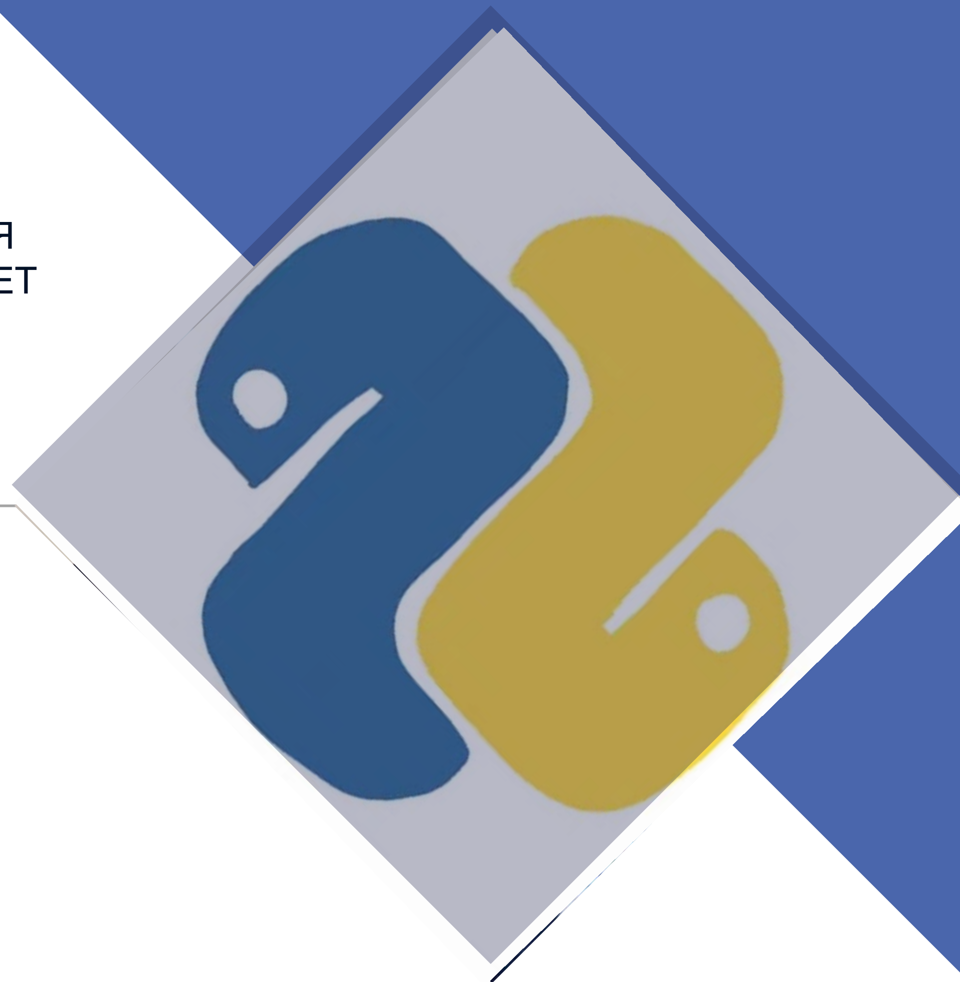
#МинцифрыРоссии

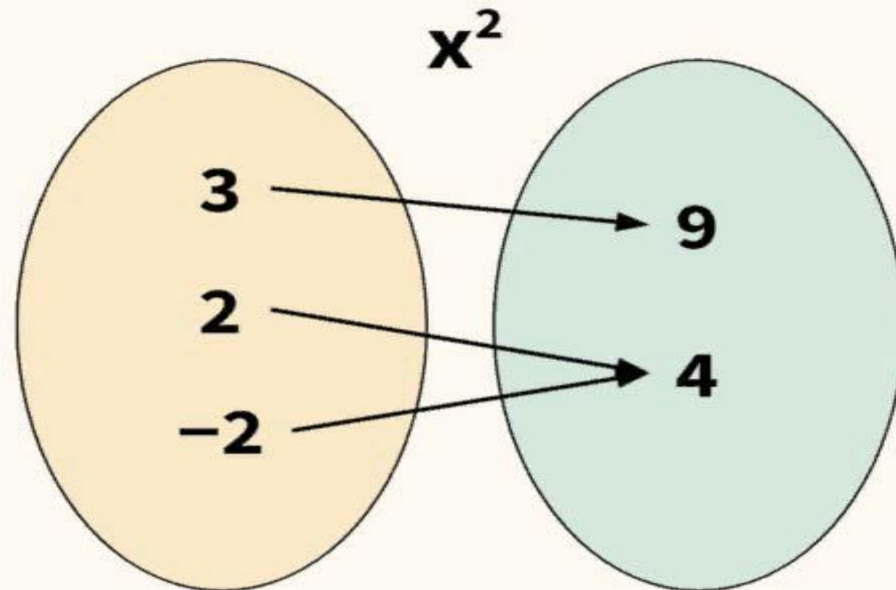
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«КАБАРДИНО-БАЛКАРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
им. Х.М. БЕРБЕКОВА»

ИНСТИТУТ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА И ЦИФРОВЫХ
ТЕХНОЛОГИЙ



Оператор RETURN, функции с произвольным числом параметров

Функция в математике



Функция в программировании

```
код вне функции

def название_функции():
    отдельный блок кода


def название_функции():
    отдельный блок кода
```

ПОНЯТИЕ ФУНКЦИИ

Функция — это фрагмент программного кода, который решает какую-либо задачу, объект, принимающий аргументы и возвращающий значение.

Функция может использоваться для обработки данных, она получает на вход значения, обрабатывает его и возвращает результат в программу. Также она может не возвращать значение, а выводить его на экран или записывать в файл.

Его можно вызывать в любом месте основной программы. При вызове происходит выполнение команд тела функции.

Объявление и вызов функции

Пример

```
# создадим функцию, которая удваив
# любое передаваемое ей значение
def double(x):
        res = x * 2
         return res
# и вызовем ее, передав число 2
double(2)
4
```

ключевое слово | название функции | параметр функции

def double(res) :

отступ → res * 2

return res

ключевое слово

тело функции

1) ключевое слово **def** необходимо для объявления функции;
