



Функции

Часть II



Лямбда-функции, анонимные функции

Раньше мы использовали функции, обязательно связывая их с каким-то именем. В Python есть возможность создания однострочных анонимных функций

Конструкция:

```
lambda [param1, param2, ..]: [выражение]
```

lambda – функция, возвращает свое значение в том месте, в котором вы его объявляете.



Лямбда-функция

функция, которая возвращает свой параметр

```
def identity(x):
```

```
    return x
```

```
identity(100)
```

#identity() функция тождества принимает передаваемый аргумент в x и возвращает его при вызове.

Лямбда-функция :

```
lambda x: x
```

Ключевое слово: lambda

Параметр: x

Выражение (тело) : x



Вызов lambda

Для вызова lambda обернем функцию и ее аргумент в круглые скобки. Передадим функции аргумент.

```
>>> (lambda x: x + 1) (2)
```

```
3
```



Именованное **lambda**

Поскольку лямбда-функция является выражением, оно может быть именовано. Поэтому вы можете написать предыдущий код следующим образом:

```
>>> add_one = lambda x: x + 1
```

```
>>> add_one(2)
```

```
3
```



Аргументы

Как и обычный объект функции, определенный с помощью `def`, лямбда поддерживают все различные способы передачи аргументов. Это включает:

- Позиционные аргументы
- Именованные аргументы (иногда называемые ключевыми аргументами)
- Переменный список аргументов (часто называемый `*args`)
- Переменный список аргументов ключевых слов `*kwargs`



Аргументы функции

Функции с несколькими аргументами (функции, которые принимают более одного аргумента) выражаются в лямбда-выражениях Python, перечисляя аргументы и разделяя их запятой (,), но не заключая их в круглые скобки:

```
>>> full_name = lambda first, last: f'Full name: {first.title()} {last.title()}'
```

```
>>> full_name('guido', 'van rossum')  
'Full name: Guido Van Rossum'
```



Именованные параметры

Как и в случае **def**, для аргументов **lambda** можно указывать стандартные значения.

```
>>>str = ( lambda a='He', b='ll' , c='o': a+b+c)
str(a='Ze')
'Zello'
```




Пример

```
>>> (lambda x, y, z: x + y + z) (1, 2, 3)
```

```
6
```

```
>>> (lambda x, y, z=3: x + y + z) (1, 2)
```

```
6
```

```
>>> (lambda x, y, z=3: x + y + z) (1, y=2)
```

```
6
```

```
>>> (lambda *args: sum(args)) (1, 2, 3)
```

```
6
```

```
>>> (lambda **kwargs: sum(kwargs.values())) (one=1,  
two=2, three=3)
```

```
6
```

```
>>> (lambda x, *, y=0, z=0: x + y + z) (1, y=2, z=3)
```



Функции высокого порядка

Лямбда-функция может быть функцией более высокого порядка, принимая функцию (нормальную или лямбда-функцию) в качестве аргумента, как в следующем надуманном примере:

```
>>> high_ord_func = lambda x, func: x + func(x)
```

```
>>> high_ord_func(2, lambda x: x * x)
```


```
6
```

```
>>> high_ord_func(2, lambda x: x + 3)
```

```
7
```



Для чего используется lambda ?



Использование лямбда-выражения как литерала списка.

```
import random

# Словарь, в котором формируются три случайные числа
# с помощью лямбда-выражения
L = [ lambda : random.random(),
      lambda : random.random(),
      lambda : random.random() ]

# Вывести результат
for l in L:
    print(l())
```



Лямбда-выражения как литералы кортежей

Формируется кортеж, в котором элементы умножаются на разные числа.

```
import random
# Кортеж, в котором формируются три литерала-строки
# с помощью лямбда-выражения
T = ( lambda x: x*2,
      lambda x: x*3,
      lambda x: x*4 )
# Вывести результат для строки 'abc'
for t in T:
    print(t('abc'))
```

Результат:

abcaabc

abcaabcaabc

abcaabcaabcaabc

Использование лямбда-выражения для формирования таблиц переходов.

