Computação I - Python
Aula 6 - Teórica: Listas

João C. P. da Silva

Carla A. D. M. Delgado

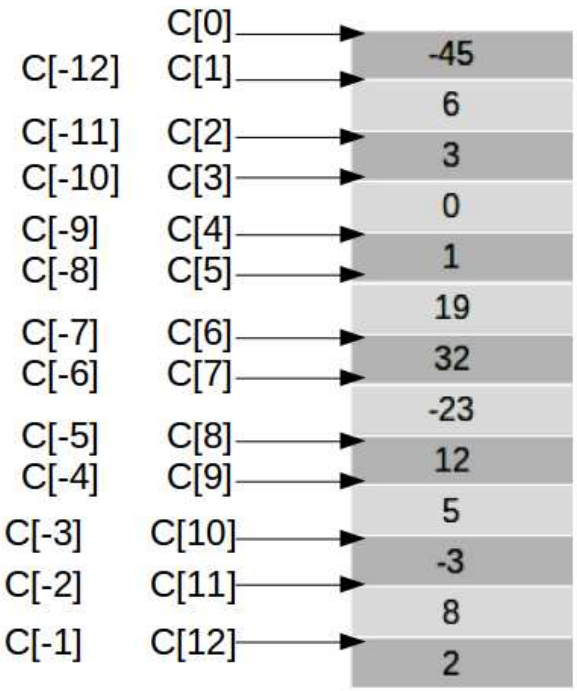
Ana Luisa Duboc

Dept. Ciência da Computação - UFRJ

Listas - Fatias

Podemos usar a notação de fatias (slices) em listas:

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

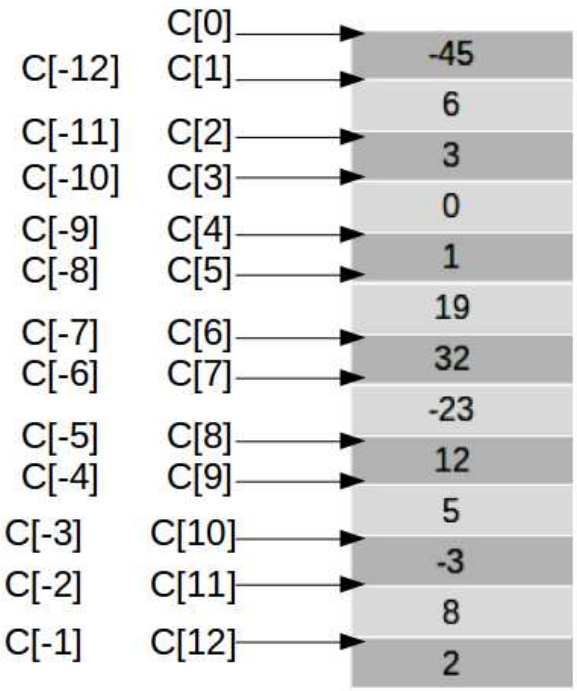



```
1 >>> c = [-45,6,3,0,1,19,32,-23,12,5,-3,8,2]
2
3 >>> c[-1:-5:1]
4
5
6 >>> c[-5:-1:1]
```

Listas - Fatias

Podemos usar a notação de fatias (slices) em listas:

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




```
>>> c = [-45,6,3,0,1,19,32,-23,12,5,-3,8,2]

>>> c[-1:-5:1]
[ ]

>>> c[-5:-1:1]
[12, 5, -3, 8]
```

Listas - Fatias

Podemos usar a notação de fatias (slices) em listas:

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

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

>>> lista [1:]


>>> lista [:1]


>>> lista [1:2]


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

Listas - Fatias

Podemos usar a notação de fatias (slices) em listas:

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

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

>>> lista [1:]
[2, [3, 'f'], 'q']

>>> lista [:1]
['a']

