно использовать и термин массив, и термин список, так как list обладает атрибутами и массивов, и списков.

Список — это упорядоченная коллекция элементов произвольных типов. Внутри одного списка могут быть элементы нескольких типов, но обычно в него помещают однородные элементы.

Основные функции списков:

- arr.append(el) — добавляет элемент el в конец списка;
- arr.insert(i, el) — вставляет на i-ю по индексу позицию значение el;
- arr.remove(el) — удаляет первый элемент в списке, имеющий значение el;
- arr.pop(i) — удаляет элемент с индексом i и возвращает его;
- arr.sort() — сортирует список;
- arr.reverse() — разворачивает список;
- arr.clear() — очищает список.

Все эти функции изменяют исходный список.

Пример № 1

Пустой список создаётся с помощью квадратных скобок или функции *list*. С помощью функции *append* в конец списка можно поместить новые элементы.

Встроенные функции позволяют выполнить базовые операции со списками: *len* — найти количество элементов списка, *sum* — их сумму, *min* и *max* — минимальный и максимальный элементы соответственно.

В следующем примере требуется заполнить массив 20 случайными целыми числами (диапазон чисел от 0 до 1000) и найти: сумму минимального и максимального элементов

```
import random
nums = [] # Пустой массив
# Заполняем массив 20 случайными числами
for i in range(20):
    nums.append(random.randint(0, 1000))
# Выводим получившийся массив
print(nums)
# Ищем минимум и максимум с помощью встроенных функций
print(min(nums) + max(nums))
```

Пример № 2

Элементы списка удобно перебирать с помощью *for . . . in*. К конкретным элементам списка можно обращаться по их индексам с помощью квадратных скобок.

```
# Пользователь задает количество элементов массива и вводит их.
# Найдите количество отрицательных элементов,
# стоящих на чётных по индексу местах.
arr_len = int(input('Введите количество элементов массива: '))
if arr_len <=0: # Проверяем ввод
    print('Количество элементов должно быть положительным')
    exit()
arr = [] # Заводим новый массив
# Считываем столько элементов массива, сколько задал пользователь
for i in range(arr_len):
    arr.append(int(input(f'arr[{i}] = ')))
# Счётчик отрицательных элементов,
# стоящих на чётных по индексу местах
counter = 0
# Перебираем элементы на чётных по индексу местах
for i in range(0, len(arr), 2):
```

```
    if arr[i] < 0: # Подсчитываем отрицательные элементы
        counter += 1
print(counter) # Выводим ответ
```

Пример № 3

Помимо индексов, для списков определён оператор извлечения среза. Он имеет синтаксис: $[X : Y]$, где X — начало среза, Y — окончание +1.

```
arr = [2, 3, 8, 0, 15, 2]
print(arr[1:4]) # [3, 8, 0]
print(arr[2:-2]) # [8, 0]
# По умолчанию в срезе первый индекс равен 0,
# а второй — длине списка.
print(arr[1:]) # [3, 8, 0, 15, 2]
print(arr[:3]) # [2, 3, 8]
# Можно задать шаг, с которым нужно извлекать срез
print(arr[::-1]) # [2, 15, 0, 8, 3, 2]
print(arr[2::2]) # [8, 15]
```

Пример № 4

Функция *split* преобразует строку в список строк, используя пробел в качестве разделителя по умолчанию.

```
# Считываем строку и разбиваем на слова по пробелам
words = input('Введите слова через пробел:').split()
counter = 0 # Счётчик палиндромов
for word in words: # Перебираем все слова
    # Если слово совпадает со своей перевёрнутой версией,
    # это палиндром
    if word == word[::-1]:
        counter += 1
print(counter) # Выводим количество палиндромов
```

Элементы списка можно перебирать с помощью $for \ldots in$ не только через индексы, но и через непосредственное обращение к самим элементам, как в предыдущем примере.

