

Тема 5. Функции в Python



python

Введение в функции

В программировании **функция** - это фрагмент кода, к которому можно неоднократно обращаться из разных точек программы.

Преимущества функций

- **Возможность многократного использования:** очень часто задача требует выполнения множества однотипных операций, причём операции могут быть очень сложными, а их код - очень длинным. Если не вынести такую операцию в функцию, а перепечатывать её каждый раз - код будет перегруженным и плохо читаемым, а мы нарушим один из важных принципов программирования - **Don't repeat yourself** (*не повторяйся*).
- **Абстракция:** использование функций позволяет абстрагироваться от каждого конкретного случая и работать с множеством однотипных задач.
- **Модульность:** разбиение кода на функции для решения отдельных подзадач упрощает тестирование и поддержку кода, а также улучшает его читаемость.

Объявление и определение функции

```
def myFunction():  
    # тело функции
```

Чтобы выполнить код функции, её необходимо вызвать. Синтаксисом вызова функции: имя функции и круглые скобки, в которых указаны аргументы - значения, передаваемые в функцию.

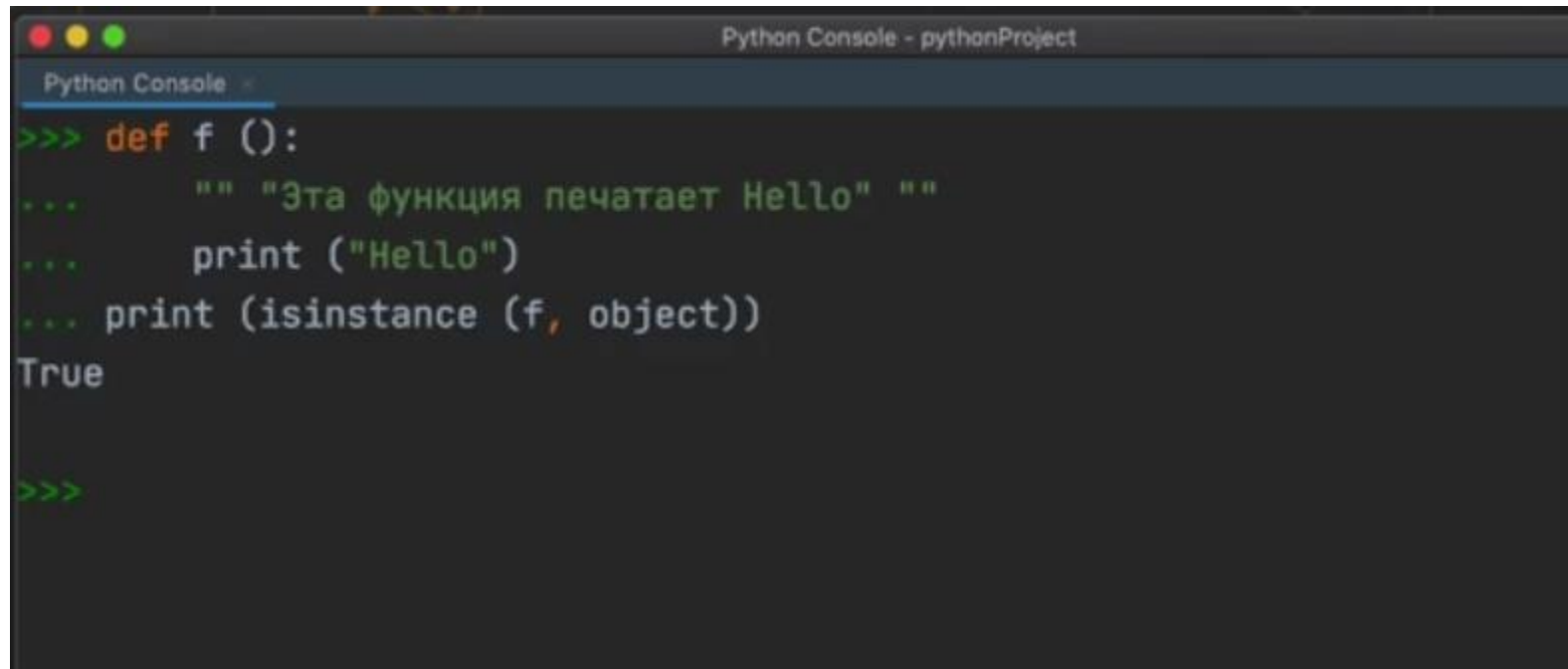
Аргументы часто путают с параметрами:

