Изучаем Python

Содержание

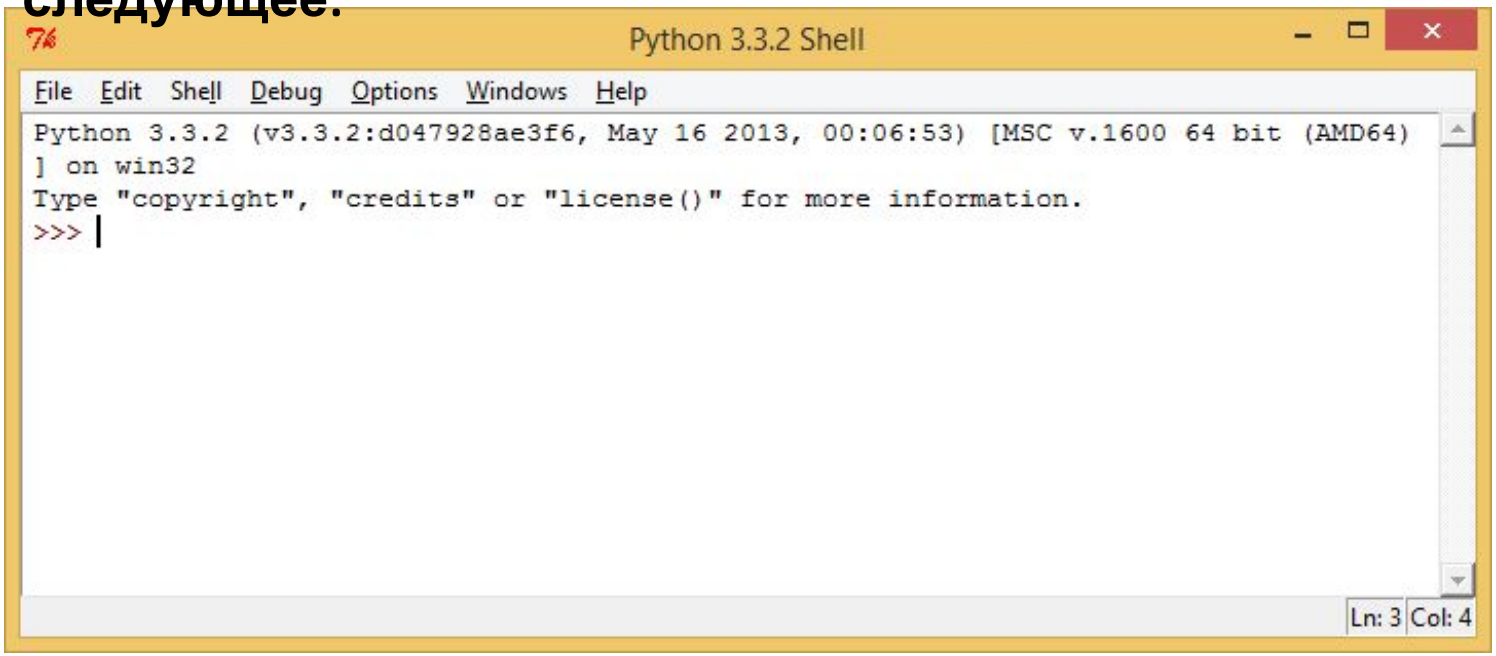
Язык Python

Добро пожаловать в мир Python!

Python — это интерпретируемый объектно-ориентированный язык программирования высокого уровня, предназначенный для самого широкого круга задач. С его помощью можно обрабатывать различные данные, создавать изображения, работать с базами данных, разрабатывать Web-сайты и приложения с графическим интерфейсом. ***Python*** является кроссплатформенным языком, позволяющим создавать программы, которые будут работать во всех операционных системах. «Скачать» и установить ***Python*** можно на официальном сайте ***Python*** http://www.python.org .

Запуск Питона

После запуска Питона вы увидите примерно следующее:



Вы находитесь в режиме командной строки среды «Питон» и можно
уже вводить команды.

Режим калькулятора

Смело вводите команды и наслаждайтесь результатом. А что можно вводить? Несколько примеров:

```
>>> 2 + 2
4
 >>> 2 ** 100
1267650600228229401496703205376
 >>> 'Hello' + 'World'
'HelloWorld'
>>> 'Привет ' * 4
'Привет Привет Привет Привет'
>>>
```

Арифметические операции

A + B сумма

A – B разность

A * B произведение

A / B частное

AB** возведение в степень

Полезно помнить, что квадратный корень из числа это **X**0.5**, а корень степени n – это **X**(1/n)**

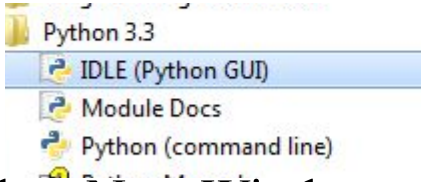
Самостоятельно: Вычислить в среде Python значение выражений (результат округлить до 2-х знаков после запятой)

1. $15{,}6 * \frac{2^{12}-3^{7}}{15*2^{13}}$ — 0.24
2. $4^{4} \cdot \sqrt[3]{15 \cdot 2{,}5}$ — 856.87
3. $\frac{-5+\sqrt{8^{2}-24}}{4}$ — 0.33

Запуск простейшей программы

Задача: Вычислить периметр прямоугольника, если заданы две стороны.

