

# IEC037

## Introdução à Programação de Computadores

### Aula 11 – Listas em Python

Professor: André Carvalho

Sala:1211

E-mail:[andre@icomp.ufam.edu.br](mailto:andre@icomp.ufam.edu.br)

Página:[iccupfam.wordpress.com](http://iccupfam.wordpress.com)

# Conteúdo



Introdução a listas



Índices



Operações com listas

# Conteúdo



Introdução a listas



Índices



Operações com listas

# Listas

- Em muitos programas, precisamos manipular um **grande número** de valores:
  - ▣ Folha de pagamento
  - ▣ Medições ao longo do tempo
- Uma lista é um **conjunto de dados** organizados em ordem **sequencial**.



# Listas em Python

## :: Características

- Listas são identificadas por um **único nome**
- Cada elemento da lista é referenciado por um **índice**
- Os elementos de uma lista podem ser **modificados**
- O nome de uma lista **aponta** para o início dos elementos. Funciona apenas como **referência**.

Notas →	6.1	2.3	9.4	5.1	8.9	9.8	10	7.0	6.3	4.4
Posição:	0	1	2	3	4	5	6	7	8	9

# Listas em Python

## :: Atribuição de valores

- Uma lista é definida por uma sequência de valores, separados por **vírgulas** e envolvidos por **colchetes**.

```
<nome_lista> = [<valor1>, <valor2>, <valor3>, ...]
```

- Exemplos:

```
lista1 = [1, 2, 3, 4]  
notas = [10, 9.5, 5, 7.75, 10]  
vazio = []
```

# Conteúdo



Introdução a listas



Índices

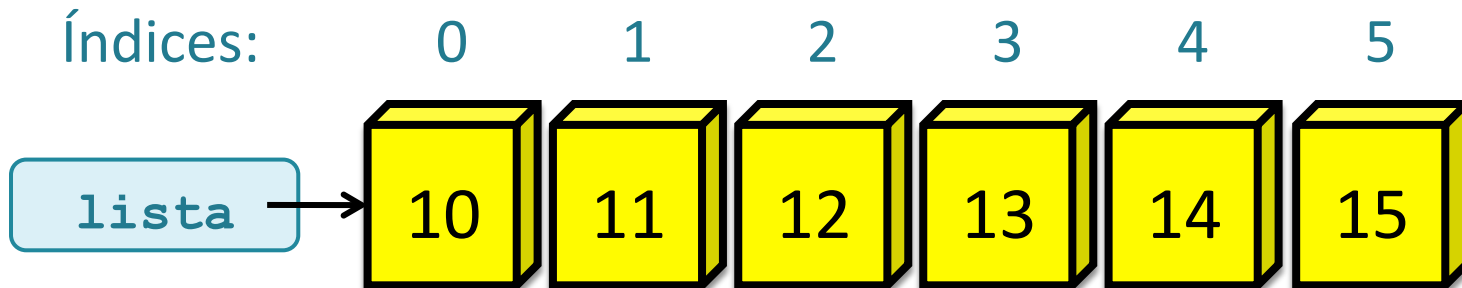


Operações com listas

# Listas em Python

## :: Índices

```
lista = [10, 11, 12, 13, 14, 15]
```



Índice do **primeiro** elemento: 0

Índice do **último** elemento: 5

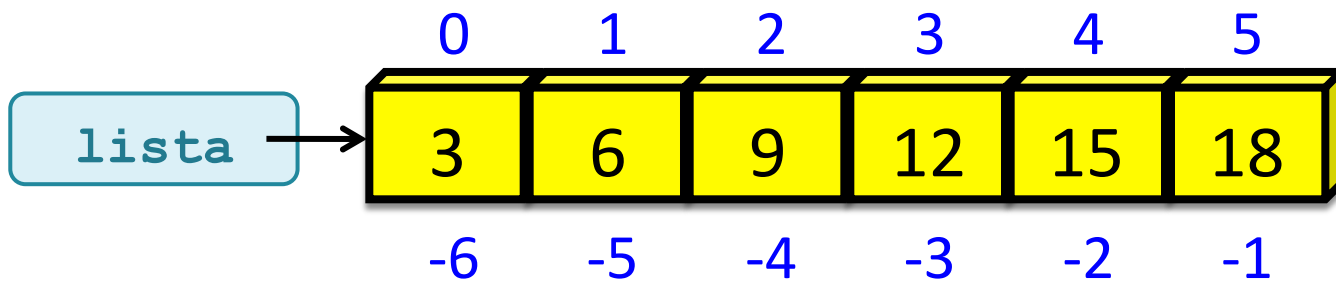
**Quantidade** de elementos: 6



# Listas em Python

## :: Índices

- Índices fora dos limites causam erro.
- Índices podem ser negativos, permitindo o acesso na ordem **inversa**.



```
lista = [3, 6, 9, 12, 15, 18]
lista[0]    # O primeiro elemento da lista: 3
lista[-1]   # O ultimo elemento da lista: 18
lista[6]    # ERRO
```



# Listas em Python

## :: Tamanho

- Para saber o **tamanho** (*length*) de uma lista, utilizamos a função **len**:

```
len(<nome_lista>)
```

```
lst1 = [3, 6, 9, 12, 15, 18]
```

```
x1 = len(lst1)      # x1 = 6
```

```
lst2 = [99]
```

```
x2 = len(lst2)     # x2 = 1
```