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

>>> lista [0:-1]
['a', 2, [3, 'f']]
```

Listas - Fatias

Incremento: podemos usar incremento / decremento para selecionar os elementos de uma fatia:

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

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

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


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


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


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

Listas - Fatias

Incremento: podemos usar incremento / decremento para selecionar os elementos de uma fatia:

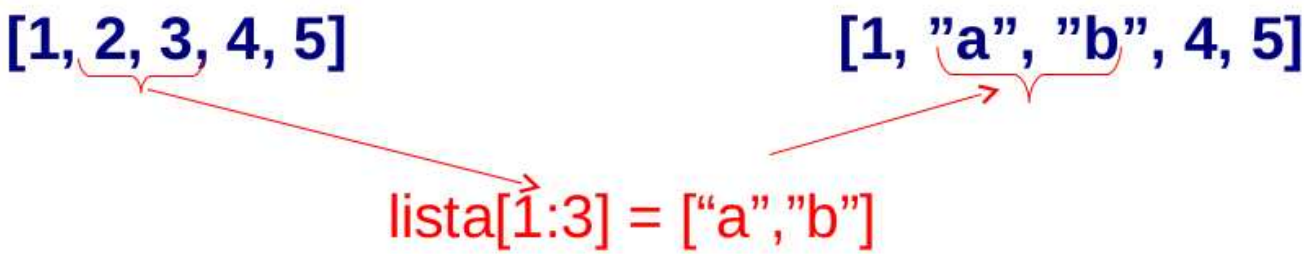
- [start : end : step] : vai do índice *start* até *end* (sem ultrapassá-lo), com passo *step*.
- **Exemplo**

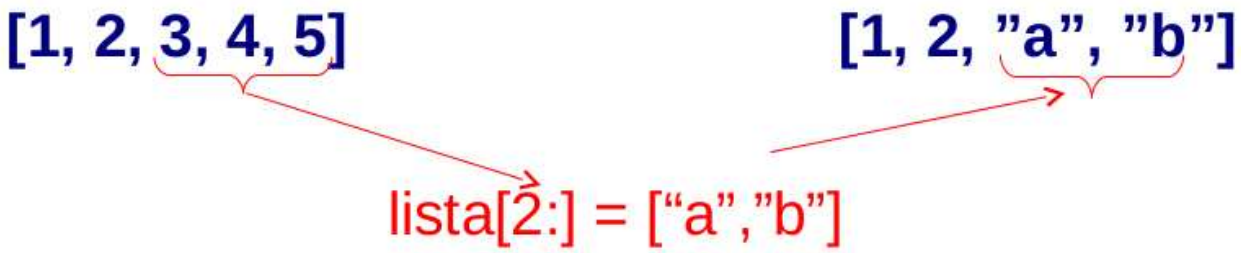
```
>>> lista = [1,2,3,4,5,6]

>>> lista[0:-1:2]   # Indice 0 ate o indice -2 de 2 em 2
[1, 3, 5]

>>> lista[5:0:-1]   # Indice 5 ate o indice 1 de 1 em 1
[6, 5, 4, 3, 2]

>>> lista[0:-1:3]   # Indice 0 ate o indice -2 de 3 em 3
[1, 4]