- Параметры - переменные, которым присваиваются переданные в функцию значения.
- Аргументы - сами значения, переданные в функцию в момент её вызова.

```
def sumAndPrint(a,b):  
    result = a + b  
    print(result)
```

```
sumAndPrint(2,3)
```

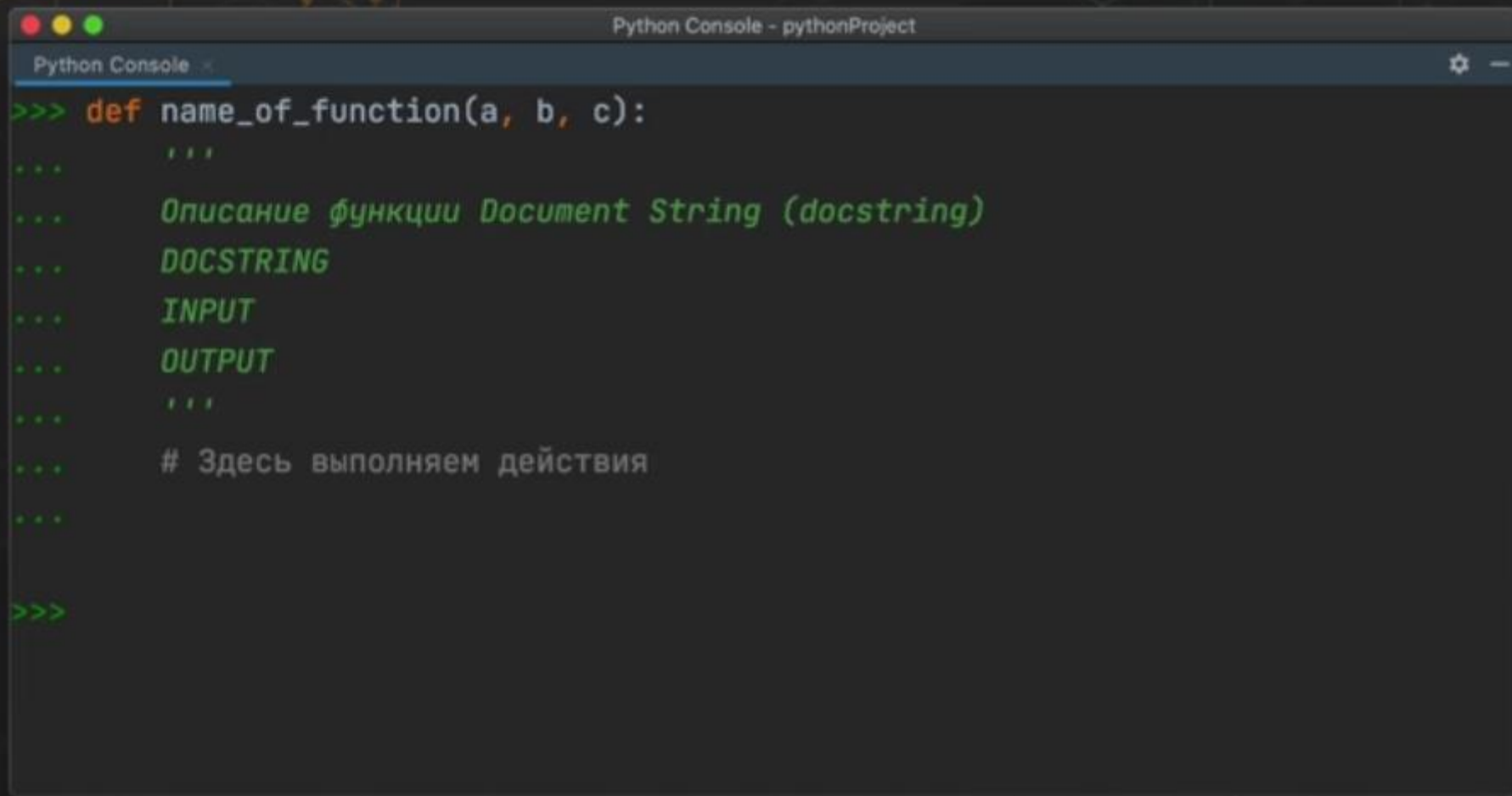
```
#>>> 5
```



A screenshot of a Python Console window titled "Python Console - pythonProject". The console shows a function definition and its execution. The function `f` is defined with a docstring in Russian and a `print` statement. The function is then called, and the output `True` is displayed. The prompt `>>>` is shown at the end of the execution.

```
>>> def f ():  
...     """ "Эта функция печатает Hello" """  
...     print ("Hello")  
...     print (isinstance (f, object))  
True  
  
>>>
```

help() dir()



The image shows a screenshot of a Python Console window titled "Python Console - pythonProject". The console displays a function definition for `name_of_function(a, b, c)`. The function has a docstring in Russian: "Описание функции Document String (docstring)". Below the docstring, there are labels for `DOCSTRING`, `INPUT`, and `OUTPUT`. The function body contains a comment: `# Здесь выполняем действия`. The console prompt `>>>` is visible at the end of the code block.

```
>>> def name_of_function(a, b, c):  
...     '''  
...     Описание функции Document String (docstring)  
...     DOCSTRING  
...     INPUT  
...     OUTPUT  
...     '''  
...     # Здесь выполняем действия  
...  
>>>
```

```
Python Console - pythonProject
Python Console x
>>> def func(a, b):
...     summ = a + b
...     product = a * b
...     return summ, product
... print(func(5, 6))
(11, 30)

>>> |
```

```
