

# **COMPUTAÇÃO 1 – PYTHON**

## **AULA 6 TEÓRICA**

**SILVIA BENZA**

**SILVIABENZA@COS.UFRJ.BR**

# **LISTAS**

**VAMOS A APRENDER COMO TRABALHAR  
COM A VERSATILIDADE DAS LISTAS!**

[start : end]	: vai do índice start até o índice end-1
[start : ]	: vai de start até o final da lista
[ : end]	: vai do início da lista até end-1
[ : ]	: copia a lista toda

# FATIAS

Como vimos na última aula podemos usar a notação de fatias (slices) para criar novas listas.

```
>>> lista = ['a',2,[3,'f'], 'q']
```

```
>>> lista[1:]
```

```
>>> lista[:1]
```

```
>>> lista[1:2]
```

```
>>> lista[0:-1]
```

**[start : end]** : vai do índice *start* até o índice *end-1*  
**[start : ]** : vai de *start* até o final da lista  
**[ : end]** : vai do início da lista até *end-1*  
**[ : ]** : copia a lista toda

# FATIAS

Como vimos na última aula podemos usar a notação de fatias (slices) para criar novas listas.

```
>>> lista = ['a',2,[3,'f'], 'q']
```

```
>>> lista [1:]
```

```
[2,[3,'f'], 'q']
```

Índice 1 até o índice final

```
>>> lista [:1]
```

```
['a']
```

Índice 0 até o índice 0

```
>>> lista [1:2]
```

```
[2]
```

Índice 1 até o índice 1

```
>>> lista [0:-1]
```

```
['a',2,[3,'f']]
```

Índice 0 até o índice -2

[start:end:step] : vai do índice *start* até *end* (sem ultrapassá-lo),  
com passo *step*

# FATIAS

**Incremento:** assim também podemos usar incremento ou decremento para selecionar os elementos de uma lista.

```
>>> lista = [1,2,3,4,5,6]
```

```
>>> lista[0:-1:2]
```

```
>>> lista[5:0:-1]
```

```
>>> lista[0:-1:3]
```

```
>>> lista[::-1]
```

[start:end:step] : vai do índice *start* até *end* (sem ultrapassá-lo),  
com passo *step*

# FATIAS

**Incremento:** assim também podemos usar incremento ou decremento para selecionar os elementos de uma lista.

```
>>> lista = [1,2,3,4,5,6]
```

```
>>> lista[0:-1:2]
```

```
[1,3,5]
```

Índice 0 até o índice -2 de 2 em 2

```
>>> lista[5:0:-1]
```

```
[6,5,4,3,2]
```

Índice 5 até o índice 1 de 1 em 1

```
>>> lista[0:-1:3]
```

```
[1,4]
```

Índice 0 até o índice -2 de 3 em 3

```
>>> lista[::-1]
```

```
[6,5,4,3,2,1]
```

Inverte a lista

# FATIAS

**Atribuição:** ao atribuir uma sequência a uma fatia, os elementos desta devem ser substituídos pelos elementos daquela.

```
>>> lista = [1,2,3,4,5]
```

```
>>> lista[1:3] = ['a','b']
```

```
>>> lista[2:] = ['a','b']
```

# FATIAS

**Atribuição:** ao atribuir uma sequência a uma fatia, os elementos desta devem ser substituídos pelos elementos daquela.

```
>>> lista = [1,2,3,4,5]
```

[1, 2, 3, 4, 5]

[1, "a", "b", 4, 5]

lista[1:3] = ["a","b"]

[1, 2, 3, 4, 5]

[1, 2, "a", "b"]

lista[2:] = ["a","b"]

# FATIAS

**Atribuição:** ao atribuir uma sequência a uma fatia, os elementos desta devem ser substituídos pelos elementos daquela.

```
>>> lista = [1,2,3,4,5]
```

```
>>> lista[:2] = ['a','b']
```

```
>>> lista[:2] = [['a','b']]
```

# FATIAS

**Atribuição:** ao atribuir uma sequência a uma fatia, os elementos desta devem ser substituídos pelos elementos daquela.

```
>>> lista = [1,2,3,4,5]
```

[1, 2, 3, 4, 5]

["a", "b", 3, 4, 5]

lista[:2] = ["a","b"]

[1, 2, 3, 4, 5]

[ ["a", "b"], 3, 4, 5]

lista[:2] = [["a","b"]]