>>> lista[::-1]     # Inverte a lista
[6, 5, 4, 3, 2, 1]
```

Listas - Fatias

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

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

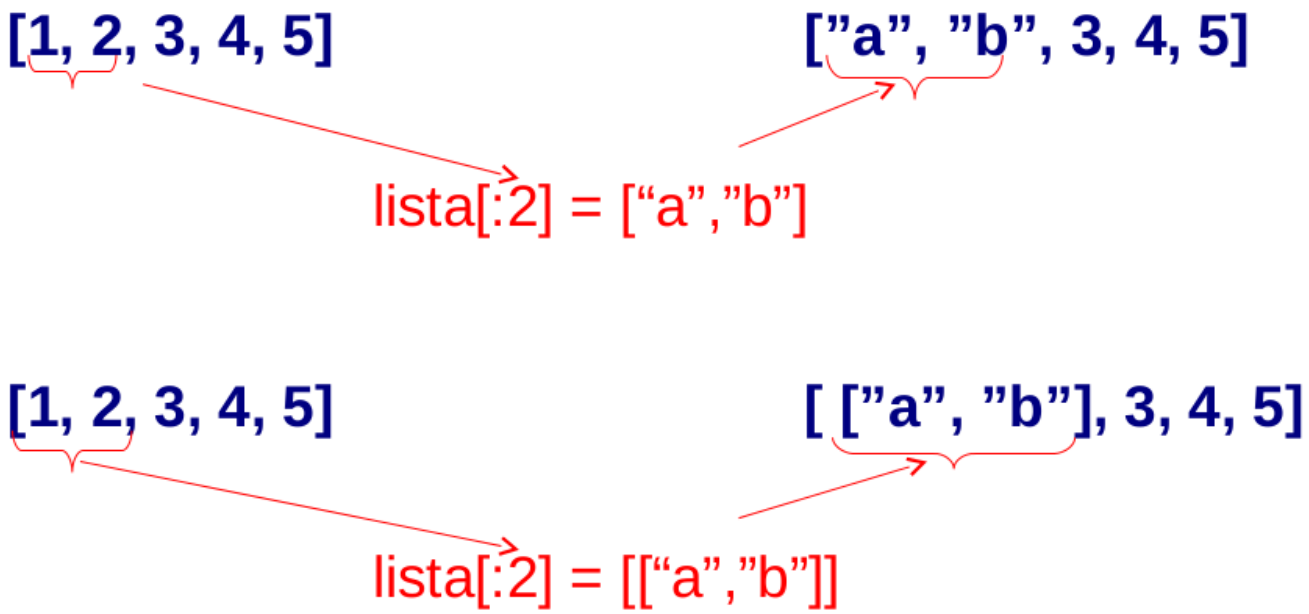




Listas - Fatias

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

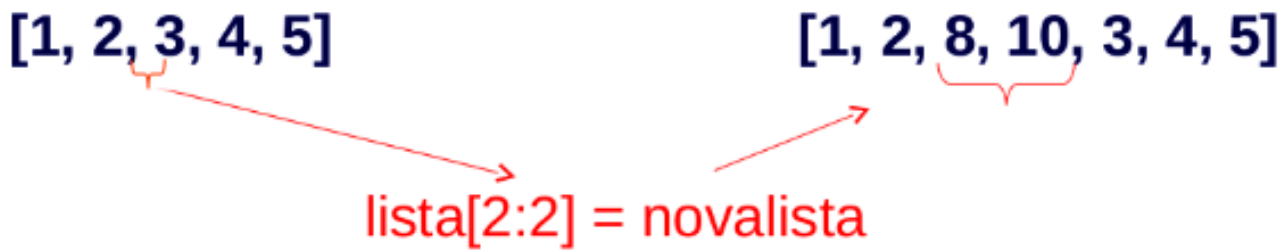
```
1 >>> lista = [1,2,3,4,5]
```

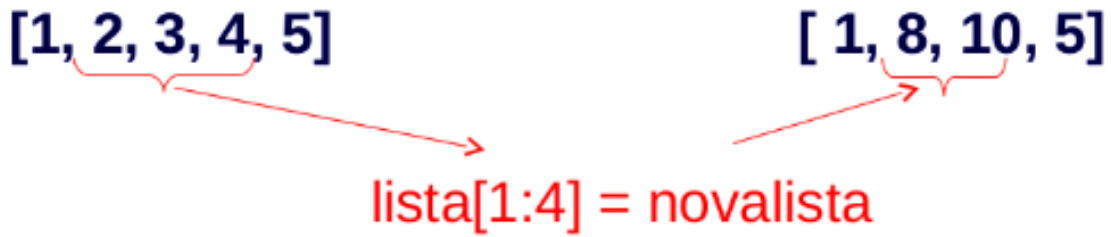


Listas - Fatias

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