>>> def display (name, age, job):
...     print ("Name: ", name)
...     print ("Age: ", age)
...     print ("Job: ", job)
... display (age = 33, name = "Garry", job = "Seller")
... display (name = "Joana", age = 25, job = "Teacher")
Name:  Garry
Age:  33
Job:  Seller
Name:  Joana
Age:  25
Job:  Teacher
```

```
>>> def myfunc(*args):  
...     return sum(args) * 2  
...  
>>> myfunc(1, 2, 3, 4, 5)  
30  
>>> myfunc(5, 8, 10)  
46
```

```
>>> def myfunc(**kwargs):  
...     if 'apple' in kwargs:  
...         return(kwargs['apple'])  
...     else:  
...         return('apple')  
... print(myfunc(a = 'banana', b = 'orange'))  
apple
```


Стек вызовов

При работе с большим количеством функций, вызывающих друг друга, полезно знать, как они устроены в техническом плане.

Для работы с функциями python реализует такой механизм, как стек вызовов.

Стек вызовов можно представить как контейнер, на дно которого "падают" функции при их вызове и "снимаются" при их завершении. Главный концепт стека вызовов в том, что в каждый момент времени выполняется верхняя функция в стеке, то есть, последняя вызванная.

```
def f():  
    print('выполняется функция f')  
  
def g():  
    print('выполняется функция g')  
    f()  
    print('функция g продолжает выполнение')  
  
print('вход в программу')  
g()  
print('функции закончили выполнение')  
  
#>>> вход в программу  
#>>> выполняется функция g  
#>>> выполняется функция f  
#>>> функция g продолжает выполнение  
#>>> функции закончили выполнение
```

1. пустой стек

2. вызов `print('вход в программу')`

print()

3. выход из функции `print()`

4. вызов функции `g()`

g()

5. вызов функции `print('выполняется функция g')`

g()

print()

6. выход из функции `print()`

g()