# FATIAS

**Atribuição:** ao atribuir uma sequência a uma fatia, os elementos desta devem ser substituídos pelos elementos daquela.

```
>>> lista = [1,2,3,4,5]
```

```
>>> novalista = [8,10]
```

```
>>> lista[2:2] = novalista
```

```
>>> lista[1:4] = novalista
```



# FATIAS

**Atribuição:** ao atribuir uma sequência a uma fatia, os elementos desta devem ser substituídos pelos elementos daquela.

```
>>> lista = [1,2,3,4,5]
```

```
>>> novalista = [8,10]
```

[1, 2, 3, 4, 5]

[1, 2, 8, 10, 3, 4, 5]

lista[2:2] = novalista

[1, 2, 3, 4, 5]

[1, 8, 10, 5]

lista[1:4] = novalista

# FATIAS

**Atribuição:** ao atribuir uma **sequência** a uma fatia, os elementos desta devem ser **substituídos** pelos elementos daquela.



# EXERCÍCIO!

**Atribuição:** ao atribuir uma **sequência** a uma fatia, os elementos desta devem ser **substituídos** pelos elementos daquela.

```
>>> lista = [1,2,3,4,5]
```

```
>>> lista[1:1] = ['z']
```

```
>>> lista[1:3] = [[7]]
```

```
>>> lista[1:-1]= [8,9,10]
```

```
>>> lista[:3]="xyz"
```

```
>>> lista[:3]="a,b,c"
```

```
>>> lista[:2]=1,2,3
```

# EXERCÍCIO!

**Atribuição:** ao atribuir uma **sequência** a uma fatia, os elementos desta devem ser **substituídos** pelos elementos daquela.

```
>>> lista = [1,2,3,4,5]
```

```
>>> lista[1:1] = ['z']
```

```
[1, 'z', 2, 3, 4, 5]
```

```
>>> lista[1:3] = [[7]]
```

```
[1, [7], 3, 4, 5]
```

```
>>> lista[1:-1] = [8,9,10]
```

```
[1, 8, 9, 10, 5]
```

```
>>> lista[:3] = "xyz"
```

```
['x','y','z',10,5]
```

```
>>> lista[:3] = "a,b,c"
```

```
['a', ',', 'b', ',', 'c', 10, 5]
```

```
>>> lista[:2] = 1,2,3
```

```
[1, 2, 3, 'b', ',', 'c', 10, 5]
```



# CÓPIA

**Copia com atribuição:** muito cuidado na hora de fazer copias de listas!!!

Lembrem como o python salva os valores das variáveis ;)

```
>>> l1 = [1,2,3,4,5]
```

```
>>> l2 = l1
```

```
>>> l1
```

```
[1,2,3,4,5]
```

```
>>> l2
```

```
[1,2,3,4,5]
```

# CÓPIA

**Copia com atribuição:** muito cuidado na hora de fazer copias de listas!!!

Lembrem como o python salva os valores das variáveis ;)

```
>>> l2[0]=9
```

```
>>> l2
```

```
[9,2,3,4,5]
```

```
>>> l1
```

# CÓPIA

**Copia com atribuição:** muito cuidado na hora de fazer copias de listas!!!

Lembrem como o python salva os valores das variáveis ;)

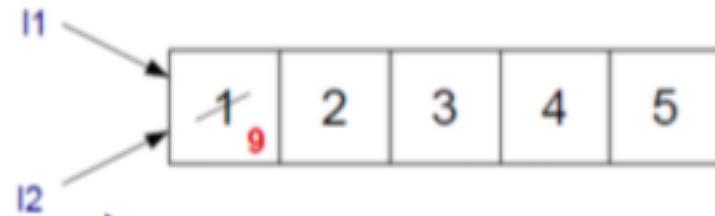
```
>>> l2[0]=9
```

```
>>> l2
```

```
[9,2,3,4,5]
```

```
>>> l1
```

```
[9,2,3,4,5]
```



# CÓPIA (DE VERDADE)

**Copia com atribuição:** muito cuidado na hora de fazer copias de listas!!!

Lembrem como o python salva os valores das variáveis ;)

```
>>> l1 = [1,2,3,4,5]
```

```
>>> l2 = l1[:]
```

```
>>> l1
```

```
[1,2,3,4,5]
```

```
>>> l2
```

```
[1,2,3,4,5]
```

```
>>> l2[0]=9
```

```
>>> l2
```

```
[9,2,3,4,5]
```

```
>>> l1
```

```
[1,2,3,4,5]
```



