Notas de Aula #11: Listas (Funções e Métodos)

Uma lista (*list*) é uma coleção heterogênea de valores onde cada elemento (ou item) dessa coleção pode ser acessado através de um índice (posição do elemento na coleção) onde a primeira posição é 0 (zero).

Listas são similares a *strings*, que são uma sequência de caracteres, no entanto, diferentemente de *strings*, os itens de uma lista podem ser de tipos diferentes.

Na linguagem Python, a lista é delimitada por colchetes (**[ ]**) e cada item é separado por vírgulas.

Vejamos alguns exemplos a seguir:

```python
# Lista contendo apenas valores strings
lista_1 = ['Cachorro', 'Gato', 'Periquito']

# Lista contendo apenas valores inteiros
lista_2 = [5, 29, 4, -80]

# Lista contendo valores de tipos de variáveis diferentes
lista_3 = ['Carro', 80, 25.4, True]
```

Nesses exemplos, caso se deseje acessar um determinado elemento, fazemos referência a posição (índice) do elemento na lista. Vejamos o exemplo a seguir:

```python
# Lista contendo apenas valores strings
lista_1 = ['Cachorro', 'Gato', 'Periquito']

# Lista contendo apenas valores inteiros
lista_2 = [5, 29, 4, -80]

# Lista contendo valores de tipos de variáveis diferentes
lista_3 = ['Carro', 80, 25.4, True]

# Imprimindo todo o conteúdo da varíavel LISTA_1
print(lista_1)

# Imprimindo apenas o elemento de índice 2 da variável LISTA_2
print(lista_2[2])

# Imprimindo os elemento de índice 1 até o índice 3 (exceto) da variável LISTA_3
print(lista_3[1:3])
```

Podemos observar que na linha 11 não referenciamos nenhuma posição (índice) da lista_1, dessa forma será exibida a lista completa (['Cachorro', 'Gato', 'Periquito'])

Entretanto, nas linhas 14 e 17 foram referenciadas as posições (índices) que se desejam imprimir, nesse caso a linha 11 fará com que seja exibido o valor armazenado na posição 2 de lista_2 (4) e na linha 12 as posições de 1 a 3 (exceto) de lista_3 (80, 25,4).

E quando referenciamos uma posição que não existe na lista? Vejamos o exemplo a seguir:

```
1 # Lista contendo apenas valores strings
2 lista_1 = ['Cachorro', 'Gato', 'Periquito']
3
4 # Lista contendo apenas valores inteiros
5 lista_2 = [5, 29, 4, -80]
6
7 # Lista contendo valores de tipos de variáveis diferentes
8 lista_3 = ['Carro', 80, 25.4, True]
9
10 # Imprimindo a posição 4 de  LISTA_1
11 print(lista_1[4])
```

Na linha 11 estamos referenciando a posição 4 de lista_1. Sendo que essa lista só possui 3 posições (0 a 2). Nesse caso será disparara uma exceção[1] conforme pode ser visto abaixo.

```
Traceback (most recent call last):
  File "c:\Users\charl\OneDrive\Área de Trabalho\exemplo_00.py", line 11, in <module>
    print(lista_1[4])
IndexError: list index out of range
```

[1] Iremos estudar o conceito e como tratar as exceções em aulas futuras.

Um elemento de uma lista também pode ser uma lista. Nesse caso, quanto temos uma lista como elemento de uma outra lista, a lista mais externa é dita aninhada (*nested*) e a lista mais interna é chamada frequentemente de sub-lista (*sublist*). Vejamos o exemplo a seguir:

```
lista_estados = [
                    ['BA', 'Salvador'    , 2872000],
                    ['RN', 'Natal'       ,  884000],
                    ['CE', 'Fortaleza'   , 2669000],
                    ['PE', 'Recife'      , 1645000],
                    ['MA', 'São Luiz'    , 1101000],
                    ['AL', 'Maceió'      , 1015000],
                    ['PI', 'Teresina'    ,  864000],
                    ['PB', 'João Pessoa',  809000],
                    ['SE', 'Aracaju'     ,  657000]
                ]
```

Observa-se que a lista_estados, que é a *nested list*, é composta por 9 sub-listas. Nesse caso, ao se referir a posição (índice) de lista_estados, estamos referenciando a uma das sub-listas de lista_estados.