7. вызов функции `f()`

g()

f()

```
def f():  
    print('выполняется функция f')
```

```
def g():  
    print('выполняется функция g')  
    f()  
    print('функция g продолжает')
```

```
print('вход в программу')  
g()  
print('функции закончили выпол')
```

```
#>>> вход в программу  
#>>> выполняется функция g  
#>>> выполняется функция f  
#>>> функция g продолжает вы  
#>>> функции закончили выпол
```

8. вызов функции print('выполняется функция f')

g()

f()

print()

9. выход из функции print

g()

f()

10. выход из функции f()

g()

11. вызов функции print('функция g продолжает выполни

g()

print()

12. выход из функции print()

g()

13. выход из функции g()

14. вызов print('функции закончили выполнение')

print()

15. выход из функции print()

```
def f():  
    print('выполняется функция f')
```

```
def g():  
    print('выполняется функция g'  
    f()  
    print('функция g продолжает')
```

```
print('вход в программу')  
g()  
print('функции закончили выпол
```

```
#>>> вход в программу  
#>>> выполняется функция g  
#>>> выполняется функция f  
#>>> функция g продолжает вы  
#>>> функции закончили выпол
```

Возвращаемые значения, return

Функции в python могут возвращать результат. Для этого используется оператор return, после которого указывается возвращаемое значение.

```
def getSquare(a):  
    return a * a
```

```
number = 3  
square = getSquare(number)  
print(square)  
#>>> 9
```

Распаковка возвращаемых значений

Функции в python могут возвращать сразу несколько значений. Для этого нужно перечислить возвращаемые значения через запятую. Возвращаемым типом будет кортеж, который можно распаковать в переменные.

```
def getCalculations(a):  
    return 2 * a, a * a, a * a * a
```

```
number = 3  
print(getCalculations(number))  
#>>> (6, 9, 27)
```

```
double, square, cube =  
getCalculations(number)  
print(double, square, cube)  
#>>> 6 9 27
```

return как выход из функции

Оператор **return** не только возвращает значение, но и производит выход из функции.

Мы можем использовать это свойство оператора `return` в таких функциях, которые ничего не возвращают. Например, может быть необходимо совершить выход из функции в зависимости от определённого условия:

```
def greet(grade, name):  
    if grade < 1 or grade > 11:  
        print("Ошибка: в нашей школе такого класса нет")  
        return  
    print(f"Привет! Меня зовут {name} и я учусь в {grade} классе")
```

```
greet(9, "Иван")  
#>>> Привет! Меня зовут Иван и я учусь в 9 к
```

```
greet(12, "Сергей")  
#>>> Ошибка: в нашей школе такого класса нет
```

Параметры и аргументы

Позиционные и именованные аргументы

```
def showArgs(a,b,c):  
    print(f"a = {a}, b = {b}, c = {c}")
```

```
showArgs(1,2,3)  
#>>> a = 1, b = 2, c = 3
```

Позиции аргументов 1, 2 и 3 соответствуют позициям параметров a, b и c. Такие аргументы называются **позиционными**.

В python также возможно явно задавать соответствия между аргументами и параметрами:

```
def showArgs(a,b,c):  
    print(f"a = {a}, b = {b}, c = {c}")
```

```
showArgs(c = 1,a = 2,b = 3)  
#>>> a = 2, b = 3, c = 1
```

В этом случае соответствие определяется по именам, а не по позициям аргументов. Такие аргументы называются **именованными**.

В одном вызове функции можно использовать как позиционные, так и именованные аргументы, в этом случае важно обратить внимание на порядок их перечисления: **сначала идут позиционные аргументы, затем - именованные.**

```
def showArgs(a, b, c, d, e):  
    print(f"a = {a}, b = {b}, c = {c}, d = {d}, e = {e}")
```

```
showArgs(1,2,d = 3, e = 4, c = 5)
```

```
#>>> a = 1, b = 2, c = 5, d = 3, e = 4
```

Позиционные и именованные параметры

Именованными и позиционными могут быть не только аргументы, но и параметры. Для этого используются символы **/** и *****.