Словарь, который есть таблицей переходов

```
Dict = {  
    1 : (lambda: print('Monday')),  
    2 : (lambda: print('Tuesday')),  
    3 : (lambda: print('Wednesday')),  
    4 : (lambda: print('Thursday')),  
    5 : (lambda: print('Friday')),  
    6 : (lambda: print('Saturday')),  
    7 : (lambda: print('Sunday'))  
}
```

Вызвать лямбда-выражение, выводящее название вторника

```
Dict[2]() # Tuesday
```




Таблица переходов , в которой вычисляется площадь известных фигур.

```
import math
```

```
Area = {  
    'Circle' : (lambda r: math.pi*r*r), # окружность  
    'Rectangle' : (lambda a, b: a*b), # прямоугольник  
    'Trapezoid' : (lambda a, b, h: (a+b)*h/2.0) # трапеция  
}
```

```
# Вызвать лямбда-выражение, которое выводит площадь окружности  
радиуса 2
```

```
print('Area of circle = ', Area['Circle'](2))
```


```
# Вывести площадь прямоугольника размером 10*13
```

```
print('Area of rectangle = ', Area['Rectangle'](10, 13))
```

```
# Вывести площадь трапеции для a=7, b=5, h=3
```

```
areaTrap = Area['Trapezoid'](7, 5, 3)
```

```
print('Area of trapezoid = ', areaTrap)
```



Совместное использование lambda-функции со встроенными функциями

Функция **filter()** принимает два параметра — функцию и список для обработки. В примере мы применим функцию `list()`, чтобы преобразовать объект `filter` в список.

Пример 1

```
numbers=[0,1,2,3,4,5,6,7,8,9,10]
```

```
list(filter(lambda x:x%3==0,numbers))
```

```
[0, 3, 6, 9]
```

Код берет список `numbers`, и отфильтровывает все элементы из него, которые не делятся нацело на 3. При этом фильтрация никак не изменяет изначальный список.



filter()

Пример 2

```
even = lambda x: x%2 == 0
```

```
list(filter(even, range(11)))
```

```
[0, 2, 4, 6, 8, 10]
```

Обратите внимание, что `filter()` возвращает итератор, поэтому необходимо вызывать `list`, который создает список с заданным итератором.

Реализация, использующая конструкцию генератора списка, дает одинаковый результат:

```
>>> [x for x in range(11) if x%2 == 0]
```

```
[0, 2, 4, 6, 8, 10]
```



Функция **map()**

Функция **map()** в отличие от функции **filter()** возвращает значение выражения для каждого элемента в списке.

#Пример 1

```
numbers=[0,1,2,3,4,5,6,7,8,9,10]
```

```
list(map(lambda x:x%3==0,numbers))
```

```
[True, False, False, True, False, False, True, False,  
False, True, False]
```

#Пример 2

```
list(map(lambda x: x.capitalize(), ['cat', 'dog',  
'cow']))
```

```
['Cat', 'Dog', 'Cow']
```



reduce()

функция `reduce()` принимает два параметра — функцию и список. Сперва она применяет стоящую первым аргументом функцию для двух начальных элементов списка, а затем использует в качестве аргументов этой функции полученное значение вместе со следующим элементом списка и так до тех пор, пока весь список не будет пройден, а итоговое значение не будет возвращено. Для того, чтобы использовать `reduce()`, вы должны сначала импортировать ее из модуля `functools`.



reduce()

Пример

```
from functools import reduce
numbers=[0,1,2,3,4,5,6,7,8,9,10]
reduce(lambda x,y:y-x,numbers)
5
```

$$1-0=1$$

$$2-1=1$$

$$3-1=2$$

$$4-2=2$$

$$5-2=3$$


$$6-3=3$$

$$7-3=4$$

$$8-4=4$$

$$9-4=5$$

$$10-5=5$$



Можно ли в лямбда-выражениях
использовать стандартные операторы
управления **if, for, while**?

НЕТ



НО !

Можно использовать тернарный оператор.

```
lower = (lambda x, y: x if x < y else y)
```

```
# Вызов 1 способ
```

```
(lambda x, y: x if x < y else y) (10, 3)
```

```
# Вызов 2 способ
```

```
lower(10, 3)
```

```
3
```



Заключение

- Избегать чрезмерного использования лямбд
- Использовать лямбды с функциями высшего порядка или ключевыми функциями Python
- Функции более высокого порядка, такие как `map()`, `filter()` и `functools.reduce()`, могут быть преобразованы в более элегантные формы с небольшими изменениями, в частности, со списком или генератором выражений.