No exemplo a seguir estamos imprimindo a posição (índice) 1 de lista_estados. Nesse caso, será exibida a lista ['RN', 'Natal', 884000].

```
lista_estados = [
                    ['BA', 'Salvador'    , 2872000],
                    ['RN', 'Natal'       ,  884000],
                    ['CE', 'Fortaleza'   , 2669000],
                    ['PE', 'Recife'      , 1645000],
                    ['MA', 'São Luiz'    , 1101000],
                    ['AL', 'Maceió'      , 1015000],
                    ['PI', 'Teresina'    ,  864000],
                    ['PB', 'João Pessoa',  809000],
                    ['SE', 'Aracaju'     ,  657000]
                ]

# Imprimindo o conteúdo da posição 1 da variável LISTA_ESTADOS
print(lista_estados[1])
```

Mas se quisermos acessar um dos elementos de uma das sub-listas temos de referenciar primeiro qual sub-lista queremos acessar e em seguida qual elemento dessa sub-lista.

Vemos no exemplo a seguir que nas linhas 14, 15 e 15 foram referenciados 2 índices. O primeiro índice referencia-se a qual sub-lista estou querendo acessar e o segundo índice a qual elemento dessa sub-lista.

```
 1 lista_estados = [
 2                     ['BA', 'Salvador'    , 2872000],
 3                     ['RN', 'Natal'       ,  884000],
 4                     ['CE', 'Fortaleza'   , 2669000],
 5                     ['PE', 'Recife'      , 1645000],
 6                     ['MA', 'São Luiz'    , 1101000],
 7                     ['AL', 'Maceió'      , 1015000],
 8                     ['PI', 'Teresina'    ,  864000],
 9                     ['PB', 'João Pessoa',  809000],
10                     ['SE', 'Aracaju'     ,  657000]
11                 ]
12
13 # Imprimindo o conteúdo da posição 1 da variável LISTA_ESTADOS
14 print(lista_estados[1][0])
15 print(lista_estados[1][1])
16 print(lista_estados[1][2])
```

A seguir iremos estudas algumas funções e alguns métodos de manipulação de listas.

APPEND()

O método **APPEND()** insere um elemento no final de uma lista.

Sintaxe:

```
lista.append(valor)
```

Vejamos o código a seguir:

```
1  lista_siglas  = ['BA', 'CE', 'PE', 'MA', 'AL', 'PI', 'PB', 'SE']
2
3  lista_estados = [
4                    ['BA', 'Salvador'    , 2872000],
5                    ['CE', 'Fortaleza'   , 2669000],
6                    ['PE', 'Recife'      , 1645000],
7                    ['MA', 'São Luiz'    , 1101000],
8                    ['AL', 'Maceió'      , 1015000],
9                    ['PI', 'Teresina'    ,  864000],
10                   ['PB', 'João Pessoa',  809000],
11                   ['SE', 'Aracaju'     ,  657000]
12                ]
13
14 # Exibindo o conteúdo da variável LISTA_SIGLAS
15 print(lista_siglas)
16
17 # Exibindo o conteúdo da variável LISTA_ESTADOS
18 print(lista_estados)
```

Percebemos que, ao imprimir o conteúdo das variáveis lista_siglas e lista_estados (linhas 15 e 16), estão faltando as informações de um estado (RN) em ambas as listas.