Запустите Питон:



Выберите команду File – New Window:

Наберите текст программы:

```
a=3
b=4
c=(a + b)*2
print(c)
```

Сохраните программу в рабочей папке и запустите программу (F5)

(На экране должно появиться: `14`)

Ввод данных: функция input()

```
a=input()     # Ввод строки с клавиатуры и запись в
                  # переменную a
a=int(a)      # преобразование строки в целое число
```

Можно объединить считывание строк и преобразование типов, если вызывать функцию `int` для того значения, которое вернет функция `input`:

```
a = int(input())
```

Ввод трех чисел через пробел

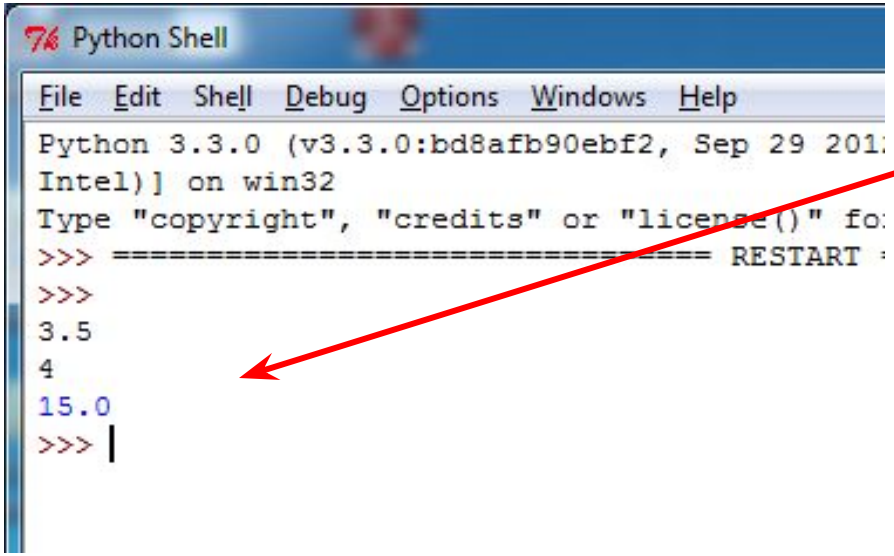
```
a, b, c = map(int, input().split())
```

Если число `a` вещественного типа, то вместо функции `int`, нужно использовать `float`, например:

```
a = float(input())
```

Решение предыдущей задачи (нахождение периметра прямоугольника) будет намного привлекательнее, если ввод сторон будет производится с клавиатуры:

```
a = float(input())
b = float(input())
c = (a + b)*2
print(c)
```



Протокол

Вывод данных: функция print()

Функция print может выводить не только значения переменных, но и значения любых выражений.

Например, допустима запись `print(2 ** 3 + 2)`.

Также при помощи функции `print` можно выводить значение не одного, а нескольких выражений,

для этого нужно перечислить их через запятую:

```
a = 1
b = 2
print(a, '+', b, '=', a + b)
```

В данном случае будет напечатан текст `1 + 2 = 3` Сначала выводится значение переменной a,

затем строка из знака “+”, затем значение переменной b, затем строка из знака “=”,

Особенности функции print()

- Выводимые данные разделяются одним пробелом;
- Если понадобится изменить это правило, то применяют специальный параметр **sep** (*separator – разделитель)*;

Например:

```
a,b,c = 2,3,1
print(a, b, c, sep='-')
```

Вывод значений a, b, c разделенных символом ‘-’

Результат:

2-3-1

- Для того, чтобы совсем убрать разделитель при выводе нужно передать параметр sep, равный пустой строке:

```
print(a, '+', b, '=', a + b, sep = '')
```

Особенности функции print()

- Для того, чтобы значения выводились с новой строке, нужно в качестве параметра **sep** передать строку, состоящую из специального символа новой строки, которая задается так:

```
print(a,b,sep='\n')
```

- Символ **'\'** в текстовых строках является указанием на обозначение специального символа, в зависимости от того, какой символ записан после него. Наиболее часто употребляется символ новой строки **'\n'**.
- Для того, чтобы вставить в строку сам символ , нужно повторить его два раза: **'\\'.**

Особенности функции print()

Вторым полезным именованным параметром функции **print** является параметр **end**, который указывает на то, что выводится после вывода всех значений, перечисленных в функции **print**. По умолчанию параметр **end** равен **'\n'**, то есть следующий вывод будет происходить с новой строки. Этот параметр также можно исправить, например, для того, чтобы убрать все дополнительные выводимые символы можно вызывать функцию **print** так:

```
print(a, b, c, sep = '', end = '')
```

Целочисленная арифметика

Операции `+, -, *, **` - могут применятся как к целым, так и к вещественным числам.

Операция деления `/` для целых чисел возвращает значение типа `float`. Также функция возведения в степень возвращает значение типа `float`, если показатель степени — отрицательное число.

Операция целочисленного деления, выполняющегося с отбрасыванием дробной части, обозначается `//`.

Например:

```
>>> 22 // 4
5
>>> 122 // 10
12
```

Целочисленная арифметика

Операция которая применяется часто к целым числам - это операция взятия остатка от деления, обозначаемая **%**:

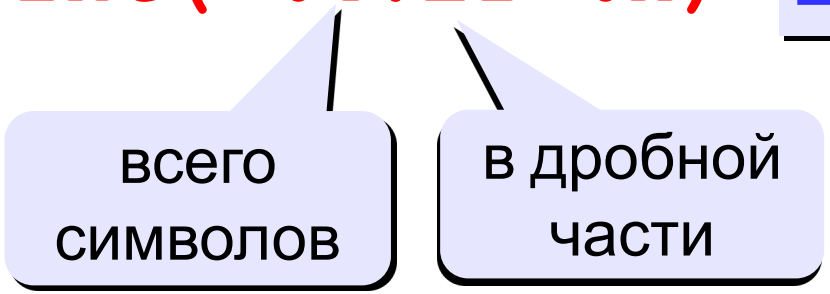
Например:

```
>>>22 % 4
2
125 % 10
5
125 % 100
25
```

Форматирование вещественных чисел при выводе

```
x=2.71828
print(“%4.2f”%x)
```

```
2.72
```




```
x,y=2.71828,3.1415
print("%4.2f"%x,"%4.2f"%y)
```

```
2.72 3.14
```

Самостоятельно: Составить программы на языке Python для решения следующих задач:

1) Даны стороны прямоугольника *a* и *b*. Найти его площадь *S* и периметр *P*.