Задачи уровня A

1. Пользователь задаёт количество элементов массива и вводит их. Найдите количество чётных положительных элементов массива.
2. Считайте строку, разбейте её на слова по пробелам. Подсчитайте, сколько слов имеют длину, заданную пользователем.
3. Считайте строку, разбейте её на слова по пробелам. Подсчитайте, сколько слов содержат подстроку, заданную пользователем.
4. Даны два вектора (массива). Найдите их скалярное произведение.

Задачи уровня B

1. Заполните массив 20 целыми числами. Образуйте новый массив, элементами которого будут элементы исходного массива, оканчивающиеся на цифру 3, отсортированные по убыванию.
2. Заданы две матрицы m × k и k × n. Найдите их произведение.
3. Считайте строку, разбейте её на слова по пробелам. Подсчитайте, сколько слов начинаются с заглавной буквы и сколько слов заканчиваются знаком препинания из списка .,:;!?.
4. Удалите все элементы списка, равные заданному.

Задачи уровня C

1. В массиве действительных чисел есть нулевые элементы. Найдите длину максимальной последовательности подряд идущих нулей и индекс первого элемента этой последовательности.
2. Модой массива называется элемент, который встречается в массиве наиболее часто. Если в массиве имеется несколько наиболее часто встречающихся элементов и число их вхождений совпадает, то считается, что массив не имеет моды. Найдите моду данного массива, если его элементы — целые числа.
3. Пользователь вводит строку из слов, разделённых пробелами. Выведите на экран два слова: наиболее часто встречающееся и самое длинное.

Множества

В рамках Python множества представляют собой ещё одну коллекцию элементов, которая имеет несколько важных особенностей. Во-первых, каждый элемент множества должен быть уникален – коллекция не может содержать дубликатов. Во-вторых, элементы множества неупорядочены, и, как следствие, множество не предполагает наличие таких операций, как обращение к элементу по индексу или взятие среза. В-третьих, множество может содержать в себе только иммутабельные элементы (хотя при этом само множество является изменяемой коллекцией). Создать множество можно двумя способами. Во-первых, можно расположить необходимые элементы внутри фигурных скобок. Например:

```
# создание множества на основе списка чисел
number_set = {1, 2, 3}
# создание множества на основе строки
string_set = {'abc'}
# создание множества на основе числа и строки
num_string_set = {'a', 1}
```

Во-вторых, можно создать множество на основе любого другого итерируемого объекта. Для этого необходимо использовать функцию set(), которая преобразует переданный ей в качестве аргумента итерируемый объект во множество. Тогда создание аналогичных множеств будет выглядеть следующим образом:

```
# создание множества на основе списка чисел
number_set = set([1, 2, 3])
# создание множества на основе строки
string_set = set('abc')
# создание множества на основе числа и строки
num_string_set = set('a', 1)
```

При создании множества важно помнить, что множество хранит только уникальные объекты, поэтому если в исходном итерируемом объекте встречаются повторяющиеся элементы, то они будут удалены во время преобразования объекта ко множеству.

Пытаясь создать пустое множество, программист должен учитывать нюанс: создать пустое множество возможно лишь с помощью функции set(), так как оператор уже зарезервирован в Python для создания словаря.

Для множества в Python предусмотрены различные встроенные функции и методы. Рассмотрим некоторые из них.

- len(string) – возвращает количество элементов в строке string;
- x in string – проверяет, содержится ли элемент x в строке string или нет;
- st.add(x) – добавляет во множество st элемент x;
- st.discard(x) – удаляет элемент x из множества st. Метод не выдаст ошибку, если искомого элемента не оказалось во множестве;
- st.remove(x) – удаляет элемент x из множества st. В случае отсутствия искомого элемента во множестве, метод выдаст ошибку KeyError;
- st.pop() – удаляет и возвращает случайный элемент из множества;
- st.clear() – полностью очищает множество.