Para adicionar os valores ausentes nas listas, iremos usar o método **APPEND()** para adicionar mais uma sigla ('RN') no final da lista lista_siglas e uma nova lista no final da lista lista_estados (['RN', 'Natal', 884000]) e as exibimos novamente. Iremos notar que além do que já fora impresso antes, agora teremos novos valores adicionados no final de cada lista.

```
1  lista_siglas  = ['BA', 'CE', 'PE', 'MA', 'AL', 'PI', 'PB', 'SE']
2
3  lista_estados = [
4                     ['BA', 'Salvador'   , 2872000],
5                     ['CE', 'Fortaleza'  , 2669000],
6                     ['PE', 'Recife'     , 1645000],
7                     ['MA', 'São Luiz'   , 1101000],
8                     ['AL', 'Maceió'     , 1015000],
9                     ['PI', 'Teresina'   ,  864000],
10                    ['PB', 'João Pessoa',  809000],
11                    ['SE', 'Aracaju'    ,  657000]
12                ]
13
14 # Exibindo o conteúdo da variável LISTA_SIGLAS
15 print(lista_siglas)
16
17 # Exibindo o conteúdo da variável LISTA_ESTADOS
18 print(lista_estados)
19
20 # Adicionando a sigla 'RN' na variável LISTA_SIGLAS
21 lista_siglas.append('RN')
22
23 # Adicionando a sublist ['RN', 'Natal', 884000] na variável LISTA_SIGLAS
24 lista_estados.append(['RN', 'Natal', 884000])
25
26 # Exibindo o conteúdo da variável LISTA_SIGLAS
27 print(lista_siglas)
28
29 # Exibindo o conteúdo da variável LISTA_ESTADOS
30 print(lista_estados)
```

Uma vez inseridos os valores nas suas respectivas listas (linhas 21 e 24), solicitamos ao Python que as variáveis sejam novamente exibidas e iremos perceber que os valores inseridos aparecem no final de cada uma das listas.

INSERT()

O método **INSERT()** insere um elemento em uma determinada posição de uma lista.

Sintaxe:

```
lista.insert(posição, valor)
```

Vejamos o código a seguir:

```
1 lista_siglas  = ['BA', 'CE', 'PE', 'MA', 'AL', 'PI', 'PB', 'SE']
2
3 lista_estados = [
4                   ['BA', 'Salvador'   , 2872000],
5                   ['CE', 'Fortaleza'  , 2669000],
6                   ['PE', 'Recife'     , 1645000],
7                   ['MA', 'São Luiz'   , 1101000],
8                   ['AL', 'Maceió'     , 1015000],
9                   ['PI', 'Teresina'   ,  864000],
10                  ['PB', 'João Pessoa',  809000],
11                  ['SE', 'Aracaju'    ,  657000]
12                ]
13
14 # Exibindo o conteúdo da variável LISTA_SIGLAS
15 print(lista_siglas)
16
17 # Exibindo o conteúdo da variável LISTA_ESTADOS
18 print(lista_estados)
19
20 # Adicionando a sigla 'RN' na variável LISTA_SIGLAS
21 lista_siglas.insert(1, 'RN')
22
23 # Adicionando a sublist ['RN', 'Natal', 884000] na variável LISTA_SIGLAS
24 lista_estados.insert(1, ['RN', 'Natal', 884000])
25
26 # Exibindo o conteúdo da variável LISTA_SIGLAS
27 print(lista_siglas)
28
29 # Exibindo o conteúdo da variável LISTA_ESTADOS
30 print(lista_estados)
```

Já vimos que utilizar o método **APPEND()** os valores são inseridos no final da lista. Porém, ao utilizarmos o método **INSERT()**, os valores serão inseridos na posição informada no parâmetro posição.

No exemplo, temos que os valores, anteriormente inseridos pelo método **APPEND()** haviam sido inseridos no final das respectivas listas agora estão sendo inseridos na posição 1 de cada uma.