Ввод: **9.4 5.7**

Вывод: **53.58 30.2**

2) Даны катеты прямоугольного треугольника *a* и *b*. Найти его гипотенузу *c* и периметр *P*:

Ввод: **8.8 5.7**

Вывод: **10.48 24.98**

3) Дано двузначное число. Найти сумму и произведение его цифр.

Ввод: **27**

Вывод: **9 14**

4) Дано двузначное число. Вывести число, полученное при перестановке цифр исходного числа.

Ввод: **27**

Вывод: **72**

5*) Дано число n. С начала суток прошло n минут. Определите, сколько часов и минут будут показывать электронные часы в этот момент. Программа должна вывести два числа: количество часов (от 0 до 23) и количество минут (от 0 до 59). Учтите, что число n может быть больше, чем количество минут в сутках.

Ввод: **1441**

Вывод: **0 1**

Ветвление

Условная инструкция в Питоне имеет следующий синтаксис:

if *Условие*:

Блок инструкций 1

else:

Блок инструкций 2

Для выделения блока инструкций, относящихся к инструкции if или else в языке Питон используются отступы. Все инструкции, которые относятся к одному блоку, должны иметь равную величину отступа, то есть одинаковое число пробелов в начале строки. Рекомендуется использовать отступ в 4 пробела или символ табуляции.

Логические операции

Операция	Python
равно	==
не равно	!=
больше или равно	>=
меньше или равно	<=
больше	>
меньше	<

Логические операции

Иногда нужно проверить одновременно не одно, а несколько условий. В Питоне существуют стандартные логические операторы: логическое И (and), логическое ИЛИ(or), логическое отрицание (not).

Примеры.

Проверим, что хотя бы одно из чисел a или b оканчивается на 0:

```
if a % 10 == 0 or b % 10 == 0:
```

Проверим, что число a — положительное, а b — неотрицательное:

```
if a > 0 and not (b < 0):
```

Или можно вместо not (b < 0) записать (b >= 0).

Каскадные условные инструкции

Пример программы, которая по данным ненулевым числам x и y определяет, в какой из четвертей координатной плоскости находится точка (x,y):

```
x, y = map(float, input().split())
if x > 0 and y > 0:
   print("Первая четверть")
elif x > 0 and y < 0:
   print("Четвертая четверть")
elif y > 0:
   print("Вторая четверть")
else:
   print("Третья четверть")
```

В такой конструкции условия **if ... elif** проверяются по очереди, выполняется блок, соответствующий первому из истинных условий. Если все проверяемые условия ложны, то выполняется блок **else**, если он присутствует.

Самостоятельно: Составить программы на языке Python для решения следующих задач:

1. Даны три числа. Найти сумму двух наибольших из них.

Ввод: 4 3 5

Вывод: 9

2. Даны два числа. Вывести вначале большее, а затем меньшее из них.

Ввод: 8 9

Вывод: 9 8

3. Даны три натуральных числа *a,b,c*. Определите, существует ли треугольник с такими сторонами. Если треугольник существует, вывести строку YES, иначе NO.

Ввод: 5 3 2

Вывод: NO

Самостоятельно: Составить программы на языке Python для решения следующих задач:

4. Дан номер года (положительное целое число). Определить количество дней в этом году, учитывая, что обычный год насчитывает 365 дней, а високосный — 366 дней. Високосным считается год, делящийся на 4, за исключением тех годов, которые делятся на 100 и не делятся на 400 (например, годы 300, 1300 и 1900 не являются високосными, а 1200 и 2000 — являются).

Ввод: 1248

Вывод: 366

5*. Яша плавал в бассейне размером NxM метров и устал. В этот момент он обнаружил, что находится на расстоянии x метров от одного из длинных бортиков (не обязательно от ближайшего) и y метров от одного из коротких бортиков. Какое минимальное расстояние должен проплыть Яша, чтобы выбраться из бассейна на бортик?

Программа получает на вход числа N, M, x, y. Программа должна вывести число метров, которое нужно проплыть Яше до бортика.

Ввод: 10 25 7 8

Вывод: 3

Цикл for

Цикл **for**, также называемый циклом с параметром, в языке Питон богат возможностями.
В цикле **for** указывается переменная и множество значений, по которому будет пробегать переменная. Множество значений может быть задано списком, кортежем, строкой или диапазоном. Тело цикла записывается с отступом.

В списке значений могут быть выражения различных типов, например:

```
for i in 1, 2, 3, 'one', 'two', 'three':
    print(i) # Здесь обязательно отступ
```

При первых трех итерациях цикла переменная **i** будет принимать значение типа **int**, при последующих трех — типа **str**.

Функция range

Для повторения цикла некоторое заданное число раз n можно использовать цикл for вместе с функцией range:

```
for i in range(n):          # i пробегает значения
    Тело цикла              # от 0 до n-1
```

В качестве n может использоваться числовая константа, переменная или произвольное арифметическое выражение (например, 2 ** 10). Если значение n равно нулю или отрицательное, то тело цикла не выполнится ни разу.

Можно задать цикл таким образом:

```
for i in range(a, b):   # i пробегает значения
    Тело цикла          # от a до b-1
```

Если же a≥b, то цикл не будет выполнен ни разу.

