UMA INTRODUÇÃO A

Programação funcional com Python

Eduardo Mendes
github.com/z4r4tu5tr4

Estrutura

1. Whoami
2. Funções como objeto
3. Diferenças entre geradores e iteradores
4. Funções Lambda
5. Biblioteca Operator
6. Funções Map, Reduce, Filter e Sum

@Babbage# Whoami

. Eduardo Mendes
. Fatec Americana (talvez?)
. github.com/z4r4tu5tr4
. mendesxeduardo@gmail.com

Funções como objeto (Ramalho)

. Criadas em tempo de execução;
. Atribuída a uma variável ou a um elemento em uma estrutura de dados;

. passada como argumento a uma função;
. devolvida como resultado de uma função.

Iteradores x Geradores

```
>> lista = [1,2,3,4,5]
>> for elemento in lista:
...     print(elemento)
1
2
3
4
5
```

```
>> lista_0 = [1,2,3,4,5]
>> lista_1 = [1,2,3,4,5]
>> zipper = zip(lista_0,
                lista_1)

>> for elemento in zipper:
...     print(elemento)
(1, 1)
(2, 2)
(3, 3)
(4, 4)
(5, 5)
```

Outra vez

Iteradores x Geradores

```
>> for elemento in lista:
...     print(elemento)
1
2
3
4
5
```

```
>> for elemento in zipper:
...     print(elemento)
```

Iteradores x Geradores

```
>> for elemento in lista:
...     print(elemento)
1
2
3
4
5
```

```
>> for elemento in zipper:
...     print(elemento)
```

??????????

Lambda x def

```
>> soma = lambda x,y: x+y
>> resultado = soma(5,5)
>> print(resultado)
10
```

```
>> def soma(x,y)
...     return x+y
>> resultado = soma(5,5)
>> print(resultado)
10
```

```
>> mult = lambda x,y: x*y
>> print(mult(5,5))
25
```

```
>> def mult(x,y)
...     return x+y
>> print(mult(5,5))
25
```

Lambda x def

```
>> s
>> f
>> f
10
```

```
>> mult = lam  y: x*y
>> print(mult(5,5))
25
```

```
>> def mu
...     retu  x+y
>> print(mult(5,5))
25
```

Função como objeto

from operator import *

```
add(5,10)      #15
```

5 + 10

```
eq(5,5)        #True
```

5 == 5

```
le(“c”,”cpp”)#True
```

“sp” <= “spp”

```
mult(5,5)      #25
```

5 * 5

```
pow(2,2)       #4
```

2 ** 2

Map

```
>> lista = [1,2,3,4,5]
>> n_lista = map(lambda
          x: x**2, lista)

>> type(n_lista)
#map

>> list(n_lista)    #
[1,4,9,16,25]
```

Filter

```
>> lista = [1,2,3,4,5]
>> n_lista = filter
(lambda x: x != 2,
lista)

>> type(n_lista)
#filter

>> list(n_lista)    #
[1,3,4,5]
```

Reduce | Sum

```
>> lista = [1,2,3,4,5]
>> num = reduce(add,
                lista)

>> type(num)
#int

>> print(num)
#15
```

```
>> lista = [1,2,3,4,5]
>> num = sum(lista)

>> type(num)
#int

>> print(num)
#15
```

Reduce | Sum

```
>> lista = [1,2,3,4,5]
>> num = reduce(add,
                lista)

>> type(num
#int

>> print(nu
#15
```

Trabalha com qualquer operador

```
>> lista = [1,2,3,4,5]
>> num = sum(lista)

>> type(num)
#int

>> print(num)
#15
```

Somente soma

Lambda + Map

```
>> lista = [1,2,3,4,5]
>> n_lista = map(lambda x: x**2, lista)
>> list(n_lista)
#[1,4,9,16,25]
```

List comprehension

```
>>[ x**2 for x in [1,2,3,4,5] ]
#[1,4,9,16,25]
```

Map + Sum + filter

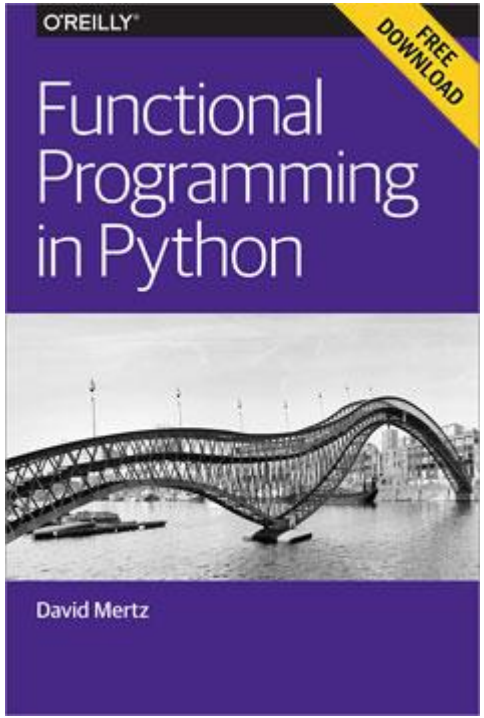
```
>> lista = [1,2,3,4,5]
>> sum(filter(lambda x: x > 4, map(lambda x: x**2, lista)))
#50
```

Compreensão de Lista + Sum

```
>> sum([x**2 for x in [1,2,3,4,5] if x**2 > 4])
#50
```

Para saber mais

watch?v=Rp_lvuXuSSA



xoxo!

mendesxeduardo@gmail.com
github.com/z4r4tu5tr4