2) далее идут **название функции**, которое вы сами определяете, и **параметры**, которые может принимать ваша функция;
3) после двоеточия на новой строке с отступом идет **тело функции**, то есть то, что будет исполняться при вызове функции;
4) в конце ставится ключевое слово **return**, возвращающее результат работы функции.

Пустое тело функции.

Оставлять тело функции совсем пустым нельзя. Нужно как минимум указать ключе слово **return** или оператор **pass**.

```
# тело функции не может быть пустым
def only_return():

  # нужно либо указать ключевое слово return
  return
only_return()
# либо оператор pass
def only_pass():
  pass
only_pass()

print(only_return())
None
```

Функция print() вместо return

Помимо ключевого слова **return**, результат работы функции можно вывести помощью **print()**.

```
def double_print(x):
  res = x * 2
  print(res)
double_print(5)
10
```

Использование return возвращает значение функции (в нашем случае значение переменной res) и прерывает ее работу

Функция print() просто выводит это значение пользователю и не влияет на дальнейшее исполнение кода, если он есть

Параметры собственных функций

у собственных функций те же самые возможности, что и у встроенных функци[й, в] частности, параметры могут быть позиционными и именованными.

```
# объявим функцию с параметрами x и y,
def calc_sum(x, y):   # которая возвращает их сумму
  return x + y
# вызовем эту функцию с одним позиционным и одним именованным параметром
calc_sum(1, y = 2)
3
```