Функция range

Например, для того, чтобы просуммировать значения чисел от 1 до n можно воспользоваться следующей программой:

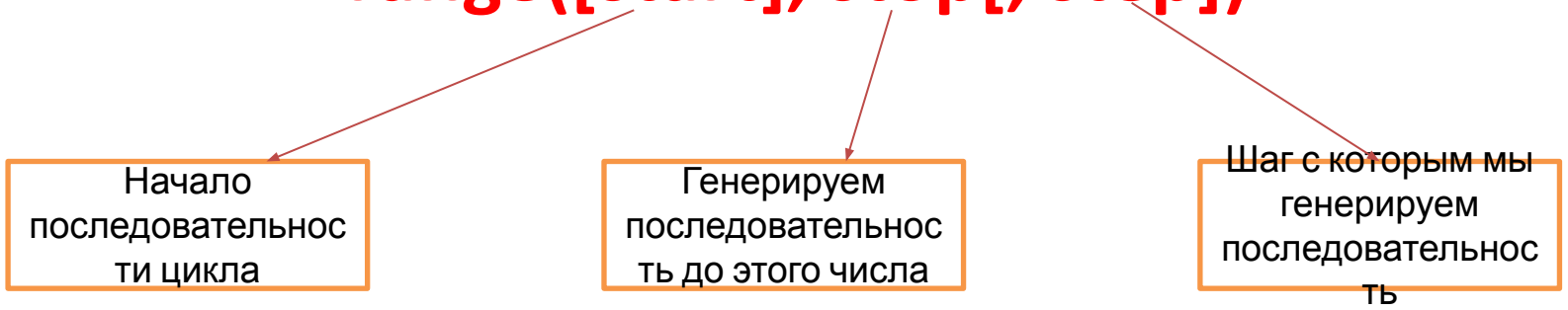
```
sum = 0
for i in range(1,5):
    sum += i
```

В этом примере переменная `i` принимает значения `1, 2, ..., n` и значение переменной `sum` последовательно увеличивается на указанные значения.

Сделать цикл по всем нечетным числам от 1 до 99 можно при помощи функции range(1, 100, 2), а сделать цикл по всем числам от 100 до 1 можно при помощи range(100, 0, -1).

- *Функция range() по умолчанию строит последовательность, в которой каждое следующее число на 1 больше предыдущего.*

- **range([start], stop[, step])**



- Все параметры должны быть целыми числами:
- Каждый из параметров может быть, как положительным, так и отрицательным.
- range() (и Python в целом) основана на индексе 0. Это означает, что список индексов начинается с 0, а не с 1. Последнее целое число, сгенерированное функцией range() зависит от stop, но не будет включать его. Например, range(0, 5) генерирует целые числа 0, 1, 2, 3, 4, не включая 5.

- **Пример 1**

```
for i in range (10, 0, -1):
    print(i*i)
```

- Программа выводит квадраты натуральных чисел от 10 до 1 в порядке убывания
- **10**: первое число последовательности.
- **0**: конечное число последовательности (не включая это число).
- **-1**: шаг

- **Пример 2**

```
for i in range (0, 101, 5):
    print(i)
```

- Программа выводит все числа от 0 до 100 с шагом 5
- **0**: первое число последовательности.
- **101**: конечное число последовательности (не включая это число).
- **5**: шаг

Самостоятельно

1. Даны два целых числа *A* и *B* (*A* < *B*). Вывести в порядке возрастания все целые числа, расположенные между *A* и *B* (включая сами числа *A* и *B*), а также количество *N* этих чисел.

Ввод: 4 10

Вывод: 4 5 6 7 8 9 10

7

2. Даны два целых числа *A* и *B* (*A* < *B*). Вывести в порядке убывания все целые числа, расположенные между *A* и *B* (не включая числа *A* и *B*), а также количество *N* этих чисел.

Ввод: 15 10

Вывод: 14 13 12 11

4

3. Даны два целых числа *A* и *B* (*A* < *B*). Найти сумму квадратов всех целых чисел от *A* до *B* включительно.

Ввод: -1 3

Вывод: 15

Самостоятельно

4. Дано целое число *N* (> 0).Найти произведение *N*! = 1·2·…·*N* (*N–факториал*). *(не использовать библиотеку math)*

Ввод: 10

Вывод: 368800

5*. Для настольной игры используются карточки с номерами от 1 до N. Одна карточка потерялась. Найдите ее, зная номера оставшихся карточек.

Дано число N, далее N-1 номер оставшихся карточек (различные числа от 1 до N). Программа должна вывести номер потерянной карточки. (Массивами и аналогичными структурами данных пользоваться нельзя.)

Ввод: 4

3 2 4

Вывод: 1

Цикл while

Цикл **while (“пока”)** позволяет выполнить одну и ту же последовательность действий, пока проверяемое условие истинно. Условие записывается до тела цикла и проверяется до выполнения тела цикла. Как правило, цикл **while** используется, когда невозможно определить точное значение количества проходов исполнения цикла.

Синтаксис цикла **while** в простейшем случае выглядит так:

while *условие*:

блок инструкций

Пример 1: Определить количество цифр натурального числа

```
n = int(input())
count = 0
while n > 0:
      n//= 10
      count+= 1
print (count)
```

В этом цикле мы отбрасываем по одной цифре числа, начиная с конца, что эквивалентно целочисленному делению на 10 (**n //= 10**), при этом считаем в переменной **count**, сколько раз это было сделано.

Пример 2: Сколько единиц в двоичной записи заданного натурального числа

```
n = int(input())
k = 0
while n != 0:
    if n % 2 == 1:
        k += 1
    n = n // 2
print(k)
```

Самостоятельно:

1. По данному целому числу N распечатайте все квадраты натуральных чисел, не превосходящие N, в порядке возрастания.

Ввод: 50

***Вывод:* 1 4 9 16 25 36 49**

2. Найти сумму цифр заданного натурального числа.

Ввод: 5523

Вывод: 15

3. Дано целое число N (> 0), являющееся некоторой степенью числа 2: $N = 2^K$. Найти целое число K — показатель этой степени.

Ввод: 512

Вывод: 9

Самостоятельно:

4. Дано целое число *N* (> 1). Если оно является *простым*, т. е. не имеет положительных делителей, кроме 1 и самого себя, то вывести 1, иначе вывести 0.