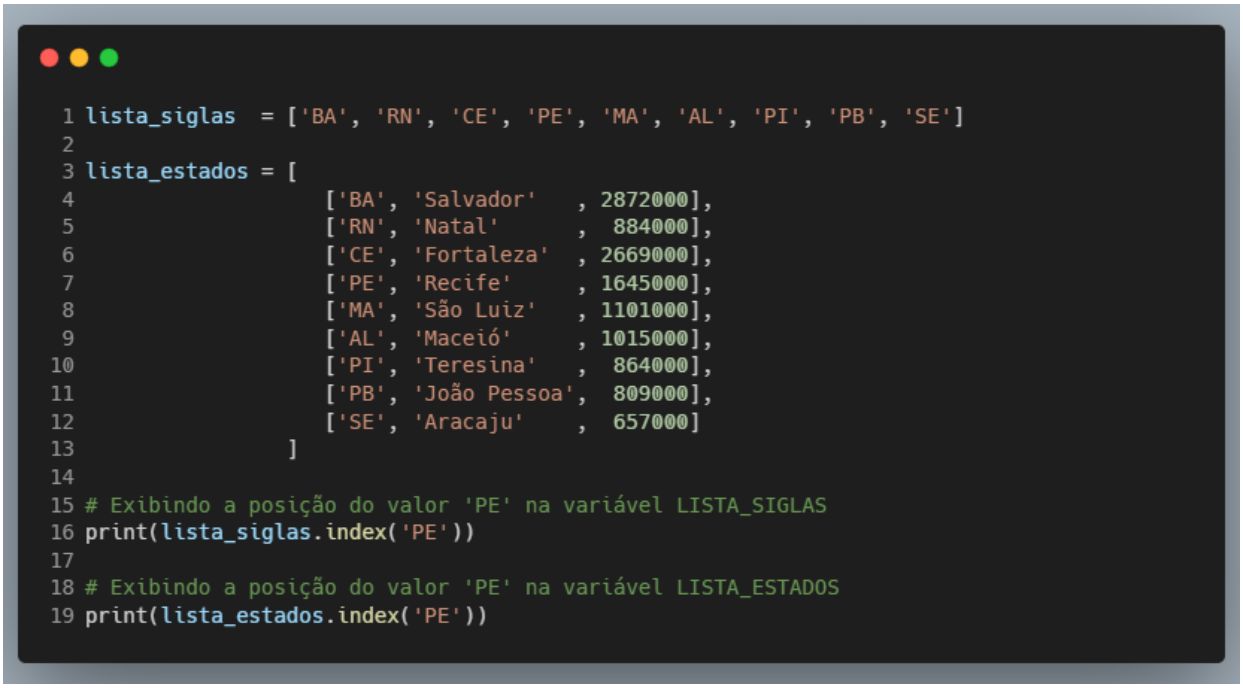
INDEX()

O método **INDEX()** retorna a posição de um determinado elemento em uma lista.

Sintaxe:

```
lista.index(valor)
```

Vejamos o código a seguir:



```
1 lista_siglas  = ['BA', 'RN', 'CE', 'PE', 'MA', 'AL', 'PI', 'PB', 'SE']
2
3 lista_estados = [
4                   ['BA', 'Salvador'   , 2872000],
5                   ['RN', 'Natal'      ,  884000],
6                   ['CE', 'Fortaleza'  , 2669000],
7                   ['PE', 'Recife'     , 1645000],
8                   ['MA', 'São Luiz'   , 1101000],
9                   ['AL', 'Maceió'     , 1015000],
10                  ['PI', 'Teresina'   ,  864000],
11                  ['PB', 'João Pessoa',  809000],
12                  ['SE', 'Aracaju'    ,  657000]
13                ]
14
15 # Exibindo a posição do valor 'PE' na variável LISTA_SIGLAS
16 print(lista_siglas.index('PE'))
17
18 # Exibindo a posição do valor 'PE' na variável LISTA_ESTADOS
19 print(lista_estados.index('PE'))
```

Nesse exemplo, na linha 16 pede que seja impressa a posição que o valor 'PE' ocupa na variável lista_siglas, nesse caso, será impresso o valor 3, lembrando que a primeira posição é 0 (zero).

