Exercicios

October 18, 2018

1 Exercício 1

Use slice para remover o último elemento de uma lista

```
In [1]: a = [1, 2, 3, 4, 5]

In [2]: a[:-1]

Out[2]: [1, 2, 3, 4]
```

2 Exercício 2

Use slice para fazer uma cópia de uma lista

```
In [3]: a[:]

Out[3]: [1, 2, 3, 4, 5]

In [4]: id(a)

Out[4]: 2349274242120

In [5]: id(a[:])

Out[5]: 2349274928520
```

3 Exercício 3

Use if expressions para reescrever:

```
x = 5
if x < 5:
    y = -1
elif x == 5:
    y = 0
else:
    y = 1
```

```
In [6]: x = 5
        if x < 5:
            y = -1
        elif x == 5:
            y = 0
        else:
            y = 1
        y

Out[6]: 0

In [7]: y = -1 if x < -5 else 0 if x == 5 else 1

In [8]: y

Out[8]: 0
```

4 Dúvida: Qual é a diferença entre *args e **kwargs?

```
In [9]: def f(*args, **kwargs):
            print("args", args)
            print("kwargs", kwargs)
        f(1, 2, 3, a=4, b=5, c=6)

args (1, 2, 3)
kwargs {'a': 4, 'b': 5, 'c': 6}
```

5 Exercício 4

Faça uma função anônima que faça o mesmo que a seguinte função:

```
def f(a, b):
    c = a ** 2 + b ** 2
    return c ** 0.5

In [10]: def f(a, b):
             c = a ** 2 + b ** 2
             return c ** 0.5
         f(3, 4)

Out[10]: 5.0

In [11]: def f(a, b):
             return (lambda c: c ** 0.5)(a ** 2 + b ** 2)
         f(3, 4)

Out[11]: 5.0
```

```
In [12]: f = lambda a, b: (lambda c: c ** 0.5)(a ** 2 + b ** 2)
         f(3, 4)

Out[12]: 5.0

In [13]: f = lambda a, b: (a ** 2 + b ** 2)** 0.5
         f(3, 4)

Out[13]: 5.0
```

6 Exercício 5

Faça uma função que calcule fatorial usando funções anônimas

```
In [14]: def fat(x):
             if x <= 1:
                 return 1
             return x * fat(x - 1)
         fat(3)

Out[14]: 6

In [15]: def fat(x):
             return 1 if x <= 1 else x * fat(x - 1)
         fat(3)

Out[15]: 6

In [16]: fat = lambda x: 1 if x <= 1 else x * fat(x - 1)
         fat(3)

Out[16]: 6

In [17]: del fat

In [18]: (lambda x: 1 if x <= 1 else x * fat(x - 1))(3)

         ---------------------------------------------------------------------------

         NameError                                 Traceback (most recent call last)

         <ipython-input-18-cda7681feb4f> in <module>()
         ----> 1 (lambda x: 1 if x <= 1 else x * fat(x - 1))(3)


         <ipython-input-18-cda7681feb4f> in <lambda>(x)
         ----> 1 (lambda x: 1 if x <= 1 else x * fat(x - 1))(3)


         NameError: name 'fat' is not defined
```

```
In [19]: (lambda fn: fn(fn, 3))(
             (lambda fat, x: 1 if x <= 1 else x * fat(fat, x - 1))
         )

Out[19]: 6

In [20]: lambda m: (lambda fn: fn(fn, m))(
             (lambda fat, x: 1 if x <= 1 else x * fat(fat, x - 1))
         )

Out[20]: <function __main__.<lambda>>

In [21]: (lambda m: (lambda fn: fn(fn, m))(
             (lambda fat, x: 1 if x <= 1 else x * fat(fat, x - 1))
         ))(3)

Out[21]: 6

In [22]: Y=lambda f:(lambda x:x(x))(lambda y: f(lambda *args: y(y)(*args)))
         func = lambda fat: (lambda x: 1 if x <= 1 else x * fat(x - 1))
         Y(func)(3)

Out[22]: 6

In [23]: (lambda f:(lambda x:x(x))(lambda y: f(lambda *args: y(y)(*args))))(lambda fat: (lambda

Out[23]: 6
```

7 Exercício 6

Faça um dicionário a partir de duas listas de forma que o elemento de uma seja a chave e o elemento da outra seja o valor. Exemplo:

```
l1 = [1, 2, 4, 8]; l2 = ["a", "b", "c", "d"]
esperado == {1: "a", 2: "b", 4: "c", 8: "d"}
```

```
In [24]: l1 = [1, 2, 4, 8]
         l2 = ["a", "b", "c", "d"]

         #esperado == {1: "a", 2: "b", 4: "c", 8: "d"}

In [25]: {v1: v2 for v1, v2 in zip(l1, l2)}

Out[25]: {1: 'a', 2: 'b', 4: 'c', 8: 'd'}
```