***Ввод:* 17**

***Вывод:* 1**

5*. Вклад в банке составляет *x* рублей. Ежегодно он увеличивается на *p* процентов, после чего дробная часть копеек отбрасывается. Определите, через сколько лет вклад составит не менее *y* рублей.

Программа получает на вход три натуральных числа: *x*, *p*, *y* и должна вывести одно целое число.

***Ввод:* 100 10 200**

***Вывод:* 8**

Списки

Большинство программ работает не с отдельными переменными, а с набором переменных. Например, программа может обрабатывать информацию об учащихся класса, считывая список учащихся с клавиатуры или из файла.

Для хранения таких данных можно использовать структуру данных, называемую в Питоне **список** (в большинстве же языков программирования используется другой термин “массив”).

Список представляет собой последовательность элементов, пронумерованных от 0, как символы в строке. Список можно задать перечислением элементов списка в квадратных скобках, например, список можно задать так:

P = [2, 3, 5, 7, 11, 13]

R = ['Red', 'Orange', 'Yellow', 'Green', 'Blue', 'Indigo', 'Violet']

Здесь P[0]==2, R[2]==‘Yellow’.

Способы создания и считывания списков

```
A = []
for i in range(int(input()):
   A.append(int(input())
```

В этом примере создается пустой список, далее считывается количество элементов в списке, затем по одному считываются элементы списка и добавляются в его конец.

Другой пример:

```
A = [0] * int(input())   # Задаём количество элементов и
                         # заполняем  нулями
for i in range(len(A)): # Заполняем список элементами
    A[i] = int(input())   # вводимыми с клавиатуры
```

Описание методов для работы со списками

A = [] – задает пустой массив;

A = [0] * n – задает массив, состоящий из n нулей;

a.append(x) - добавляет в конец массива a элемент x;

a.pop(x) – убирает из конца массива a элемент x;

b = list(a) - создание копии массива a;

a.insert(4, 0) - вставляет в массив a на четвертое место 0;

a.remove(2) - удаляет первую двойку из массива;

a.sort() – сортирует массив;

a.sort(reverse = True) - сортирует по убыванию;

b = list(a[3:6]) – записывает в массив b элементы массива a с третьего по шестой;

a.count(x) подсчитывает количество элементов равных x

Описание методов для работы со списками

b = a - создает связанную копию. При изменении массива a изменяется и массив b;

a = list(map(int, input().split())) – ввод массива. Размер массива задавать не надо, т.к. массив динамический;

print(a) – выведет так [1, 2, 3, 4, 5], где 1,2,3,4,5 элементы массива;

print(*a) - выведет как строку, где элементы массива разделены пробелами;

```
for x in a:
    print(x, end = ‘  ‘)
```

- выводит элементы массива через пробел

```
for i in range(len(a)):
    print(a[i], end = '  ')
```

- выводит элементы массива через пробел (i – номера элементов массива);

Примеры задач с использованием списков

Пример № 1 Найти сумму элементов массива.

```
s = list(map(int, input().split()))
a = sum(s)
print(a)
```

Пример №2 Найти минимальный элемент массива.

```
s = list(map(int, input().split()))
a = min(s)
print(a)
```

Примеры задач с использованием списков

Пример № 3 Найти индекс максимального элемента в массиве.

```
a= list(map(int, input().split()))
z = a.index(max(a))
print(z)
```

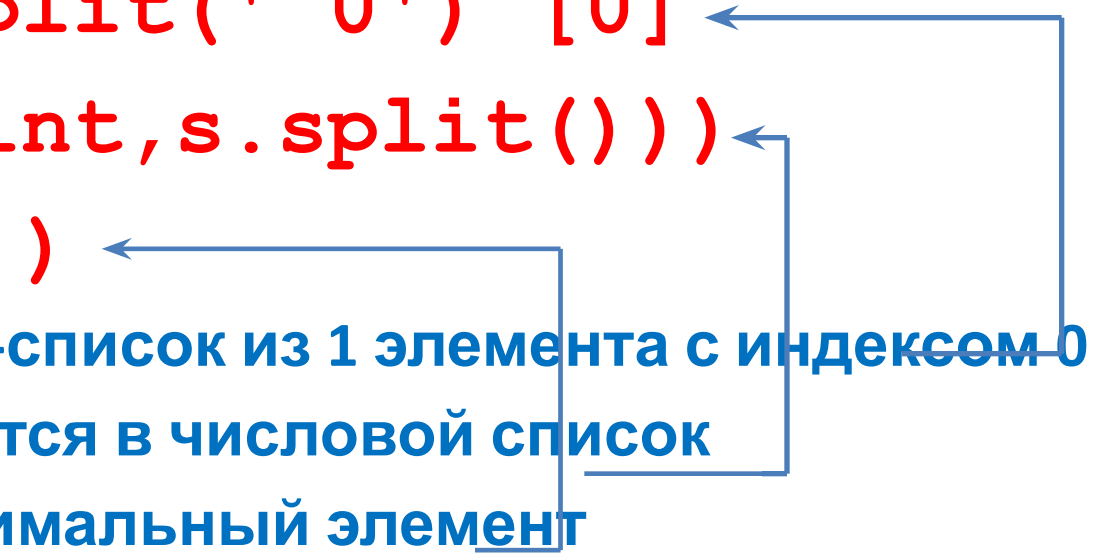
Пример №4. Отсортировать массив по возрастанию.

```
a= list(map(int, input().split()))
a.sort()
print(*a)
```

Примеры задач с использованием списков

Пример № 5. Вводятся целые числа заканчивающиеся 0. Найти максимальный элемент

```
s=input().split(' 0') [0]
x=list(map(int,s.split()))
print(max(x))
# Вводится строка-список из 1 элемента с индексом 0
# Строка переводится в числовой список
# Выводится максимальный элемент
```



Пример № 6

Сформировать и вывести целочисленный массив размера *N*, содержащий *N* первых положительных нечетных чисел: 1, 3, 5, … .