Fizemos o mesmo procedimento na linha 19. Pedimos que seja impressa a posição que o valor 'PE' ocupa na variável lista_estados, porém ocorreu foi disparada a seguinte exceção:

```
Traceback (most recent call last):
  File "c:\Users\charl\OneDrive\Área de Trabalho\exemplo_01.py", line 19, in <module>
    print(lista_estados.index('PE'))
ValueError: 'PE' is not in list
```

Isso ocorreu porque a variável lista_estados é composta por listas e não por valores individuais. Para que a busca seja feita de maneira correta devemos informar os valores em forma de lista conforme pode ser visto no exemplo a seguir:

```
lista_siglas  = ['BA', 'RN', 'CE', 'PE', 'MA', 'AL', 'PI', 'PB', 'SE']

lista_estados = [
                    ['BA', 'Salvador'   , 2872000],
                    ['RN', 'Natal'      ,  884000],
                    ['CE', 'Fortaleza'  , 2669000],
                    ['PE', 'Recife'     , 1645000],
                    ['MA', 'São Luiz'   , 1101000],
                    ['AL', 'Maceió'     , 1015000],
                    ['PI', 'Teresina'   ,  864000],
                    ['PB', 'João Pessoa',  809000],
                    ['SE', 'Aracaju'    ,  657000]
                ]

# Exibindo a posição do valor 'PE' na variável LISTA_SIGLAS
print(lista_siglas.index('PE'))

# Exibindo a posição do valor 'PE' na variável LISTA_ESTADOS
print(lista_estados.index(['PE', 'Recife', 1645000]))
```

Continuando, vejamos o código a seguir:

```
lista_siglas  = ['BA', 'RN', 'CE', 'PE', 'MA', 'AL', 'PI', 'PB', 'SE']

lista_estados = [
                    ['BA', 'Salvador'   , 2872000],
                    ['RN', 'Natal'      ,  884000],
                    ['CE', 'Fortaleza'  , 2669000],
                    ['PE', 'Recife'     , 1645000],
                    ['MA', 'São Luiz'   , 1101000],
                    ['AL', 'Maceió'     , 1015000],
                    ['PI', 'Teresina'   ,  864000],
                    ['PB', 'João Pessoa',  809000],
                    ['SE', 'Aracaju'    ,  657000]
                ]

# Exibindo a posição do valor 'RJ' na variável LISTA_SIGLAS
print(lista_siglas.index('RJ'))
```

A linha 13 do código acima também irá disparar a exceção a seguir:

```
Traceback (most recent call last):
  File "c:\Users\charl\OneDrive\Área de Trabalho\exemplo_01.py", line 16, in <module>
    print(lista_siglas.index('RJ'))
ValueError: 'RJ' is not in list
```

A exceção se deu pelo fato do valor a ser buscado (RJ) não existir na lista.

Porém, se a intenção é verificar se o valor está na lista devemos usar o operador de teste de inclusão **in**.

Esse operador irá retornar True caso o valor exista na lista e False quando o valor não existir na lista.

```
lista_siglas  = ['BA', 'RN', 'CE', 'PE', 'MA', 'AL', 'PI', 'PB', 'SE']

lista_estados = [
                    ['BA', 'Salvador'   , 2872000],
                    ['RN', 'Natal'      ,  884000],
                    ['CE', 'Fortaleza'  , 2669000],
                    ['PE', 'Recife'     , 1645000],
                    ['MA', 'São Luiz'   , 1101000],
                    ['AL', 'Maceió'     , 1015000],
                    ['PI', 'Teresina'   ,  864000],
                    ['PB', 'João Pessoa',  809000],
                    ['SE', 'Aracaju'    ,  657000]
                ]

# Exibindo a posição do valor 'RJ' na variável LISTA_SIGLAS
if 'RJ' in lista_siglas:
    print(lista_siglas.index('RJ'))
else:
    print('A sigla RJ não consta na lista')
```

REMOVE()

O método **REMOVE()** remove a primeira ocorrência do item de uma lista com base no valor passado como argumento.

Sintaxe:

```
lista.remove(valor)
```

Vejamos o código a seguir:

```
1 lista_siglas  = ['BA', 'RN', 'CE', 'PE', 'MA', 'AL', 'PI', 'PB', 'SE']
2
3 lista_estados = [
4                   ['BA', 'Salvador'   , 2872000],
5                   ['RN', 'Natal'      ,  884000],
6                   ['CE', 'Fortaleza'  , 2669000],
7                   ['PE', 'Recife'     , 1645000],
8                   ['MA', 'São Luiz'   , 1101000],
9                   ['AL', 'Maceió'     , 1015000],
10                  ['PI', 'Teresina'   ,  864000],
11                  ['PB', 'João Pessoa',  809000],
12                  ['SE', 'Aracaju'    ,  657000]
13                ]
14
15 # Removendo o valor 'PE' na variável LISTA_SIGLAS
16 lista_siglas.remove('PE')
17 print(lista_siglas)
18
19 # Removendo o valor 'PE' na variável LISTA_ESTADOS
20 lista_estados.remove('PE')
21 print(lista_estados)
```