```
1 >>> lista = [1,2,3,4,5]
2
3 >>> novalista = [8,10]
```





Listas - Fatias

```
>>> 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
```

Listas - Fatias

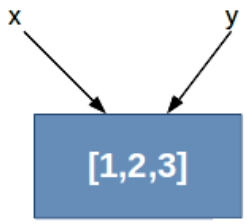
```
>>> lista = [1,2,3,4,5]
>>> lista [1:1] = ['z']
[1, 'z', 2, 3, 4, 5]

>>> 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]
```

Observe que a lista vai sendo alterada

Alias e Tipos Mutáveis

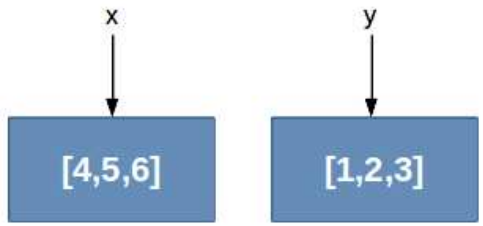
Relembrando - um *alias* acontece quando duas variáves se referem ao mesmo dado:

```
>>> y = [1,2,3]
>>> x = y
>>> x
[1,2,3]
>>> y
[1,2,3]
```



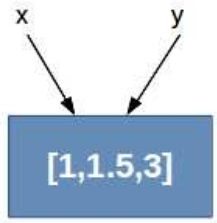
Qualquer nova atribuição feita a uma das variáveis quebrará o *alias* entre elas.

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



Porém quando o dado é mutável (lista!!), o *alias* não é quebrado caso uma atribuição modifique uma parte do dado:

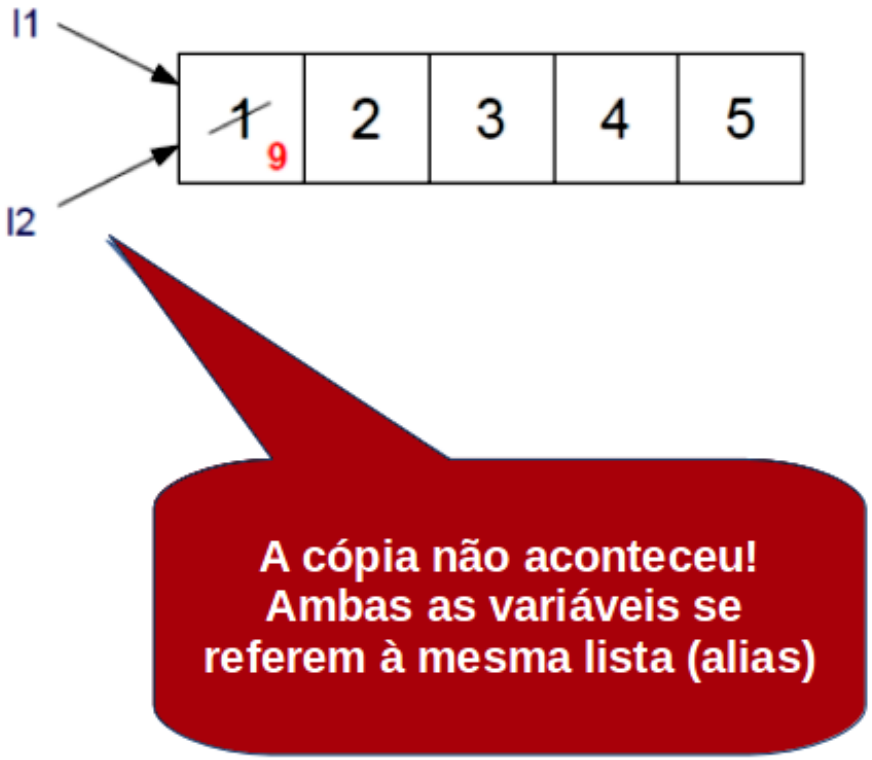
```
>>> y = [1,2,3]
>>> x = y
>>> x[1]=1.5
>>> x
[1, 1.5, 3]
>>> y
[1, 1.5, 3]
```



Listas - Cópias

Cuidado quando fizer cópia de listas!

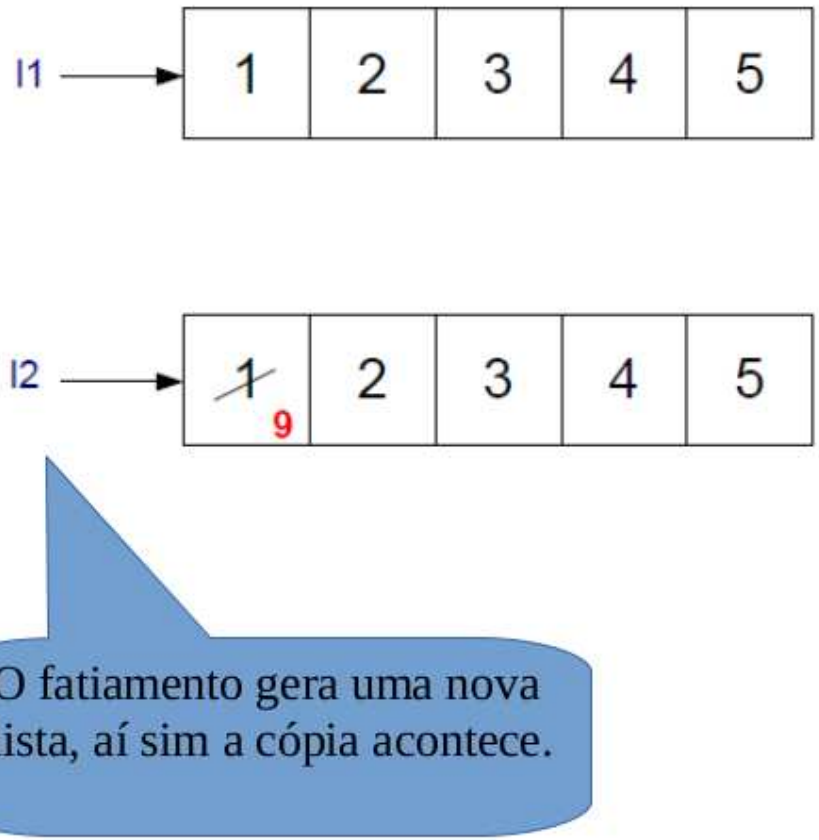
```
>>> l1 = [1,2,3,4,5]
>>> l2 = l1
>>> l1
[1,2,3,4,5]

>>> l2
[1,2,3,4,5]

>>> l2[0]=9
>>> l2
[9,2,3,4,5]

>>> l1
[9,2,3,4,5]
```



Listas - Cópias

Cuidado quando fizer cópia de listas!

```
>>> 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

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.

Manipulação de Listas

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

>>> list.append(lista,2)
>>> lista
['a', 2]

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

Manipulação de Listas

```
>>> 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']
```

Manipulação de Listas

```
>>> 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']
# Como o segundo elemento de lista e uma lista,
# posso retirar desta seu segundo elemento

>>> del lista[1][1]
>>> 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
```

Manipulação de Listas

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

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

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

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

- Como o *extend*, 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]
```

Manipulação de Listas

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

```
>>> lista = ['oi', 'alo', 'ola']

>>> list.remove(lista, 'alo')

>>> lista
['oi', 'ola']

>>> list.remove(lista, 'oba')

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

Manipulação de Listas

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

```
1 >>> lista = [1,3,6,7,1,5,1]
2 >>> list.remove(lista,1)      # Remove apenas a primeira
3                               # ocorrencia do elemento!
4 >>> lista
5 [3,6,7,1,5,1]
```

Manipulação de Listas

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)**

Manipulação de Listas

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

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

>>> lista
[1,2,3]

>>> deletado = list.pop(lista,1)
>>> deletado
2

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

A diferença entre **del** e **pop** é que este retorna o elemento deletado, enquanto o del não.