Решение 1

```
n=int(input())
a=[0]*n
for i in range(n):
    a[i]=2*i+1
print(*a)
```

Решение 2

(с использованием генераторов)

```
n=int(input())
a = [2*i+1 for i in range(n)]
print(*a)
```

Работает гораздо быстрее !

Самостоятельно:

1. Дано целое число N (> 0). Сформировать и вывести целочисленный массив размера N, содержащий степени двойки от первой до N-й: 2, 4, 8, 16, … .

***Ввод*: 7**

***Вывод*: 2 4 8 16 32 64 128**

2. Дано целое число N (> 1), а также первый член A и разность D *арифметической прогрессии*. Сформировать и вывести массив размера N, содержащий N первых членов данной прогрессии:

A, $A + D$, $A + 2 \cdot D$, $A + 3 \cdot D$,

***Ввод*: 4 1.14 3.04**

***Вывод*: 1.14 4.18 7.23 10.27**

Самостоятельно:

3. Выведите значение наименьшего из всех положительных элементов в списке. Известно, что в списке есть хотя бы один положительный элемент, а значения всех элементов списка по модулю не превосходят 1000.

***Ввод*: 5 -4 3 -2 1**

***Вывод:* 1**

4. Дан список чисел. Выведите все элементы списка, которые больше предыдущего элемента.

***Ввод:* 1 5 2 4 3**

***Вывод :* 5 4**

5*. Дан список. Выведите те его элементы, которые встречаются в списке только один раз. Элементы нужно выводить в том порядке, в котором они встречаются в списке.

***Ввод:* 2 5 2 4 4 3 2**

***Вывод :* 5 3**

Матрицы

Прямоугольные таблицы с данными называются матрицами или двумерными массивами. В языке программирования Питон таблицу можно представить в виде списка строк, каждый элемент которого является в свою очередь списком, например, чисел. Например, создать числовую таблицу из двух строк и трех столбцов можно так:

```
A = [ [1, 2, 3], [4, 5, 6] ]
```

Здесь первая строка списка A[0] является списком из чисел [1, 2, 3]. То есть A[0][0] == 1, значение A[0][1] == 2, A[0][2] == 3, A[1][0] == 4, A[1][1] == 5, A[1][2] == 6.

Матрицы

Для обработки и вывода матрицы как правило используется два вложенных цикла. Первый цикл по номеру строки, второй цикл по элементам внутри строки. Например, вывести двумерный числовой список на экран построчно, разделяя числа пробелами внутри одной строки, можно так:

```
for i in range(len(A)):
      for j in range(len(A[i]):
            print(A[i][j], end = ' ')
      print()
```

То же самое, но циклы не по индексу, а по значениям списка:

```
for row in A:
     for elem in row:
           print(elem, end = ' ')
     print()
```

Матрицы

Вывод матрицы:
(Для вывода одной строки можно воспользоваться методом join):

```
for row in A:
    print(' '.join(list(map(str, row))))
```

Используем два вложенных цикла для подсчета суммы всех чисел в списке:

```
S = 0
for i in range(len(A)):
   for j in range(len(A[i])):
      S += A[i][j]
```

Или то же самое с циклом не по индексу, а по значениям строк:

```
S = 0
for row in A:
   for elem in row:
      S += elem
```

Создание вложенных списков (матриц)

Первый способ:

```
A = [0] * n
for i in range(n):
    A[i] = [0] * m
```

Второй способ: создать пустой список, потом n раз добавить в него новый элемент, являющийся списком-строкой:

```
A = []
 for i in range(n):
     A.append([0] * m)
```

Третий способ:

```
A = [ [0] * m for i in range(n)]
```

Ввод двумерного массива

Задача. Надо ввести с клавиатуры двумерный массив, в виде n строк, каждая из которых содержит m чисел, разделенных пробелами.

1 способ:

```
A = []
for i in range(n):
    A.append(list(map(int, input().split())))
```

2 способ :

```
A = [ list(map(int, input().split())) for i in range(n)]
```

Пример обработки двумерных массивов

Задача. Пусть дан квадратный массив из n строк и n столбцов. Необходимо элементам, находящимся на главной диагонали присвоить значение 1, элементам, находящимся выше главной диагонали – значение 0, элементам, находящимся ниже главной диагонали – значение 2. То есть получить такой массив (пример для n==4):

1 0 0 0

2 1 0 0

2 2 1 0

2 2 2 1

Примеры обработки двумерных массивов

Решение:

```
n=4
A = [ [0] * n for i in range(n)]
for i in range(n):
    for j in range(n):
        if i < j:
            A[i][j] = 0
        elif i > j:
            A[i][j] = 2
        else:
            A[i][j] = 1
for row in A:
    print(' '.join(list(map(str,row))))
```

Самостоятельно:

1. Даны целые положительные числа *M* и *N*. Сформировать целочисленную матрицу размера $M \times N$, у которой все элементы i-й строки имеют значение $10 \cdot i$ ($i=1, \ldots, M$).
2. Даны целые положительные числа M и N. Сформировать целочисленную матрицу размера $M \times N$, у которой все элементы j-го столбца имеют значение $5 \cdot j$ ($j=1, \ldots, N$).
3. Дана матрица размера $M \times N$. Для каждой строки матрицы найти сумму ее элементов.

Самостоятельно:

4. Дана матрица размера *M* × *N*. В каждом столбце матрицы найти максимальный элемент.

5*. Дана матрица размера *M* × *N*. Преобразовать матрицу, поменяв местами минимальный и максимальный элемент в каждой строке.

Строки

Строка считывается со стандартного ввода функцией input(). Напомним, что для двух строк определена операция сложения (конкатенации), также определена операция умножения строки на число.

Строка состоит из последовательности символов. Узнать количество символов (длину строки) можно при помощи функции **len**:

```
>>> S = 'Привет!'
>>> print(len(S))
7
```

Срезы

Срез (slice) — извлечение из данной строки одного символа или некоторого фрагмента подстроки или подпоследовательности.