8 Exercício 7

Pegue as letras impares do albeto. Dica:

```
from string import ascii_lowercase
```

```
In [26]: from string import ascii_lowercase
         ascii_lowercase
```

```
Out[26]: 'abcdefghijklmnopqrstuvwxyz'
```

```
In [27]: [letra
          for index, letra in enumerate(ascii_lowercase)
          if index % 2 == 0
         ]
```

```
Out[27]: ['a', 'c', 'e', 'g', 'i', 'k', 'm', 'o', 'q', 's', 'u', 'w', 'y']
```

[letra for letra in ascii_lowercase if ord(letra) % 2 == 1]

```
In [28]: [ord(letra) for letra in ascii_lowercase]
```

```
Out[28]: [97,
          98,
          99,
          100,
          101,
          102,
          103,
          104,
          105,
          106,
          107,
          108,
          109,
          110,
          111,
          112,
          113,
          114,
          115,
          116,
          117,
          118,
          119,
          120,
          121,
          122]
```

9 Exercício 8

Transforme uma lista de listas em uma lista só com todos elementos. Exemplo:

```
antes = [[1, 8, 3], [2, 4, 6], [7, 3, 1]]
esperado == [1, 8, 3, 2, 4, 6, 7, 3, 1]
```

```
In [29]: antes = [[1, 8, 3], [2, 4, 6], [7, 3, 1]]
         #esperado == [1, 8, 3, 2, 4, 6, 7, 3, 1]

In [30]: from functools import reduce
         def junta(acumulador, elemento):
             return acumulador + elemento
         reduce(junta, antes)

Out[30]: [1, 8, 3, 2, 4, 6, 7, 3, 1]

In [31]: __import__('functools').reduce(
             lambda a, e: a + e,
             antes
         )

Out[31]: [1, 8, 3, 2, 4, 6, 7, 3, 1]

In [32]: from itertools import chain
         list(chain(*antes))

Out[32]: [1, 8, 3, 2, 4, 6, 7, 3, 1]

In [33]: list(chain.from_iterable(antes))

Out[33]: [1, 8, 3, 2, 4, 6, 7, 3, 1]
```

10 Exercício 9

Crie uma função que acesse a n-ésima posição de um gerador

```
In [34]: gen = (x + 2 for x in [1, 2, 4, 8])
         #access((x + 2 for x in [1, 2, 4, 8]), 3) == 10

In [35]: from itertools import islice
         def access(ger, index):
             return next(islice(ger, index, index + 1))
         access((x + 2 for x in [1, 2, 4, 8]), 3)

Out[35]: 10
```

11 Exercício 10

Faça um gerador para a sequencia de Fibonacci

```
In [36]: def fib():
             a, b = 0, 1
             while True:
                 yield a
                 a, b = b, a + b

         list(zip(fib(), range(10)))
```

```
Out[36]: [(0, 0),
          (1, 1),
          (1, 2),
          (2, 3),
          (3, 4),
          (5, 5),
          (8, 6),
          (13, 7),
          (21, 8),
          (34, 9)]
```

12 Exercício 11

Substitua o gerador por um feito com accumulate

```
In [37]: from itertools import accumulate, repeat
         from collections import namedtuple

         State = namedtuple("State", "a b")

         fib_state = accumulate(
             repeat(State(a=0, b=1)),
             lambda state, _: State(
                 a=state.b,
                 b=state.a + state.b
             ),
         )

         fib = (x[0] for x in fib_state)

         list(zip(fib, range(10)))
```

```
Out[37]: [(0, 0),
          (1, 1),
          (1, 2),
          (2, 3),
```

```
          (3, 4),
          (5, 5),
          (8, 6),
          (13, 7),
          (21, 8),
          (34, 9)]
```

13 Exercício 12

Obtenha todas as permutações "ABCD" que tenham "D" como último elemento

```
In [38]: from itertools import permutations
         [x for x in permutations("ABCD") if x[-1] == "D"]

Out[38]: [('A', 'B', 'C', 'D'),
          ('A', 'C', 'B', 'D'),
          ('B', 'A', 'C', 'D'),
          ('B', 'C', 'A', 'D'),
          ('C', 'A', 'B', 'D'),
          ('C', 'B', 'A', 'D')]

In [ ]:
```