No exemplo, solicitamos na linha 16 que seja excluído o elemento PE da lista lista_siglas e percebemos que o método **REMOVE()** foi executado sem erros, porém na linha 20 foi solicitada a exclusão do elemento PE na lista_estados foi disparada a seguinte exceção:

```
Traceback (most recent call last):
  File "c:\Users\charl\OneDrive\Área de Trabalho\exemplo_01.py", line 20, in <module>
    lista_estados.remove('PE')
ValueError: list.remove(x): x not in list
```

Isso ocorreu porque a lista lista_estados é composta por listas e não por valores individuais. Para que a remoção seja feita de maneira correta devemos informar os valores em forma de lista conforme pode ser visto no exemplo a seguir:

```python
lista_siglas  = ['BA', 'RN', 'CE', 'PE', 'MA', 'AL', 'PI', 'PB', 'SE']

lista_estados = [
                    ['BA', 'Salvador'   , 2872000],
                    ['RN', 'Natal'      ,  884000],
                    ['CE', 'Fortaleza'  , 2669000],
                    ['PE', 'Recife'     , 1645000],
                    ['MA', 'São Luiz'   , 1101000],
                    ['AL', 'Maceió'     , 1015000],
                    ['PI', 'Teresina'   ,  864000],
                    ['PB', 'João Pessoa',  809000],
                    ['SE', 'Aracaju'    ,  657000]
                ]

# Removendo o valor 'PE' na variável LISTA_SIGLAS
lista_siglas.remove('PE')
print(lista_siglas)

# Removendo o valor 'PE' na variável LISTA_ESTADOS
lista_estados.remove(['PE', 'Recife'     , 1645000])
print(lista_estados)
```

Mas o que acontece quando o valor informado não consta na lista? Vejamos o exemplo a seguir:

```python
lista_siglas  = ['BA', 'RN', 'CE', 'PE', 'MA', 'AL', 'PI', 'PB', 'SE']

# Removendo o valor 'RJ' na variável LISTA_SIGLAS
lista_siglas.remove('RJ')
```

Como foi informado o valor que não existe na lista, uma exceção será disparada.

```
Traceback (most recent call last):
  File "c:\Users\charl\OneDrive\Área de Trabalho\exemplo_01b.py", line 4, in <module>
    lista_siglas.remove('RJ')
ValueError: list.remove(x): x not in list
```

POP()

O método **POP()** remove e retorna um item da lista com base na posição (índice) passada como argumento.

O argumento (posição) passado para o método é opcional. Caso não seja informado, o método irá assumir como índice o valor -1 (índice do último item) da lista.

Sintaxe:

```
lista.pop(posição)
```

Vejamos o código a seguir:

```
1  lista_siglas  = ['BA', 'RN', 'CE', 'PE', 'MA', 'AL', 'PI', 'PB', 'SE']
2
3  lista_estados = [
4                    ['BA', 'Salvador'    , 2872000],
5                    ['RN', 'Natal'       ,  884000],
6                    ['CE', 'Fortaleza'   , 2669000],
7                    ['PE', 'Recife'      , 1645000],
8                    ['MA', 'São Luiz'    , 1101000],
9                    ['AL', 'Maceió'      , 1015000],
10                   ['PI', 'Teresina'    ,  864000],
11                   ['PB', 'João Pessoa',  809000],
12                   ['SE', 'Aracaju'     ,  657000]
13                ]
14
15 # Removendo a posição 4 da variável LISTA_SIGLAS
16 x = lista_siglas.pop(4)
17 print(x)
18 print(lista_siglas)
19
20 # Removendo a posição 3 da variável LISTA_ESTADOS
21 y = lista_estados.pop(3)
22 print(y)
23 print(lista_estados)
```

No exemplo, na linha 16 solicitamos que seja excluída a posição 4 da lista lista_siglas (‘MA’) e armazenamos o valor excluído na variável **x** e na linha 21 que seja excluída a posição 3 da lista lista_estados (['PE', 'Recife', 1645000]) e armazenamos na variável **y**.