Есть три формы срезов. Самая простая форма среза: взятие одного символа строки, а именно, S[i] — это срез, состоящий из одного символа, который имеет номер i, при этом считая, что нумерация начинается с числа 0. То есть если **S='Hello',**
то S[0]=='H', S[1]=='e', S[2]=='l', S[3]=='l', S[4]=='o'.

Номера символов в строке (а также в других структурах данных: списках, кортежах) называются *индексом*.

Срезы

Если указать отрицательное значение индекса, то номер будет отсчитываться с конца, начиная с номера -1. То есть S[-1]=='o', S[-2]=='l', S[-3]=='l',S[-4]=='e', S[-5]=='H'.

Строка S	H	e	l	l	o
Индекс	S[0]	S[1]	S[2]	S[3]	S[4]
Индекс	S[-5]	S[-4]	S[-3]	S[-2]	S[-1]

Срез с двумя параметрами: S[a:b] возвращает подстроку из b-a символов, начиная с символа с индексом a, то есть до символа с индексом b, не включая его. Например, S[1:4]=='ell', то же самое получится если написать S[-4:-1]. Можно использовать как положительные, так и отрицательные индексы в одном срезе, например, S[1:-1] — это строка без первого и последнего символа.

Срезы

При использовании такой формы среза ошибки IndexError никогда не возникает. Например, срез **S[1:5]** вернет строку **'ello'**, таким же будет результат, если сделать второй индекс очень большим, например, **S[1:100]** (если в строке не более 100 символов).

Если опустить второй параметр (но поставить двоеточие), то срез берется до конца строки. Например, чтобы удалить из строки первый символ, то можно взять срез **S[1:]**, аналогично если опустить первый параметр, то срез берется от начала строки. То есть удалить из строки последний символ можно при помощи среза **S[:-1].** Срез **S[:]** совпадает с самой строкой **S.**

Если задать срез с тремя параметрами **S[a:b:d]**, то третий параметр задает шаг, как в случае с функцией range, то есть будут взяты символы с индексами **a, a+d, a+2*d** и т.д. При задании значения третьего параметра, равному **2**, в срез попадет каждый второй символ, а если взять значение среза, равное **-1**, то символы будут идти в обратном порядке.

Методы строк

Функция или метод	Назначение
S1 + S2	Конкатенация (сложение строк)
S1 * 3	Повторение строки
S[i]	Обращение по индексу
S[i:j:step]	Извлечение среза
len(S)	Длина строки
str in S	Проверка на вхождение подстроки в строку
S.find(str, [start],[end])	Поиск подстроки в строке. Возвращает номер первого вхождения или -1
S.rfind(str, [start],[end])	Поиск подстроки в строке. Возвращает номер последнего вхождения или -1
S.replace(шаблон, замена)	Замена

Методы строк

S.split(символ)	Разбиение по разделителю
S.isdigit()	Состоит ли строка из цифр
S.isalpha()	Состоит ли строка из букв
S.isalnum()	Состоит ли строка из цифр или букв
S.islower()	Состоит ли строка из символов в нижнем регистре
S.isupper()	Состоит ли строка из символов в верхнем регистре
S.istitle()	Начинаются ли слова в строке с заглавной буквы
S.upper()	Преобразование строки к верхнему регистру
S.lower()	Преобразование строки к нижнему регистру
S.startswith(str)	Проверка начала строки

Методы строк

S.endswith(str)	**Проверка окончания строки**
S.join(список)	Сборка строки из строкового списка с разделителем S
ord(символ)	Символ в его код ASCII
chr(число)	Код ASCII в символ
S.capitalize()	Переводит первый символ строки в верхний регистр, а все остальные - в нижний
S.count(str, [start],[end])	Возвращает количество вхождений подстроки в диапазоне [начало, конец]

Методы строк

S.lstrip([chars])	Удаление пробельных символов в начале строки
S.rstrip([chars])	Удаление пробельных символов в конце строки
S.strip([chars])	Удаление пробельных символов в начале и в конце строки

Методы строк

S.swapcase()	**Переводит символы нижнего регистра в верхний, а верхнего – в нижний**
S.title()	Первую букву каждого слова переводит в верхний регистр, а все остальные - в нижний
S.zfill(width)	Делает длину строки не меньшей width, по необходимости заполняя первые символы нулями

Методы строк

S.ljust(width, fillchar=" ")	Делает длину строки не меньшей width, по необходимости заполняя последние символы символом fillchar
S.rjust(width, fillchar=" ")	Делает длину строки не меньшей width, по необходимости заполняя первые символы символом fillchar
S.format(*args, **kwargs)	Форматирование строки

Примеры задач на обработку строк:

Пример 1

Дана непустая строка *S*. Вывести строку, содержащую символы строки *S*, между которыми вставлено по одному пробелу.

Решение 1

```
s=input()
p=''
for i in range(len(s)):
    p=p+s[i:i+1]+' '
print(p)
```

Решение 2

```
s = list(input())
print(' '.join(s))
```

Примеры задач на обработку строк:

Пример 2

Дана строка, состоящая из русских слов, набранных заглавными буквами и разделенных пробелами (одним или несколькими). Найти количество слов, которые начинаются и заканчиваются одной и той же буквой.

```
s = list(input().split())
k=0
for slovo in s:
    if slovo[0]==slovo[-1]:
        k+=1
print(k)
```

Самостоятельно:

1. Дано целое число *N* (1 ≤ *N* ≤ 26).
Вывести *N* последних *строчных* (т. е. маленьких) букв латинского алфавита в обратном порядке (начиная с буквы «z»).

2. Дан символ *C*, изображающий цифру или букву (латинскую или русскую). Если *C* изображает цифру, то вывести строку «digit», если латинскую букву — вывести строку «lat», если русскую — вывести строку «rus».