Параметры собственной функции также могут быть заданы по умолчанию. В этом случае при вызове функции их указывать не обязательно.

```
def calc_sum_default(x = 1, y = 2):
  return x + y
 calc_sum_default()
3

# эта функция просто выводит текст 'Some string'
def print_string():
  print('Some string')
 print_string()
Some string
```

Аннотация функции

Аннотация функции (function annotation) позволяет явно прописать тип данных параметров (parameter annotation) и возвращаемых значений (return annotation).

```
# укажем, что на входе функция принимает тип float, а возвращает int
# значение 3,5 - это значение параметра x по умолчанию
def f(x: float = 3.5) -> int:
  return int(x)
# желаемый тип данных можно посмотреть через атрибут __annotations__
f.__annotations__
{'return': int, 'x': float}
# вызовем функцию без параметров
f()
3
```

Аннотация не является обязательной и никак не вляет на выполнение кода.

```
# сохраним аннотации, но изменим суть функции
def f(x: float) -> int:
  # теперь вместо int она будет возвращать float
  return float(x)
# вновь вызовем функцию, передав ей на входе int, и ожидая на выходе получить float
f(3)
3.0
```

Дополнительные возможности функций

Вызов функции можно совмещать с арифметическими операциями.

```
# вызовем объявленную выше функцию и умножим ее вывод на два
calc_sum(1, 2) * 2
6
```

Доступны и логические операции.

```
calc_sum(1, 2) > 2
True
```

Если результатом вывода является строка, то у этой строки также есть индекс.

```
def first_letter():
  return 'Python'

# обратимся к первой букве слова Python
first_letter()[0]
'P'
```

Дополнительные возможности функций

Вызов функции можно совмещать с арифметическими операциями.

```
# вызовем объявленную выше функцию и умножим ее вывод на два
calc_sum(1, 2) * 2
6
```

Доступны и логические операции.

```
calc_sum(1, 2) > 2
True
```

Если результатом вывода является строка, то у этой строки также есть индекс.

```
def first_letter():
  return 'Python'

# обратимся к первой букве слова Python
first_letter()[0]
'P'
```

Функция может не использовать параметры, но получать данные от пользователя через input().

```
def use_input():

  # запросим у пользователя число и переведем его в тип данных int
  user_inp = int(input('Введите число: '))

  # возведем число в квадрат
  result = user_inp ** 2

  # вернем результат
  return result

# вызовем функцию
use_input()
```

Появится окно для ввода числа.

функция может получать данные от пользователя
Введем число пять и посмотрим на результат.

Введите число: 5
25

Использование библиотек

Внутри функций можно использовать дополнительные библиотеки Питона. Например, применим функцию mean() библиотеки Numpy для расчета среднего арифметического.

```
# на входе функция примет список или массив x,
def mean_f(x):

  # рассчитает среднее арифметическое и прибавит единицу
  return np.mean(x) + 1
# перед вызовом функции нужно не забыть импортировать соответствующую библиотеку
import numpy as np

# и подготовить данные
x = [1, 2, 3]

mean_f(x)
3.0
```

Причины писать функции:

Функции помогают избегать дублирования кода при многократном его использовании.

Снижение сложности кода.

Меньше ошибок от многократного переписывания кода.

Программы легче поддерживать и понимать.

Программисты используют функции, чтобы сделать программу модульной

Правила объявления функции:

- Объявление происходит с помощью ключевого слова def, за ним идёт имя функции и круглые скобки ().
- Аргументы, передаваемые в функцию, должны находится в круглых скобках. Там же можно определить их значения по умолчанию, указав их после знака равно.
- Перед основным содержимым желательно включить строку документации (docstring), которая обычно описывает назначение и основные принципы работы функции.
- Тело функции начинается после знака двоеточия. Важно не забыть об отступах.
- Чтобы выйти из функции в Python, используют оператор return [значение]. Если оператор опущен, будет возвращено значение None.

