Aula 2

1 FUNÇÕES LAMBDA

São funções que são utilizadas apenas em um contexto especifico, e que serão usadas apenas algumas vezes. Assim em python as funções lambda que facilitam a redução do tamanho de código e tornam o código mais simples

Exemplo uma função que eleve um número ao quadrado

[4]:
```
def quadrado(var):
    return var**2
```

[5]:
```
quadrado(2)
```

[5]: 4

Posso reduzir nosso código da seguinte forma:

[6]:
```
def quadrado(var): return var**2
```

[7]:
```
quadrado(2)
```

[7]: 4

[9]:
```
lambda var: var**2
```

[9]: <function __main__.<lambda>(var)>

No caso da função lambda eu retiro o nome da função pois não vamos precisar, visto que ela só existe na linha de código que foi chamada e não precisamos do return também.

2 MAP E FILTER

Map função que atribiu os valores de um interável e passa ele como parametros de uma função

[15]:
```
seq=[1,2,3,4,5]
```

[16]:
```
map(quadrado, seq)
```

[16]: <map at 0x7fe20c469690>

[17]:
```
type(map(quadrado, seq))
```

[17]:
```
map
```

Para ver os valores do map preciso transformar ele em uma lista.

[18]:
```
list(map(quadrado, seq))
```

[18]:
```
[1, 4, 9, 16, 25]
```

Se quiser realizar o calculo apenas uma vez, posso usar uma função lambda por exemplo:

[20]:
```
list(map(lambda x:x**2, seq))
```

[20]:
```
[1, 4, 9, 16, 25]
```

3 FILTER

Filter permite filtrar elementos de uma lista ou de um iterável, dado que a condição de uma função seja verdadeira.

[21]:
```
list(filter(lambda item:item%2==0, seq))
```

[21]:
```
[2, 4]
```

A função lambda busca na lista seq os valores que respeitam o %2==0, ou seja ela busca os elementos que são pares da lista.

[22]:
```
list(filter(lambda item:item%2!=0, seq))
```

[22]:
```
[1, 3, 5]
```

Nesse caso ele buscou os elementos que são impares

4 MÉTODOS

Objetos são instâncias de uma classe, e cada objeto tem um conjunto de métodos associado aquele objeto.

Por exemplo quando temos um objeto do tipo string, temos uma série de funções que existem dentro daquela classe que são os métodos da classe string.

Nesse contexto Métodos e funções são a mesma coisa.

[23]:
```
str='Olá, meu nome é Rodrigo'
```

[24]:
```
type(str)
```

[24]:
```
str
```

[25]:
```
str.lower()
```

[25]: 'olá, meu nome é rodrigo'

[26]:
```
str.upper()
```

[26]: 'OLÁ, MEU NOME É RODRIGO'

O método split pode separar uma string em diversos pedaços

[27]:
```
str.split()
```

[27]: ['Olá,', 'meu', 'nome', 'é', 'Rodrigo']

[28]:
```
str.split(',')
```

[28]: ['Olá', ' meu nome é Rodrigo']

[32]:
```
str.split('n')
```

[32]: ['Olá, meu ', 'ome é Rodrigo']

Se observar o tamanho por meio do método len é possível notar que os dois tem tamanhos diferentes.

[29]:
```
len(str.split())
```

[29]: 5

[30]:
```
len(str.split(','))
```

[30]: 2

Outro método interessante que pode ser usado em listas é o método append, que inseri um novo elemento na lista.

[33]:
```
list=[1, 2, 3]
```

[34]:
```
list
```

[34]: [1, 2, 3]

[35]:
```
list.append(4)
```

[36]:
```
list
```

[36]: [1, 2, 3, 4]

Método pop() se não passar nenhum parametro ele vai retirar o ultimo elemento da lista.

[37]:
```
last=list.pop()
```

[38]:
```
last
```

[38]:
```
4
```

[39]:
```
list
```

[39]:
```
[1, 2, 3]
```

Posso tbm indicar o indice do elemento que quero retirar da lista

[40]:
```
first=list.pop(0)
```

[41]:
```
first
```

[41]:
```
1
```

[42]:
```
list
```

[42]:
```
[2, 3]
```

Com um TAB

Operador in checa se um dado elemento está presente na lista, indicando se é verdade ou não.

[43]:
```
'x' in [1, 2, 3]
```

[43]:
```
False
```

[44]:
```
'x' in ['x', 'y', 'z', 'w']
```

[44]:
```
True
```

[45]:
```
'z' in ['x', 'y', 'z', 'w']
```

[45]:
```
True
```

[ ]:
```

```