# **MANIPULAÇÃO DE LISTAS**

# FUNÇÕES DE MANIPULAÇÃO

Além dos operadores + (concatenação) e \* (usado para múltiplas concatenações) podemos manipular listas usando:

- **append** : outra forma de **concatenação**. Neste caso, a lista é tratada como uma **fila**.
- **extend** : permite **adicionar** os elementos de uma lista a outra
- **del** : **remover** elemento de uma lista

# FUNÇÕES DE MANIPULAÇÃO

```
>>> lista=[]
>>> list.append(lista,'a')
>>> lista
['a']
>>> list.append(lista,2)
>>> lista
['a', 2]
>>> list.append(lista,[3,'f'])
>>> lista
['a', 2, [3, 'f']]
```

# FUNÇÕES DE MANIPULAÇÃO

```
>>> lista
['a', 2, [3, 'f']]
>>> list.extend(lista,['q'])
>>> lista
['a', 2, [3, 'f'], 'q']
>>> list.extend(lista,[3,7])
>>> lista
['a', 2, [3, 'f'], 'q', 3, 7]
>>> list.extend(lista,10)
```

Traceback (most recent call last):

```
  File "<pyshell#11>", line 1, in <module>
    list.extend(lista,10)
```

TypeError: 'int' object is not iterable

```
>>> list.extend(lista,"bola")
>>> lista
['a', 2, [3, 'f'], 'q', 3, 7, 'b', 'o', 'l', 'a']
```

# FUNÇÕES DE MANIPULAÇÃO

```
>>> lista
['a', 2, [3, 'f'], 'q', 3, 7, 'b', 'o', 'l', 'a']
>>> del lista[1]
>>> lista
['a', [3, 'f'], 'q', 3, 7, 'b', 'o', 'l', 'a']
>>> del lista[7]
>>> lista
['a', [3, 'f'], 'q', 3, 7, 'b', 'o', 'a']
>>> del lista[1][1]      (Como o segundo elemento de lista é uma lista,
                           posso retirar desta seu segundo elemento)
>>> lista
['a', [3], 'q', 3, 7, 'b', 'o', 'a']
>>> del lista[2][1]
Traceback (most recent call last):
  File "<pyshell#20>", line 1, in <module>
    del lista[2][1]
TypeError: 'str' object doesn't support item deletion
```

# FUNÇÕES DE MANIPULAÇÃO

**list.insert(lista,índice, elemento):**  
insere elemento na lista na posição indicada por índice.

Ex.:

```
>>> lista = [0,1,2,3]
```

```
>>> list.insert(lista,1,'dois')
```

```
>>> lista
```

```
[0,'dois', 1, 2, 3]
```

Como o insert, altera a lista ao invés de retornar a lista. O valor retornado é None!

**Atribuições a fatias** servem para a mesma finalidade mas são menos legíveis.

```
>>> lista = [0,1,2,3]
```

```
>>> lista[1:1] = ['dois']
```

```
>>> lista
```

```
[0,'dois', 1, 2, 3]
```

# FUNÇÕES DE MANIPULAÇÃO

**list.remove(lista, elemento):** Remove da lista o primeiro elemento igual a elemento. Se não existe tal elemento, um erro é gerado.

Ex.:

```
>>> lista = ['oi', 'alo', 'ola']
>>> lista.remove('alo')
>>> lista
['oi', 'ola']
```

```
>>> lista.remove('oba')
Traceback (most recent call last):
File "<pyshell#116>", line 1, in <module>
    lista.remove('oba")
ValueError: list.remove(x): x not in list
```

# FUNÇÕES DE MANIPULAÇÃO

**list.remove(lista, elemento):** Remove da lista o primeiro elemento igual a elemento. Se não existe tal elemento, um erro é gerado.

```
>>> lista = [1,3,6,7,1,5,1]
>>> list.remove(lista,1)

>>> lista
[3,6,7,1,5,1]
```



# FUNÇÕES DE MANIPULAÇÃO

Observe a diferença entre **del** e **remove**:

Suponha lista = [4,6,7,1,2], e digamos que quero deletar o elemento 1.