Синтаксис объявления:

```
def Имя(аргументы):
    "Документация"
    Тело (инструкции)
    return [значение]
```

Пример

```
def sum(a,b):
    x=a+b
    return x
```

def – ключевое слово
sum –идентификатор
a,b – параметры
x – значение которое вернет функция когда будет вызвана после ключевого слова return

Функции в Python определяются с помощью ключевого слова def:

abs()
round ()
len()
int()
float()

print()
input()
sum()
max()
min()
…

def < имя функции>([список параметров]):

оператор 1

оператор 2

…

Параметр — это переменная, которой будет присваиваться входящее в функцию значение

Аргумент — само это значение, которое передается в функцию при её вызове.

Параметры функции и аргументы функции необходимо указывать через запятую (может быть сколько угодно).

Функции можно записывать в одну строку

Пример 1

```
def sum(x, y): print(x + y)
sum(5, 6)
```

В функции можно использовать неограниченное количество параметров, но число аргументов должно точно соответствовать параметрам. Эти параметры представляют собой позиционные аргументы.

Пример 2

```
def drawBox(a=2, b=3):
    c= a + b
    print(c)

drawBox(5,7)
```

В Python есть возможность задать для аргументов значение по умолчанию. Если значение для такого аргумента при вызове функции не передаётся, то используется значение по умолчанию.

Часто в функции передаётся большое количество аргументов и вспомнить порядок их перечисления в функции бывает сложно. В Python есть возможность передать значение аргумента по его имени. В таком случае аргумент становится уже не позиционным, а именованным. Именованному аргументу присваивается значение при вызове функции.

Пример

```
def final_price(price, discount=1):
    return price - price * discount / 100

print(final_price(1000, discount=5))
print(final_price(discount=10, price=1000))
```

Вывод программы:

```
950.0
900.0
```

оператор return

Ключевая особенность функций — возможность возвращать значение.

С помощью оператора return из функции можно вернуть одно или несколько значений.

Возвращаемым объектом может быть:

- Число;
- Строка;
- None;
- Список;
- Иной контейнер.

Пример:

```
def x(n):
    a = [1,3]
    a = a * n
    return a
print(x(2)) # выведет [1,3,1,3]
```

Возврат значений:

Чтобы вернуть несколько значений, нужно написать их через запятую. Python позволяет вернуть из функции список или другой контейнер: достаточно указать после ключевого слова return имя контейнера.

Возврат простого значения:

Аргументы можно использовать для изменения ввода и таким образом получать вывод функции. Но куда удобнее использовать инструкцию return, примеры которой уже встречались ранее. Если ее не написать, функция вернет значение None.

Пример 1

```
a=abs(-7) #Встроенные функции
print(a)
7
b=max(4,5,7,4,3,2)
print(b)
7
b=max(4,abs(-90),5,7,4,min(100,200),3,2)
print(b)
100
```

Пример 2

```
def square(x):  #Функция возводит число в квадрат
     print(x**2)
a= square(6)
print(a)
36
None
```

Пример 1

```
def example():
      print(1)
      print(2)
example()
1
2
```

Пример 2

```
def example():
      return 1
      return 2
      return 3
 print(example())
1
```

Пример 3

```
def sqr(x):
      return(x*x)
def print(sqr(a)):
      print(“sqr=“,a)
y=5
print_sqr(sqr(y))
sqr=25
```

Пример 1

```
def ss(x,y):
    if x>y:
        return x,y
    else:
        return y,x
print(ss(3,8))
```
(8,3)

Пример 3

```
def sqAndPer(a,b):#нахождение площади и периметра
    return a*b,2*(a+b)
square,perimeter = sqAndPer(2,5)
print(square,perimeter)
```
10 14

Пример 2

```
def even(x):
        return  x%2==0
for i in range (1,6):
    print(i,even(i))
```

Пример 4

```
def sqAndPer(a,b):#нахождение площади и периметра
    return a*b,2*(a+b)
print(sqAndPer(3,6))
```
(18,18)