Пример № 1

Рассмотрим программу, которая решает следующую задачу. Пользователь вводит 10 чисел. Каждое новое число вводится на отдельной строке. Затем вводится контрольное число. Необходимо написать программу, которая определяет, содержится ли контрольное число в введённом ранее наборе, а также подсчитывает количество уникальных в нём элементов.

```
# создаём пустое множество
numbers = set()

# запускаем цикл для считывания набора чисел
for i in range(10):
```

```
    # считываем число
    num = int(input())
    # добавляем число во множество
    numbers.add(num)

# считываем контрольное число
control = int(input())

# проверяем, содержится ли контрольное число в наборе
# вернётся (значение типа bool)
print(control in numbers)
# определяем количество элементов во множестве
# это (число и будет равно количеству уникальных
# элементов в исходном наборе)
print(len(numbers))
```

Python позволяет работать со множествами, как и с аналогичными структурами в математике: производить операции объединения, пересечения, разности и симметричной разности.

Выполнение операции объединения позволяет создать на основе двух множеств новое множество, содержащее в себе все элементы, встречающиеся в первых двух множествах. Для реализации этой операции следует использовать метод union() или символ |.

Операция пересечения, применённая к двум множествам, возвращает новое множество, которое состоит из элементов, имеющихся в обоих исходных множествах. Эта операция выполняется либо с помощью метода intersection(), либо с помощью символа &.

При использовании на двух других множествах операция разности возвращает новое множество, состоящее из элементов, которые находились только в первом множестве, но не во втором и не входили в их пересечение. Операция получения разности выполняется с помощью метода difference() или с помощью символа -.

Результатом операции симметричной разности является множество, содержащее в себе элементы, находящиеся либо в первом, либо во втором исходном множестве, но не в их пересечении. Эта операция реализуется либо с использованием метода symmetric_difference(), либо с помощью символа ^.

Пример № 2

Этот пример демонстрирует работу вышеописанных операций над двумя множествами.

```
# создаём исходные множества
set_a = {1, 2, 3}
set_b = {2, 3, 4}

# выводим объединение множеств,
# полученное двумя способами
print(set_a.union(set_b))  # {1, 2, 3, 4}
print(set_a | set_b)  # {1, 2, 3, 4}

# выводим пересечение множеств,
# полученное двумя способами
print(set_a.intersection(set_b))  # {2, 3}
print(set_a & set_b)  # {2, 3}

# выводим разность множеств,
# полученную двумя способами
print(set_a.difference(set_b))  # {1}
print(set_a - set_b)  # {1}

# выводим симметричную разность множеств,
# полученную двумя способами
print(set_a.symmetric_difference(set_b))  # {1, 4}
print(set_a ^ set_b)  # {1, 4}
```

Задачи уровня A

1. Пользователь вводит набор чисел. Напишите программу, которая определяет, содержит ли введённый набор повторяющиеся элементы или нет.

2. Пользователь вводит 5 строк. Выведите на консоль строку, которая содержит наибольшее количество уникальных элементов.

Задачи уровня B

1. Пользователь вводит 10 строк. Напишите программу, которая определяет, сколько среди введённых строк дубликатов.

2. Пользователь вводит 2 набора по 10 чисел. Каждый набор вводится на отдельной строке, числа в наборе разделены пробелом. Напишите программу, которая выводит на консоль те элементы, которые

   - присутствуют в каждом из наборов;
   - присутствуют только во втором наборе.

Задачи уровня C

1. Пользователь вводит 4 набора по 5 чисел. Каждый набор вводится на отдельной строке, числа в наборе разделены пробелом. Напишите программу, которая выводит на консоль числа, которые не повторяются в других наборах.

2. Пользователь вводит 3 набора по 5 чисел. Каждый набор вводится на отдельной строке, числа в наборе разделены пробелом. Выведите на консоль элементы, которые встречаются в каждом из наборов.

3. Не используя операцию симметричной разности, напишите программу, которая бы выводила те элементы двух исходных множеств, которые не встречаются в их пересечении, но встречаются в каждом из них в отдельности.