3. Дана строка, состоящая из русских слов, набранных заглавными буквами и разделенных пробелами (одним или несколькими). Найти количество слов, которые содержат хотя бы одну букву «А».

Самостоятельно:

4. Дана строка, состоящая из русских слов, разделенных пробелами (одним или несколькими). Вывести строку, содержащую эти же слова, разделенные одним пробелом и расположенные в обратном порядке.

5*. Дана строка, в которой буква h встречается как минимум два раза. Разверните последовательность символов, заключенную между первым и последним появлением буквы h, в противоположном порядке.

Ввод: **In the hole in the ground there lived a hobbit**

Вывод: **In th a devil ereht dnuorg eht ni eloh ehobbit**

Функции

Рассмотрим задачу вычисления числа сочетаний сочетаний из n элементов по k, для чего необходимо вычисление факториалов трех величин: n, k и n-k.

$$\frac{N!}{K!\,(N-K!)}$$

Это можно сделать три цикла, что приводит к увеличению размера программы за счет трехкратного повторения похожего кода. Вместо этого лучше сделать одну *функцию*, вычисляющую факториал любого данного числа n и трижды использовать эту функцию в своей программе. Соответствующая функция может выглядеть так:

```
def factorial(n):
    f = 1
    for i in range(2, n + 1):
        f *= i
    return f
```

Функции

Полное решение будет выглядеть так:

```
def factorial(n):
    f = 1
    for i in range(2, n + 1):
        f *= i
    return f
n = int(input())
k = int(input())
print factorial(n) // (factorial(k) * factorial(n - k))
```

Самостоятельно:

1. Описать функцию ***DigitCount(K)*** целого типа, находящую количество цифр целого положительного числа ***K***. Используя эту функцию, найти количество цифр для каждого из пяти данных целых положительных чисел.

2. Описать функцию ***SumRange(A, B***) целого типа, находящую сумму всех целых чисел от ***A*** до ***B*** включительно (***A*** и ***B*** — целые). Если ***A > B***, то функция возвращает ***0***. С помощью этой функции найти суммы чисел от ***A*** до ***B*** и от ***B*** до ***C***, если даны числа ***A, B, C.***

3.Описать функцию ***TriangleP(a, h)***, находящую периметр равнобедренного треугольника по его основанию **a** и высоте ***h***, проведенной к основанию (***a u*** **h** — вещественные). С помощью этой функции найти периметры трех треугольников, для которых даны основания и высоты. Для нахождения боковой стороны *b* треугольника использовать *теорему Пифагора*: $b^2 = (a/2)^2 + h^2$.

Самостоятельно:

4. Дано натуральное число ***n*>1**. Проверьте, является ли оно простым. Программа должна вывести слово YES, если число простое и NO, если число составное.

Решение оформите в виде функции IsPrime(n), которая возвращает True для простых чисел и False для составных чисел.

5*. Дано действительное положительное число *a* и целое неотрицательное число ***n***. Вычислите a^n не используя циклы и стандартную функцию pow, а используя рекуррентное соотношение $a^n = a \cdot a^{n-1}$.

Решение оформите в виде функции **power(a, n).**

Модуль math

Модуль **math** – один из важнейших библиотек в Python. Этот модуль предоставляет обширный функционал для работы с числами.

Подключение:

	Запись перед использованием	Использование в программе
1 способ	import math	math.sqrt(x)
2 способ	from math import sqrt	sqrt(x)

Функции модуля math

- **math.ceil(X)** – округление до ближайшего большего числа.
- **math.copysign(X, Y)** - возвращает число, имеющее модуль такой же, как и у числа X, а знак - как у числа Y.
- **math.fabs(X)** - модуль X.
- **math.factorial(X)** - факториал числа X.
- **math.floor(X)** - округление вниз.
- **math.fmod(X, Y)** - остаток от деления X на Y.
- **math.frexp(X)** - возвращает мантиссу и экспоненту числа.
- **math.ldexp(X, I)** - $X * 2^i$. Функция, обратная функции math.frexp().
- **math.fsum(последовательность)** - сумма всех членов последовательности. Эквивалент встроенной функции sum(), но math.fsum() более точна.
- **math.isfinite(X)** - является ли X числом.
- **math.isinf(X)** - является ли X бесконечностью.

Функции модуля math

- **math.isnan(X)** - является ли X NaN (Not a Number - не число).
- **math.modf(X)** - возвращает дробную и целую часть числа X. Оба числа имеют тот же знак, что и X.
- **math.trunc(X)** - усекает значение X до целого.
- **math.exp(X)** - e^X.
- **math.expm1(X)** - e^X - 1. При X -> 0 точнее, чем math.exp(X)-1.
- **math.log(X, [base])** - логарифм X по основанию base. Если base не указан, вычисляется натуральный логарифм.
- **math.log1p(X)** - натуральный логарифм (1 + X). При X -> 0 точнее, чем math.log(1+X).
- **math.log10(X)** - логарифм X по основанию 10.
- **math.pow(X, Y)** - X^Y.
- **math.sqrt(X)** - квадратный корень из X.

Функции модуля math

- **math.acos(X)** - арккосинус X. В радианах.
- **math.asin(X)** - арксинус X. В радианах.
- **math.atan(X)** - арктангенс X. В радианах.
- **math.atan2(Y, X)** - арктангенс Y/X. В радианах. С учетом четверти, в которой находится точка (X, Y).
- **math.cos(X)** - косинус X (X указывается в радианах).
- **math.sin(X)** - синус X (X указывается в радианах).
- **math.tan(X)** - тангенс X (X указывается в радианах).
- **math.hypot(X, Y)** - вычисляет гипотенузу треугольника с катетами X и Y (math.sqrt(x * x + y * y)).
- **math.degrees(X)** - конвертирует радианы в градусы.
- **math.radians(X)** - конвертирует градусы в радианы.
- **math.pi** - pi = 3,1415926...
- **math.e** - e = 2,718281...