Mais uma vez lembro que a posição inicial de uma lista é 0 (zero).

Mas o que acontece quando a posição informada não consta na lista? Vejamos o exemplo a seguir:

```
1 lista_siglas  = ['BA', 'RN', 'CE', 'PE', 'MA', 'AL', 'PI', 'PB', 'SE']
2
3 # Removendo a posição 4 da variável LISTA_SIGLAS
4 lista_siglas.pop(20)
5 print(lista_siglas)
```

Como foi informado o valor que não existe na lista, uma exceção será disparada.

```
Traceback (most recent call last):
  File "c:\Users\charl\OneDrive\Área de Trabalho\exemplo_01c.py", line 4, in <module>
    lista_siglas.pop(20)
IndexError: pop index out of range
```

O método **POP()** é semelhante a função **DEL()**. Vejamos o código a seguir:

```
1 lista_siglas  = ['BA', 'RN', 'CE', 'PE', 'MA', 'AL', 'PI', 'PB', 'SE']
2
3 lista_estados = [
4                     ['BA', 'Salvador'   , 2872000],
5                     ['RN', 'Natal'      ,  884000],
6                     ['CE', 'Fortaleza'  , 2669000],
7                     ['PE', 'Recife'     , 1645000],
8                     ['MA', 'São Luiz'   , 1101000],
9                     ['AL', 'Maceió'     , 1015000],
10                    ['PI', 'Teresina'   ,  864000],
11                    ['PB', 'João Pessoa',  809000],
12                    ['SE', 'Aracaju'    ,  657000]
13                ]
14
15 # Removendo a posição 4 da variável LISTA_SIGLAS
16 del lista_siglas[4]
17 print(lista_siglas)
18
19 # Removendo a posição 3 da variável LISTA_ESTADOS
20 del lista_estados[3]
21 print(lista_estados)
```

A linha 16 é equivalente a `lista_siglas.pop(4)` e a linha 20 é equivalente a `lista_estados.pop(3)`.

Caso não seja informado um índice da lista, o comando DEL irá excluir todos os elementos da lista.

REMOVE()	POP()	DEL
Método.	Método.	Palavra chave.
Exclui uma determinada posição da lista.	Exclui uma determinada posição da lista, retornando o valor que excluído.	Para excluir deve-se informar uma posição específica da lista. Caso não seja informada, todos os valores da lista serão excluídos. Nenhum valor é retornado.
Exclui apenas um valor por vez.	Exclui apenas um valor por vez.	Exclui apenas um valor por vez ou a lista inteira.
Dispara uma exceção de IndexError caso o índice não exista na lista.	Dispara uma exceção de IndexError caso o índice não exista na lista.	Dispara uma exceção de IndexError caso o índice não exista na lista.

SORT()

O método **SORT()** permite ordenar os valores de uma lista.

Sintaxe:

```
lista.sort([key=Função_usuario],[reverse=True|False])
```

Vejamos o código a seguir:

```
1 lista_siglas  = ['BA', 'RN', 'CE', 'PE', 'MA', 'AL', 'PI', 'PB', 'SE']
2
3 # Imprimindo a variável LISTA_SIGLAS
4 print(lista_siglas)
5
6 # Ordenando a variável LISTA_SIGLAS
7 lista_siglas.sort()
8 print(lista_siglas)
9
10 # Ordenando a variável LISTA_SIGLAS em ordem reversa
11 lista_siglas.sort(reverse=True)
12 print(lista_siglas)
```

Iremos perceber que a instrução da linha 4 irá imprimir a lista lista_siglas na ordem em que foi instanciada, porém na linha 7 a lista será ordenada através do método SORT() e em seguida impressa na linha 8, sendo que dessa vez ela estará ordenada de forma alfabética, pois não foi informado nenhum parâmetro (o valor default é que a lista seja ordenada de forma crescente/alfabética).

