

Функции

Функциональное

программирование

<https://compscicenter.ru/courses/python/2012-autumn/classes/667/>

Функции

Ключевое слово def

```
>>> def empty_func():  
...     pass
```

Функция без return возвращает None

```
>>> print empty_func()  
None
```

Функции

```
>>> def gcd(a, b):  
...     """Greatest Common Divisor"""    #docstring  
...     while a != 0:  
...         a, b = b%a, a  
...     return b  
...  
>>> print gcd.__doc__
```

Функция = объект

```
>>> gcd
```

```
<function gcd at 10042e12>
```

```
>>> new_funcbon = gcd
```

```
>>> print new_funcbon(14, 7)
```

```
7
```

Передача параметров

```
>>> def magic(v):  
...     v.append("Blue")  
...  
>>> my_list = ["Red", "Green"]  
>>> magic(my_list)  
>>> print my_list  
['Red', 'Green', 'Blue']
```

Передача параметров

```
>>> my_list = ["Red", "Green"]
>>> def magic2(v):
...     v = ["Hue", "Saturabon", "Value"]
...
>>> magic2(my_list)
>>> print my_list
['Red', 'Green']
```

Использование кортежей

```
>>> def mulbout():  
...     return 1, 2, 3  
...  
>>> print mulbout()  
(1, 2, 3)
```

Области видимости

- Что такое область видимости?
- Сколько существует в каждый момент выполнения программы?

Области видимости

В каждый момент существует 3 области видимости:

- Локальная
- Средняя (глобальные имена модуля)
- Внешняя (встроенные имена)

Глобальным переменным невозможно прямо присвоить значения внутри функций, хотя ссылки на них могут использоваться.

Параметры по-умолчанию

```
>>> def greet(adr="mr.", name="X"):
...     print "Hello" + adr + name + "!"
...
```

```
>>> greet("mrs.", "Anderson")           #Обычный вызов
Hello mrs.Anderson!
```

```
>> greet(name="Gates")                   #именованный параметр
Hello mr.Gates!
```

```
>> greet()                               #с параметрами по-умолчанию
Hello mr.X!
```

Параметры по-умолчанию

Значения по умолчанию вычисляются в точке определения функции

```
>>> i = 5
```

```
>>> def double_print(arg1, arg2=i):
```

```
...     print arg1, arg2
```

```
>>> i = 6
```

```
>>> double_print (1)
```

```
1 5
```

```
>>> double_print(1, 2)
```

```
1 2
```

```
>>> double_print (arg2=1, 5)
```

```
SyntaxError: non-keyword arg after keyword arg
```

Параметры по-умолчанию

Значение по умолчанию вычисляется лишь единожды.

```
>>> def list_funcbon(a, my_list=[]):
```

```
...     my_list.append(a)
```

```
...     return my_list
```

```
...
```

```
>>> print list_funcbon(1)
```

```
[1]
```

```
>>> print list_funcbon(2)
```

```
[1, 2]
```

***args**

```
>>> def avg(*args):  
...     sum = 0.0  
...     for arg in args:  
...         sum += arg  
...     return sum/len(args)  
...  
>> avg(1, 2)  
1.5  
>> avg(3, 5, 2)  
3.3333333333333335
```

****kwargs**

```
>>> def foo_kwargs(farg, **kwargs):  
...     print "formal arg:", farg  
...     for key in kwargs:  
...         print "keyword arg: %s: %s" % (key, kwargs[key])  
>>> foo_kwargs(farg=1, myarg2="two", myarg3=3)  
formal arg: 1  
keyword arg: myarg2: two  
keyword arg: myarg3: 3
```

Вызов с распаковкой

Вызов с распаковкой

```
>>> range(3, 6)          # вызов с отдельными аргументами
```

```
[3, 4, 5]
```

```
>>> args = [3, 6]
```

```
>>> range(*args)         # с распакованными аргументами
```

```
[3, 4, 5]
```

Аналогично для **-оператора.

Лямбда-функции

Лямбда-функция – неименованная функция.

Можно вызывать в месте определения:

```
>>> (lambda x: x*x)(5)
```

```
25
```

```
>>> foo = lambda x: x*x
```

```
>>> print foo(7)
```

```
49
```


Замыкания

Что такое замыкание?

Замыкания

Функция, которая ссылается на свободные переменные в своем лексическом контексте.

```
>>> def make_adder(x):  
...     def adder(n):  
...         return x + n      #захват 'x' из внешнего контекста  
...     return adder  
>>> adder = make_adder(10)  
>>> print adder(5)  
15
```

Функции высших порядков

Функции, принимающие в качестве аргументов другие функции или возвращающие другие функции в качестве результата.

Функции высших порядков

`filter`(функция, последовательность) – возвращает последовательность, состоящую из тех элементов последовательности, для которых функция является истиной.

```
>>> list = [10, 4, 2, -1, 6]
```

```
>>> filter(lambda x: x < 5, list)
```

```
[4, 2, -1]
```

Функции высших порядков

`map(функция, последовательность)` – совершает вызов `функция(элемент)` с каждым элементом последовательности и возвращает список из возвращавшихся функцией значений.

```
>>> list1 = [7,2, 3, 10, 12]
```

```
>>> list2 = [-1, 1, -5, 4, 6]
```

```
>>> map(lambda x, y: x*y, list1, list2)
```

```
[-7, 2, -15, 40, 72]
```

Функции высших порядков

`reduce(функция, последовательность)` – возвращает единственное значение, собранное из результатов вызовов двухаргументной функции с первыми двумя элементами последовательности, затем с полученным результатом и последующим элементом

```
>>> list = [2, 3, 4, 5, 6]
```

```
>>> reduce(lambda res, x: res*x, list, 1)
```

```
720
```

Порядок вычислений:

$$((((1*2)*3)*4)*5)*6$$