- Все параметры, которые располагаются справа от символа *****, получают значения только по имени.
- Все параметры, которые располагаются слева от символа **/**, получают значения только по позиции.

```
def showArgs(a, b, /, c, *, d, e):  
    print(f"a = {a}, b = {b}, c = {c}, d = {d}, e = {e}")
```

```
showArgs(1,2,3, e = 4, d = 5)  
#>>> a = 1, b = 2, c = 3, d = 5, e = 4
```

```
showArgs(1,2,e = 3, c = 4, d = 5)  
#>>> a = 1, b = 2, c = 4, d = 5, e = 3
```

В этом примере **a** и **b** - позиционные параметры, **d** и **e** - именованные, а **c** может получать значения как по позиции, так и по имени.

Необязательные параметры

Python позволяет делать отдельные параметры функции необязательными. Если при вызове функции значение такого аргумента не передается, этому параметру будет присвоено значение по умолчанию.

```
def greet(age, name = "никто", city = "нигде"):  
    print(f"Привет! Мне {age} лет, меня зовут {name} и я живу в городе  
{city}")
```

```
greet(20, "Иван", "Москва")
```

```
#>>> Привет! Мне 20 лет, меня зовут Иван и я живу в городе Москва
```

```
greet(27, name = "Сергей", city = "Санкт-Петербург")
```

```
#>>> Привет! Мне 27 лет, меня зовут Сергей и я живу в городе Санкт-  
Петербург
```

```
greet(18)
```

```
#>>> Привет! Мне 18 лет, меня зовут никто и я живу в городе нигде
```

Важно! Если у функции есть как обязательные, так и необязательные параметры, необязательные параметры должны идти после обязательных.

Аргументы переменной длины

В python существует возможность передавать в функции произвольное количество аргументов. Для этого используются аргументы переменной длины.

`*args`

```
def squareAll(*args):  
    print(args)  
    for number in args:  
        print(number * number)
```

```
squareAll(1, 3, 6, 2)
```

```
#>>> (1, 3, 6, 2)
```

```
#>>> 1
```

```
#>>> 9
```

```
#>>> 36
```

```
#>>> 4
```

Функция `squareAll` принимает только один параметр - `*args`, но на его место можно передать произвольное количество аргументов. Переменная `args` будет являться кортежем из переданных на её место аргументов.

****kwargs**

Аналогичным способом можно передать в функцию произвольное количество именованных аргументов. Для этого используется двойной символ ******.

```
def showKwargs(**kwargs):  
    print(kwargs)  
    for key, value in kwargs.items():  
        print(f"ключу {key} соответствует значение  
{value}")
```

```
showKwargs(name = "Иван", city = "Москва",  
age = "22")  
#>>> {'name': 'Иван', 'city': 'Москва', 'age': '22'}  
#>>> ключу name соответствует значение  
Иван  
#>>> ключу city соответствует значение  
Москва  
#>>> ключу age соответствует значение 22
```

Переменная `kwargs` является словарём, где ключ - имя аргумента, а значение - собственно, его значение.

При работе с аргументами переменной длины нужно помнить несколько моментов:

`args` и `kwargs` - это просто слова. Вся логика заключается именно в символах распаковки `*` и `**`.

Тем не менее, для имён таких параметров рекомендуется использовать именно слова `args` и `kwargs` - это стандарт.

Нельзя располагать `**kwargs` до `*args`.

Передача по значению и по ссылке

Python бывают **изменяемыми** и **неизменяемыми**. При передаче в функцию изменяемые и неизменяемые объекты ведут себя по-разному.

В классических языках программирования объекты передаются **по значению** или **по ссылке**.

В python, **всё есть объекты**, и они предаются в функцию только по ссылке.

```
def change(numbers):  
    numbers.append(30)  
    print(id(numbers))  
    print(numbers)
```

```
n = [10, 20]  
change(n)  
#>>> 140186460722240  
#>>> [10, 20, 30]
```

```
print(id(n))  
#>>> 140186460722240
```

```
print(n)  
#>>> [10, 20, 30]
```

n = tuple() - ?