Já na linha 11 estamos ordenando a lista lista_siglas sendo que agora de forma decrescente (reverse=True).

Agora vejamos o código a seguir:

```python
lista_estados = [
                    ['BA', 'Salvador'    , 2872000],
                    ['RN', 'Natal'       ,  884000],
                    ['CE', 'Fortaleza'   , 2669000],
                    ['PE', 'Recife'      , 1645000],
                    ['MA', 'São Luiz'    , 1101000],
                    ['AL', 'Maceió'      , 1015000],
                    ['PI', 'Teresina'    ,  864000],
                    ['PB', 'João Pessoa',  809000],
                    ['SE', 'Aracaju'     ,  657000]
                ]

# Imprimindo a variável LISTA_ESTADOS
print(lista_estados)

# Ordenando a variável LISTA_ESTADOS
lista_estados.sort()
print(lista_estados)

# Ordenando a variável LISTA_ESTADOS usando a coluna 1 ordenador
lista_estados.sort(key=lambda coluna: coluna[1])
print(lista_estados)
```

Na linha 14 estamos imprimindo a lista lista_estados na ordem em que foi instanciada. Vemos na linha 17 que estamos ordenando a lista. Embora a lista_estados seja composta por listas, o método **SORT()** irá utilizar a primeira posição (0) de cada lista como valor (chave) de ordenação.

Notamos que a utilização do método **SORT()** em uma lista simples não requer a utilização do parâmetro key, uma vez que em cada posição há apenas um valor. *Mas e quando a lista possuir, ao invés de um único valor em cada posição, possuir uma lista contendo diversos valores, onde não queremos utilizar o primeiro valor de cada lista como chave de ordenação?*

Caso queiramos que a chave de ordenação não seja a primeira posição de cada lista (0), mas uma outra posição, (no caso, o nome das capitais que é a posição 1) devemos utilizar o parâmetro key conforme especificado na linha 21, passando como argumento uma função, que nesse caso utilizamos o recursos de uma função *lambda*[2].

O método **SORT()** não retorna valor (None). A ordenação que ele realiza é sobre a própria lista, porém quando queremos ordenar uma lista, retornando-a para uma outra lista usamos a função **SORTED()**.

[2] Iremos abordar as funções lambda mais adiante.

SORTED()

A função **SORTED()** permite ordenar os valores de uma lista, retornando uma lista como resultado.

Sintaxe:

```
sorted(variável_iteravel, [key=Função_usuario], [reverse=True|False])
```

Vejamos o código a seguir:

```
1 lista_siglas  = ['BA', 'RN', 'CE', 'PE', 'MA', 'AL', 'PI', 'PB', 'SE']
2
3 # Imprimindo a variável LISTA_SIGLAS
4 print(lista_siglas)
5
6 # Ordenando a variável LISTA_SIGLAS
7 lista_siglas_2 = lista_siglas.sort()
8 print(lista_siglas_2)
```

Na linha 7 estamos ordenando a lista lista_siglas com o método **SORT()** e armazenando o seu resultado na lista lista_siglas_2.

Porém ao solicitarmos a impressão da lista lista_siglas_2 na linha 8 iremos observar que o valor impresso será None, pois o método **SORT()** não retorna valor algum, ele ordena dentro da mesma lista apenas (lista_siglas).

Para que possamos gerar a lista_siglas_2, mantendo a lista lista_siglas do mesmo jeito que foi instanciada devemos utilizar a função **SORTED()** conforme pode ser visto a seguir:

```
1 lista_siglas  = ['BA', 'RN', 'CE', 'PE', 'MA', 'AL', 'PI', 'PB', 'SE']
2
3 # Imprimindo a variável LISTA_SIGLAS
4 print(lista_siglas)
5
6 # Ordenando a variável LISTA_SIGLAS
7 lista_siglas_2 = sorted(lista_siglas)
8 print(lista_siglas_2)
```

Os argumentos `key` e `reverse` funcionam da mesma forma que o método **SORT().**