Пример 1

```
def sqr(x):
     return(x*x)
def print_sqr(a):
     print("sqr=",a)
y=5
print_sqr(sqr(y))
sqr=25
```

Пример 3

```
Def factorial(x):
    pr=1
    for i in range(2,x+1):
         pr=pr*i
    return pr
```

Пример 2

```
def ss(x,y):
    if x>y:
        return x,y
    else:
        return y,x
print(ss(3,8))
  (8,3)
```

Пример 4

```
def sochet(n,k):
     return
factorial(n)/(factorial(k)*factorial(n-k))
print(sochet(5,3))
10.0
```

Аргументы функции

Часто возникает необходимость создать такую функцию, которая может принимать разное количество аргументов. Это можно реализовать с помощью передачи списка или массива, однако Python позволяет использоваться более удобный подход (также с использованием списка).

Функция может принимать произвольное количество аргументов или не принимать их вовсе.

Распространены функции:

- с произвольным числом аргументов;
- с позиционными и именованными аргументами;
- с обязательными и необязательными аргументами.

ПРИМЕР 1

```
>>> def func(*args):
...     return args
...
>>> func(1, 2, 3, 'abc')
(1, 2, 3, 'abc')
>>> func()
()
>>> func(1)
(1,)
```

ПРИМЕР 2

```
>>> def func(**kwargs):
...     return kwargs
...
>>> func(a=1, b=2, c=3)
{'a': 1, 'c': 3, 'b': 2}
>>> func()
{}
>>> func(a='python')
{'a': 'python'}
```

Функция с переменным числом аргументов

Для того чтобы функция могла принять переменное количество аргументов, перед именем аргумента ставится символ: * .

Когда программист передаёт аргументы, они записываются в кортеж, имя которого соответствует имени аргумента.

Вместо одного символа звёздочки можно использовать два, тогда аргументы будут помещаться не в список, а в словарь.

Пример 1

```
def variable_len(*args):
    for x in args:
        print(x)
variable_len(1,2,3) # Выведет 1,2,3
```

Пример 2

```
def variable_len(**args):
    print(type(args))
    for x, value in args.items():
        print(x, value)
variable_len(apple = "яблоко", bread = "хлеб")
# Выведет apple яблоко bread хлеб
```

Модифицируем нашу функцию из примера про скидки так, чтобы мы могли передать в неё любое количество цен, а вернуть список цен со скидкой:

```
def final_price(*prices, discount=1):
    return [price - price * discount / 100 for price in prices]

print(final_price(100, 200, 300, discount=5))
```

Вывод программы: [95.0, 190.0, 285.0]

Аргументы переменной длины (args, kwargs)

Переменная args составляет кортеж из переданных в функцию аргументов.

ПРИМЕР 1

```
def infinity(*args):
    print(args)
infinity(42, 12, 'test', [6, 5])
 (42, 12, 'test', [6, 5])
```

ПРИМЕР 2

```
def func(*args):
    return args
func(1, 2, 3, 'abc')
(1, 2, 3, 'abc')
func()
()
func(1)
(1,)
```

ПРИМЕР 3

```
def func(**kwargs):
    return kwargs
func(a=1, b=2, c=3)
{'a': 1, 'c': 3, 'b': 2}
func()
{}
func(a='python')
{'a': 'python'}
```

Чтобы функция могла принимать неограниченное количество именованных аргументов, нужно при её объявлении поставить параметр с **(сокращение от "keyword arguments").

kwargs уже заключает аргументы не в кортеж, а в словарь

ПРИМЕР 4

```
def named_infinity(**kwargs):
    print(kwargs)
named_infinity(first='nothing', second='else', third='matters')
{'first': 'nothing', 'second': 'else', 'third': 'matters'}
```

СПАСИБО ЗА ВНИМАНИЕ!