Встроенные функции

Built-in Functions

abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	

<u>abs()</u>	<u>delattr()</u>	<u>hash()</u>	<u>memoryview()</u>	<u>set()</u>
<u>all()</u>	<u>dict()</u>	<u>help()</u>	<u>min()</u>	<u>setattr()</u>
<u>any()</u>	<u>dir()</u>	<u>hex()</u>	<u>next()</u>	<u>slice()</u>
<u>ascii()</u>	<u>divmod()</u>	<u>id()</u>	<u>object()</u>	<u>sorted()</u>
<u>bin()</u>	<u>enumerate()</u>	<u>input()</u>	<u>oct()</u>	<u>staticmethod()</u>
<u>bool()</u>	<u>eval()</u>	<u>int()</u>	<u>open()</u>	<u>str()</u>
<u>breakpoint()</u>	<u>exec()</u>	<u>isinstance()</u>	<u>ord()</u>	<u>sum()</u>
<u>bytearray()</u>	<u>filter()</u>	<u>issubclass()</u>	<u>pow()</u>	<u>super()</u>
<u>bytes()</u>	<u>float()</u>	<u>iter()</u>	<u>print()</u>	<u>tuple()</u>
<u>callable()</u>	<u>format()</u>	<u>len()</u>	<u>property()</u>	<u>type()</u>
<u>chr()</u>	<u>frozenset()</u>	<u>list()</u>	<u>range()</u>	<u>vars()</u>
<u>classmethod()</u>	<u>getattr()</u>	<u>locals()</u>	<u>repr()</u>	<u>zip()</u>
<u>compile()</u>	<u>globals()</u>	<u>map()</u>	<u>reversed()</u>	<u>__import__()</u>
<u>complex()</u>	<u>hasattr()</u>	<u>max()</u>	<u>round()</u>	

Условно, их можно поделить на:

- ввод/вывод информации (print, input, open),
- форматирование (format, ascii, repr),
- преобразование типов (int, list и др.),
- математические (abs, pow и др.),
- готовая обработка последовательностей (min, max, sum, len и т.п.),
- создание последовательностей (iter, next, enumerate, range),
- обработка последовательностей с помощью функций (filter, map, zip)
- служебные (hash, id, hasattr и др.),
- остальные...

Функции ввода и вывода данных

- **print()** вывод информации на экран
- **input()** которая считывает строку из консоли
- **open()/close()**, которая открывает/закрывает файл для чтения или записи.

- **bin()** – возвращает строку с числом преобразованным в двоичную форму
- **hex()** – возвращает строку с числом преобразованным в шестнадцатеричную форму
- **oct()** – возвращает строку с числом преобразованным в восьмеричную форму

- **int()** – возвращает целое число
- **bool()** – возвращает логическую интерпретацию переданных данных
- **float()** – возвращает число с дробной частью
- **complex()** – возвращает комплексное число
- **str()** – возвращает строку
- **list()** – возвращает список
- **tuple()** – возвращает кортеж
- **set()** – возвращает множество
- **frozenset()** – возвращает неизменяемое множество

Функции для обработки последовательностей

round()

```
Python Console - pythonProject
Python Console >
>>> round(5.76, 1)
5.8
>>> round(5.72, 1)
5.7
>>> round(1564, -2)
1600
>>> round(1564, -1)
1560
```

abs() и pow()

```
Python Console - pythonProject
Python Console >
>>> abs(-1)
1
>>> pow(3, 2)
9
```

max() и min()

```
Python Console - pythonProject
Python Console >
>>> max(1, 2, 3, 4, 5) # 5 - максимальный элемент
5
>>> s = ['1', '12', '120', '22']
>>> max(s) # '22' - максимальный в лексикографическом порядке
'22'
>>> max(s, key=len) # '120' - максимальный по длине
'120'
>>> min('c', 'a', 'b') # 'a'
'a'
>>>
```

sum()

```
Python Console - pythonProject
Python Console >
>>> sum([1, 2, 3, 4, 5])
15
```

all()

```
Python Console - pythonProject
Python Console x
>>> def all(iterable):
...     for element in iterable:
...         if not element:
...             return False
...     return True
```

any()

```
Python Console - pythonProject
Python Console x
>>> def any(iterable):
...     for element in iterable:
...         if element:
...             return True
...     return False
```