Manipulação de Listas

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

```
>>> lista = [9,8,33,12,33]
>>> list.count(lista,33)
2
```

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

```
>>> list.index(lista, 33)
2
>>> list.index(lista,7)
Traceback (most recent call last):
File "<pyshell#110>", line 1, in <module>
lista.index(7)
ValueError: 7 is not in list
```

Manipulação de Listas

- **OBSERVAÇÃO**: Usar o index para saber se o elemento está numa lista não é uma boa idéia, porque se não estiver, dará erro.
- Uma forma de saber se um elemento está numa lista é usar o in, conforme exemplificado abaixo:

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

>>> 10 in lista
False
```

Manipulação de Listas

Faça uma 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.

Manipulação de Listas

Faça uma 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.

```
1  def procura(lista,elemento):
2    "Funcao que procura um elemento em uma lista, e retorna
3     a posicao em que ele esta ou uma mensagem de erro
4     caso o elemento nao esteja na lista
5     Parametro de Entrada: list, any type
6     Valor de Retorno: int / str"
7
8    if elemento in lista:
9      return list.index(lista,elemento)
10   else:
11     return 'Nao esta na lista'
```

Manipulação de Listas

Faça uma 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.

```
1  def procura(lista,elemento):
2    "Funcao que procura um elemento em uma lista, e retorna
3     a posicao em que ele esta ou uma mensagem de erro
4     caso o elemento nao esteja na lista
5     Parametro de Entrada: list, any type
6     Valor de Retorno: int / str"
7
8    if elemento in lista:
9      return list.index(lista,elemento)
10   else:
11     return 'Nao esta na lista'
```

```
1 >>> posicao = procura([1,4,8,3,2],2)
2 >>> posicao
3 4
4 >>> posicao = procura([1,4,8,3,2],7)
5 >>> posicao
6 'Nao esta na lista'
```

Manipulação de Listas

- **list.reverse(lista)**: inverte a ordem dos elementos da lista.

```
>>> lista=[1,2,3]
>>> list.reverse(lista)
>>> lista
[3,2,1]
```

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

```
>>> lista=[2,1,3]
>>> list.sort(lista)
>>> lista
[1,2,3]
```

Manipulação de Listas

ATENÇÃO

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

Manipulação de Listas

Considere a função **alteraLista** abaixo:

```
1  def alteraLista(lista):
2    "Parametro de Entrada: list
3     Valor de Retorno: list"
4
5    list.append(lista,10)
6    list.append(lista,[3,'bola'])
7    list.append(lista,'lua')
8    list.extend(lista,[1,2,3])
9    list.extend(lista,'lua')
10   del lista[2]
11   list.insert(lista,2,1)
12   list.remove(lista,2)
13   elemento = list.pop(lista,3)
14   list.insert(lista,1,elemento)
15   return lista
```

Qual será a saída da função se a chamada for **alteraLista([4,5])**

Autores

- João C. P. da Silva ▸ Lattes
- Carla Delgado ▸ Lattes
- Ana Luisa Duboc ▸ Lattes

Colaboradores

- Anamaria Martins Moreira ▸ Lattes
- Fabio Mascarenhas ▸ Lattes
- Leonardo de Oliveira Carvalho ▸ Lattes
- Charles Figueiredo de Barros ▸ Lattes
- Fabrício Firmino de Faria ▸ Lattes

Computação I - Python
Aula 6 - Teórica: Listas

João C. P. da Silva

Carla A. D. M. Delgado

Ana Luisa Duboc

Dept. Ciência da Computação - UFRJ