Para o del é preciso indicar o índice do elemento da lista que se deseja deletar:

**del lista[3]**

Enquanto que para o remove basta indicar o elemento a ser deletado:

**list.remove(lista, 1)**

# FUNÇÕES DE MANIPULAÇÃO

**list.pop(lista, índice):** Remove da lista o elemento na posição índice e o retorna. Se índice não for mencionado, é assumido o último.

Ex.:

```
>>> lista = [1,2,3,4]
>>> lista.pop(lista)
4
>>> lista
[1,2, 3]
>>> deletado = lista.pop(lista,1)
>>> deletado
2
>>> lista
[1,3]
```

Enquanto que o `del` não retorna nada, o `pop` **retorna** o elemento deletado!!!!

# FUNÇÕES DE MANIPULAÇÃO

**list.count(lista, elemento)**: Retorna quantas vezes o elemento aparece na lista

Ex.:

```
>>> lista = [9,8,33,12,33]
```

```
>>> lista.count(33)
```

2

**index(elemento)**: Retorna o índice da **primeira** ocorrência de elemento na lista.  
Um erro ocorre se elemento não consta da lista.

Ex.:

```
>>> lista.index(33)
```

2

```
>>> lista.index(7)
```

Traceback (most recent call last):

File "<pyshell#110>", line 1, in <module>

lista.index(7)

ValueError: 7 is not in list

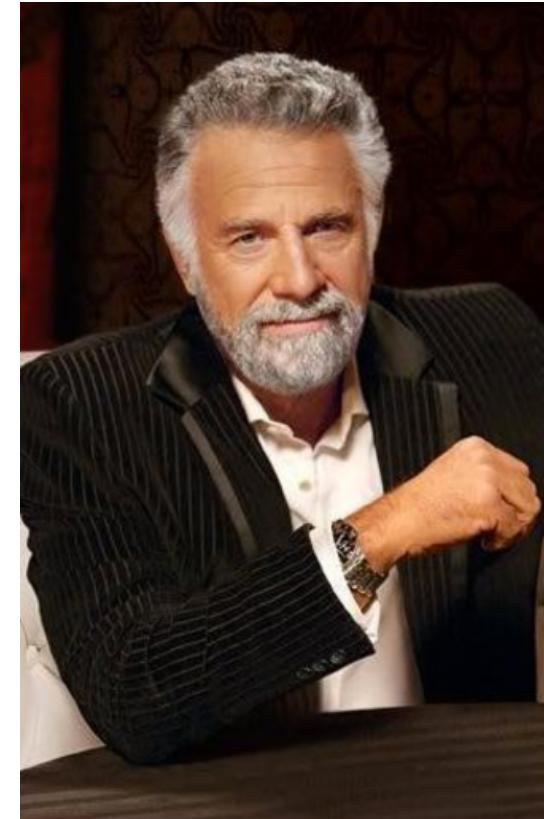
# FUNÇÕES DE MANIPULAÇÃO

Quando precisamos procurar um elemento na lista, não é uma boa utilizar o método **index**, se não existe da erro!!!!!

Melhor usar o método **in**.

```
>>> lista = [1, 4, 8, 3, 2]
>>> 2 in lista
True
>>> 10 in lista
False
```

Se não existe retorna False ;)



# EXERCÍCIO

Faça um função que dada uma lista e um elemento, retorna em que posição da lista aquele elemento se encontra. Se o elemento não estiver na lista, retorne uma mensagem.

**Obs: Garanta que não haverá erro.**

# EXERCÍCIO

Faça um função que dada uma lista e um elemento, retorna em que posição da lista aquele elemento se encontra. Se o elemento não estiver na lista, retorne uma mensagem.

**Obs: Garanta que não haverá erro.**

```
# |Função que procura um elemento em uma lista, e retorna
# a posição em que ele está ou uma mensagem de erro
# caso o elemento não esteja na lista
# list, any type -> int/str
def procura(lista,elemento):
    if elemento in lista:
        return lista.index(elemento)
    else:
        return 'não está na lista'
```

# EXERCÍCIO

Faça um função que dada uma lista e um elemento, retorna em que posição da lista aquele elemento se encontra. Se o elemento não estiver na lista, retorne uma mensagem.