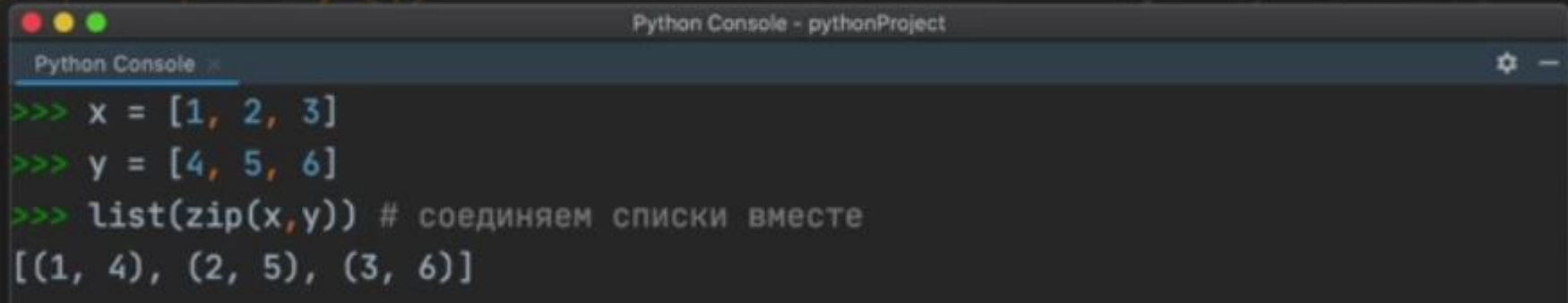
```
>>> s = [1, 3, 5]
>>> print(any(element % 3 == 0 for element in s))
True
```

enumerate()

```
Python Console - pythonProject
Python Console x
>>> lst = ['a', 'b', 'c']
>>> for number, item in enumerate(lst):
...     print(number, item)
...
0 a
1 b
2 c
```

```
Python Console - pythonProject
Python Console x
>>> months = ['March', 'April', 'May', 'June']
>>> print(list(enumerate(months, start=3)))
[(3, 'March'), (4, 'April'), (5, 'May'), (6, 'June')]
```

zip()



```
Python Console - pythonProject
Python Console
>>> x = [1, 2, 3]
>>> y = [4, 5, 6]
>>> list(zip(x,y)) # соединяем списки вместе
[(1, 4), (2, 5), (3, 6)]
```

```
>>> category = ['name', 'last_name', 'age', 'job']
>>> value = ['John', 'Smith', '35', 'Seller']
>>> person = dict(zip(category, value))
>>> print(person)
{'name': 'John', 'last_name': 'Smith', 'age': '35', 'job': 'Seller'}
```


globals() и locals()

Python Console - pythonProject

```
Python Console x
>>> x = 10
>>> y = 100
>>> def my_function():
...     m = 13
...     n = 27
...     print(locals())
... my_function()
{'m': 13, 'n': 27}
```

```
...     print(globals())
... my_function()
{'__name__': '__main__', '__doc__': None, '__package__': None, '__loader__':
<_frozen_importlib_external.SourceFileLoader object at 0x100a43e80>,
 '__spec__': None, '__file__': '<input>', '__builtins__': {'__name__':
'builtins', '__doc__': "Built-in functions, exceptions, and other objects
|
```


Служебные функции

- **id(object)** - возвращает id объекта object.
- **hash(object)** - возвращает хэш-значение объекта object.
- **hasattr(object, name)** - проверяет существование атрибута с именем name в объекте object.