Словари

Словарь в Python представляет собой неупорядоченную коллекцию элементов, организованных по принципу «ключ: значение». Эта коллекция представляет собой реализацию структуры данных, более известной как ассоциативный массив. Чтобы создать словарь, можно воспользоваться либо функцией dict() и передать в неё итерируемый объект, либо, как и в случае со множеством, оператором {}. Тонкость, которую следует учитывать, заключается в том, что и итерируемый объект, передаваемый в функцию, и операнды, подаваемые оператору, должны быть в определённом формате. Например:

```
num_dict = dict([(1, 1), (2, 4)])


num_dict ={
          1: 1,
          2: 2,
}
```

Существует ещё одна важная особенность при работе со словарями: ключи в словаре должны быть уникальны. Если попытаться создать словарь, содержащий в себе повторяющиеся ключи, то в итоге будет сохранено лишь значение, присвоенное последнему дубликату.

```
num_dict = {
     1: 1,
     2: 2,
     1: 4
}

print(num_dict)  # {1: 4, 2: 2}
```

Для получения значения из словаря необходимо после имени словаря указать в квадратных скобках нужный ключ. Если же обратиться к несуществующему ключу с целью получения значения, то будет возвращена ошибка KeyError.

```
num_dict = {
    10: "десять",
    100: "сто",
    1000: "тысяча"
}

print(num_dict[10])   # десять
print(num_dict[1])   # KeyError
```

Чтобы изменить значение, хранящееся в словаре, нужно обратиться к словарю по соответствующему ключу и присвоить ему новое значение. Если же указанный ключ отсутствует в словаре, то при попытке присвоить этому ключу какое-либо значение пара будет просто добавлена в словарь. Таким образом реализуется добавление новых элементов в словарь.

Словарям доступен набор встроенных функций и методов. Ниже приведены наиболее полезные из них.

- dict.clear() — удаляет все элементы из словаря;
- dict.get(key, default) — возвращает значение по ключу key. Если же такого ключа нет, то метод возвращает значение default (по умолчанию None);
- dict.items() — возвращает пары (ключ, значение);
- dict.keys() — возвращает список ключей в словаре;
- dict.values() — возвращает список значений в словаре;
- dict.update(other) – расширяет исходный словарь dict, добавляя пары «ключ: значение» из словаря other.

Пример № 1

Рассмотрим программу, которая считывает с консоли некоторое значение, а затем проверяет, является ли это значение ключом в словаре или нет.

```
# создаём словарь
num_dict = {
    "winter": "зима",
    "spring": "весна",
    "summer": "лето",
    "fall": "осень"
}

# считываем слово, введённое пользователем
key_word = input()

# проверяем, является ли введённое слово
# ключом в словаре
print(key_word in num_dict.keys())
```

Задачи уровня A

1. Пользователь вводит две строки. В первой через пробел перечислены музыкальные исполнители, а во второй в таком же формате их произведения. Количество произведений равно количеству исполнителей. На основе этих данных создайте словарь, в котором ключом будет имя исполнителя, а значением – его произведение. Выведите содержимое словаря на консоль в формате «исполнитель — произведение».

2. Пользователь вводит 5 пар «Год рождения: Имя Фамилия» (без кавычек). На основе этих данных создайте словарь, а затем выведите на консоль имя и фамилию самого молодого человека. Гарантируется, что год рождения в каждой записи уникален.

Задачи уровня B

1. В словаре хранится список участников конкурса и их баллов за конкурс: Иванов 20, Сидоров 68, Петров 26, Смирнов 68. Дополните словарь своими данными. Выведите на консоль список участников, балл которых выше среднего. Выведите на экран минимальный, максимальный и средний баллы. Минимальный и максимальный

баллы выведите вместе с фамилиями всех участников, которые их получили.

2. Таблица ASCII представляет каждую букву английского алфавита в виде уникального двоичного кода. Взяв за основу это сопоставление, напишите программу, которая считывает слово, а затем выводит на консоль это же слово, но зашифрованное с помощью таблицы.

Задачи уровня C

1. Создайте русско-английский словарь. Введите строку текста и выполните пословный перевод. В случае если нужное слово отсутствует в словаре, выведите None вместо его перевода. Постарайтесь учесть пунктуацию исходной строки.