**Obs: Garanta que não haverá erro.**

```
>>> posicao = procura([1,4,8,3,2],2)
>>> posicao
4
>>> posicao = procura([1,4,8,3,2],7)
>>> posicao
'O elemento nao esta na lista'
```

# FUNÇÕES DE MANIPULAÇÃO

**reverse()**: Inverte a ordem dos elementos da lista.

```
>>> lista=[1,2,3]
```

```
>>> list.reverse(lista)
```

```
>>> lista
```

```
[3,2,1]
```

**sort()**: ordena uma lista.

```
>>> lista=[2,1,3]
```

```
>>> list.sort(lista)
```

```
>>> lista
```

```
[1,2,3]
```

# EXERCÍCIO

Considere a função `alteraLista` abaixo:

```
# list -> list
def alteralista(lista):
    list.append(lista,10)
    list.append(lista, [3, 'bola'])
    list.append(lista, 'lua')
    list.extend(lista, [1,2,3])
    list.extend(lista, 'lua')
    del lista[2]
    list.insert(lista,2,1)
    list.remove(lista,2)
    elemento = list.pop(lista,3)
    list.insert(lista,1,elemento)
    return lista
```

Qual será a saída da função se a chamada for:

```
>>> alteraLista([4,5])
```

# LEMBREM!!!

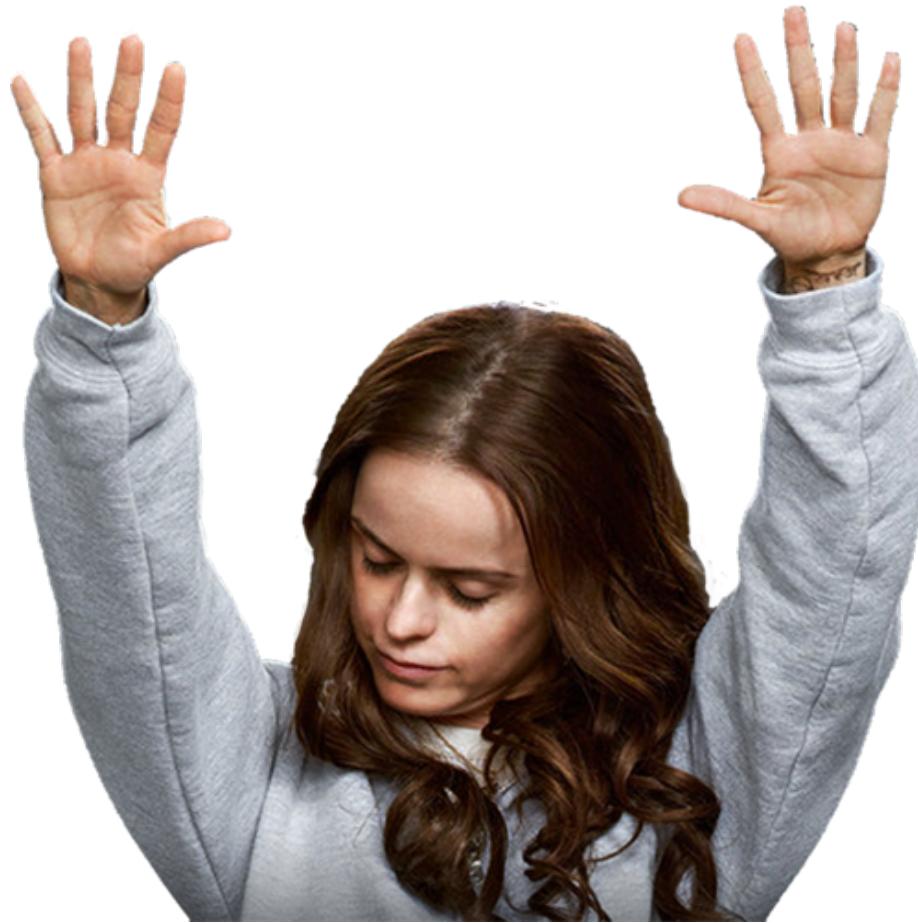
Algumas funções que manipulam listas não possuem valor de retorno:

- `list.append`
- `list.extend`
- `list.insert`
- `list.remove`
- `list.reverse`
- `list.sort`

Enquanto outras possuem:

- `list.pop`
- `list.count`
- `list.index`

# **LIBERADOS!**



# **COMPUTAÇÃO 1 – PYTHON**

## **AULA 6 TEÓRICA**

**SLIDES BASEADOS NOS TRABALHOS:**  
**AULAS TEÓRICAS DO DCC UFRJ**  
**AULA DO CLAUDIO ESPERANÇA DO PESC**