2. Создайте словарь книг и их авторов. У одного автора может быть несколько книг, имя автора должно быть ключом. Организуйте работу с пользователем: если он вводит фамилию автора, то выведите на экран все книги этого автора. Кроме этого, разработайте интерфейс, позволяющий пользователю добавлять в словарь новую информацию. Предусмотрите ситуацию, когда данные, введённые пользователем, будут дублировать уже имеющиеся данные в словаре.

Работа с файлами

Python позволяет работать с простыми текстовыми файлами. Файлы можно открывать, читать, перезаписывать, дозаписывать и закрывать. В конце работы файл обязательно нужно закрыть, чтобы корректно завершить запись данных в файл и освободить файловый ресурс.

Для успешной работы программы в разных ОС и файловых системах рекомендуется не прописывать путь к файлу в виде строки текста, так как в разных ОС используются разные сепараторы. Вместо этого нужно использовать средства из системных библиотек языка программирования для указания пути к файлам и указывать относительные пути вместо абсолютных.

Синтаксическая конструкция $with open(\ldots) as \ldots$: позволяет открыть файл в начале блока кода, выполнить с ним действия в блоке, а в конце она позаботится о том, чтобы файл был закрыт.

Пример № 1

Для первого примера кода имеется файл с названием *dataIMT.txt*, расположенный в каталоге *data*. Каталог *data* находится рядом с файлом кода. Фрагмент *dataIMT.txt*:

```
Male
73.847017017515
241.893563180437
Male
68.7819040458903
162.310472521300
Male
74.1101053917849
```

```
212.7408555565
```

Этот файл содержит пол, вес и рост людей. Каждая строка содержит один из этих трёх элементов данных.

Следующий фрагмент кода считывает данные из такого файла, выбирает только данные о росте и находит максимальный, минимальный и средний рост среди всех.

Для того чтобы правильно сформировать путь к файлу, лежащему в отдельном каталоге, используется модуль *os.path* и его функция *join*. Она создаёт строку-путь в формате файловой системы ОС, в которой запускается программа.

```
import os
# Открывается файл на чтение
with open(os.path.join('data', 'dataIMT.txt'), 'r')\
        as input_file:
    # readlines считывает все строки из файла как список
    lines = input_file.readlines()
# Срез по массиву позволяет взять только рост людей
height_strings = lines[2::3]
# map применяет функцию float к каждому элементу списка.
# Таким образом список строк преобразуется в список
# действительных чисел
heights = list(map(float, height_strings))
# Ищем максимальный, минимальный и средний рост
# с помощью встроенных функций
max_height = max(heights)
min_height = min(heights)
avg_height = sum(heights) / len(heights)
print(f'Максимальный рост: {max_height},'
      f' минимальный: {min_height},'
      f' средний: {avg_height}')
```

Индекс массы тела измеряется в кг/м^2 и рассчитывается по формуле: $= m/h^2$, где: m — масса тела в килограммах, h — рост в метрах.

Пример № 2

Следующий пример кода по данным файла *dataIMT.txt* рассчитывает ИМТ каждого человека, находит минимальный и максимальный ИМТ.

```
import os
# Открывается файл на чтение
with open(os.path.join('data', 'dataIMT.txt'), 'r')\
        as input_file:
    # readlines считывает все строки из файла как список
    lines = input_file.readlines()
imts = [] # Список ИМТ
# Перебираем индексы, соответствующие элементам с весом:
# это каждый третий, начиная с индекса 1
for i in range(1, len(lines), 3):
    m = float(lines[i]) # i - это индекс для веса
    # i+1 - индекс следующего элемента - роста в см.
    # Переводим рост в метры
    h = float(lines[i+1]) / 100
    imts.append(m/h**2) # Считаем ИМТ и добавляем в список
print(f'Максимальный ИМТ: {max(imts)},'
      f' минимальный: {min(imts)}')
```

Пример № 3

Для записи данных в файл можно использовать функцию *writelines*:

```
with open("text.txt", "w") as f:
    f.writelines(['Строка 1', 'Строка 2', 'Строка 3'])
```

Аргумент *w* обозначает открытие файла на перезапись. Если файла *text.txt* не существует, он будет создан.

Дополнительно: при открытии файла можно указать его кодировку, например $utf-8$:

```
open(os.path.join('data', 'data_school_subject.txt'),
     'r', encoding='utf-8')
```

Задачи уровня А

1. Выведите на консоль первую строку из файла.
2. Прочитайте из файла матрицу. Транспонируйте её и выведите результат в отдельный файл.

3. В файле задан набор названий российских городов. Каждое название города находится на своей строке. Выберите те названия, которые начинаются на заданную букву. Букву вводит пользователь, возможно, в нижнем регистре.

Задачи уровня B

1. Прочитайте из файла вектор действительных чисел. Напишите функцию, которая удаляет из конца вектора заданное количество чисел. Проверьте, что функции переданы корректные данные. Новый вектор выведите в отдельный файл.

2. В файле заданы строки. Считайте их из файла и отсортируйте по количеству появлений буквы по убыванию.

3. Дан файл с текстом из нескольких предложений. Текст может располагаться на нескольких строках. Предложение может заканчиваться одним из трёх символов '.', '?', '!'. Предложение — это набор слов, разделённых пробелами. Найдите в тексте самое длинное слово.

Задачи уровня C

1. Дан файл с текстом из нескольких предложений. Текст может располагаться на нескольких строках. Предложение может заканчиваться одним из трёх символов '.', '?', '!'. Предложение — это набор слов, разделённых пробелами. Выведите в новый файл все предложения, в которых более заданного количества слов.

2. Дан файл с текстом из нескольких предложений. Текст может располагаться на нескольких строках. Предложение может заканчиваться одним из трёх символов '.', '?', '!'. Предложение — это набор слов, разделённых пробелами. Для каждого слова определите, сколько раз оно встречается.

3. Дан файл с текстом из нескольких глав. Каждая глава начинается со слова «Глава» и её номера на одной строке, на следующей строке идёт её название, текст главы идёт с новой строки. Между данными могут находиться пустые строки. Выведите в отдельный файл оглавление.

Пример:

Глава 1

Python

Python — это декларативный язык программирования.

Глава 2

C

C — это компилируемый статически типизированный язык программирования.

Оглавление

Глава 1. Python

Глава 2. C

4. В файле задан набор названий российских городов. Каждое название города находится на своей строке. Используя файл, напишите игру в города для пользователя и компьютера. Первым игру начинает пользователь, названия городов нельзя повторять. Можно использовать только названия городов, указанные в файле.

Использованная литература

1. Лутц, М. Изучаем Python / Марк Лутц. — 5-е изд. — СПб. : Диалектика, 2020. — Т. 2. — 713 с.

2. Лутц, М. Программирование на Python / Марк Лутц. — 4-е изд. — СПб. : Символ-Плюс, 2011. — Т. 2. — 991 с.

3. Beazley, D. Python cookbook: Recipes for mastering Python 3 / D. Beazley, B. K. Jones. — Sebastopol : O'Reilly Media, 2013. — 688 p.

4. Slatkin, B. Effective Python: 90 specific ways to write better Python / B. Slatkin. — Boston : Addison-Wesley Professional, 2019. — 480 p.

5. Python 3.11.1 documentation. — URL: https://docs.python.org/3/ (дата обращения: 08.01.2023).

Оглавление

Учебное издание

Лагутина Ксения Владимировна
Адрианова Алла Михайловна

Основы информатики на языке Python

Практикум

Редактор, корректор Л. Н. Селиванова
Компьютерный набор и верстка
К. В. Лагутина, А. М. Адрианова

Подписано в печать 28.02.2023. Формат 60×84 1/16.
Усл. печ. л. 3,3. Уч.-изд. л. 2,2.
Тираж 2 экз. Заказ

Оригинал-макет подготовлен
в редакционно-издательском отделе ЯрГУ.
Ярославский государственный университет им. П. Г. Демидова.
150003, Ярославль, ул. Советская, 14.