# Listas em Python

## :: Selecionando parte de um lista

Código	Objetivo
<code>lista[i]</code>	Seleciona o elemento de índice <code>i</code> da lista <code>lista</code>
<code>lista[i:j]</code>	Seleciona os elementos da lista <code>lista</code> cujos índices estão compreendidos entre <code>i</code> e <code>j-1</code>
<code>lista[:j]</code>	Seleciona os elementos da lista <code>lista</code> do início até o elemento <code>j-1</code> da lista
<code>lista[i:]</code>	Seleciona os elementos da lista <code>lista</code> do índice <code>i</code> até o final da lista
<code>lista[-i:]</code>	Seleciona os <code>i</code> últimos elementos da lista <code>lista</code>



# Conteúdo



Introdução a listas



Índices



Operações com listas

# Funções comuns

## :: Mínimo, Máximo, Soma

- Encontra o **menor** elemento de uma lista:

```
min(lista)
```

- Encontra o **maior** elemento de uma lista:

```
max(lista)
```

- Determina a **soma** dos elementos de uma lista:

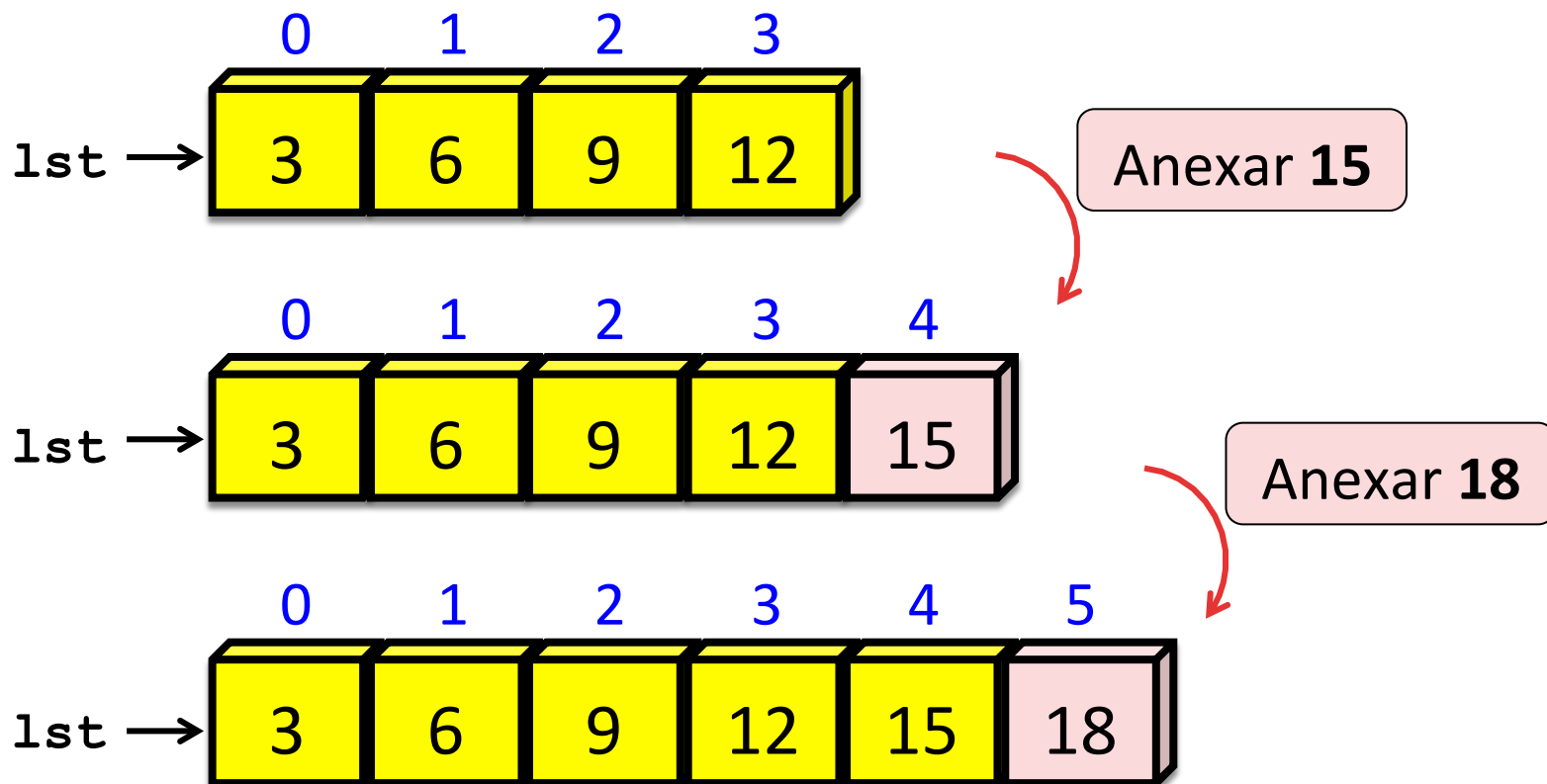
```
sum(lista)
```



# Operações com Listas

## :: Anexação

- Quando operamos com listas, anexar significa inserir um elemento ao **final** da lista:



# Operações com Listas

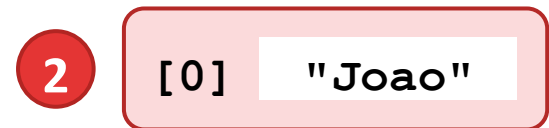
## :: Anexação – método `append`

- Em Python, a anexação é feita pelo método `append`, que tem apenas **um argumento**:

```
1 amigos = []  
2 amigos.append("Joao")  
  
amigos.append("Jose")  
3 amigos.append("Maria")  
amigos.append("Julia")
```



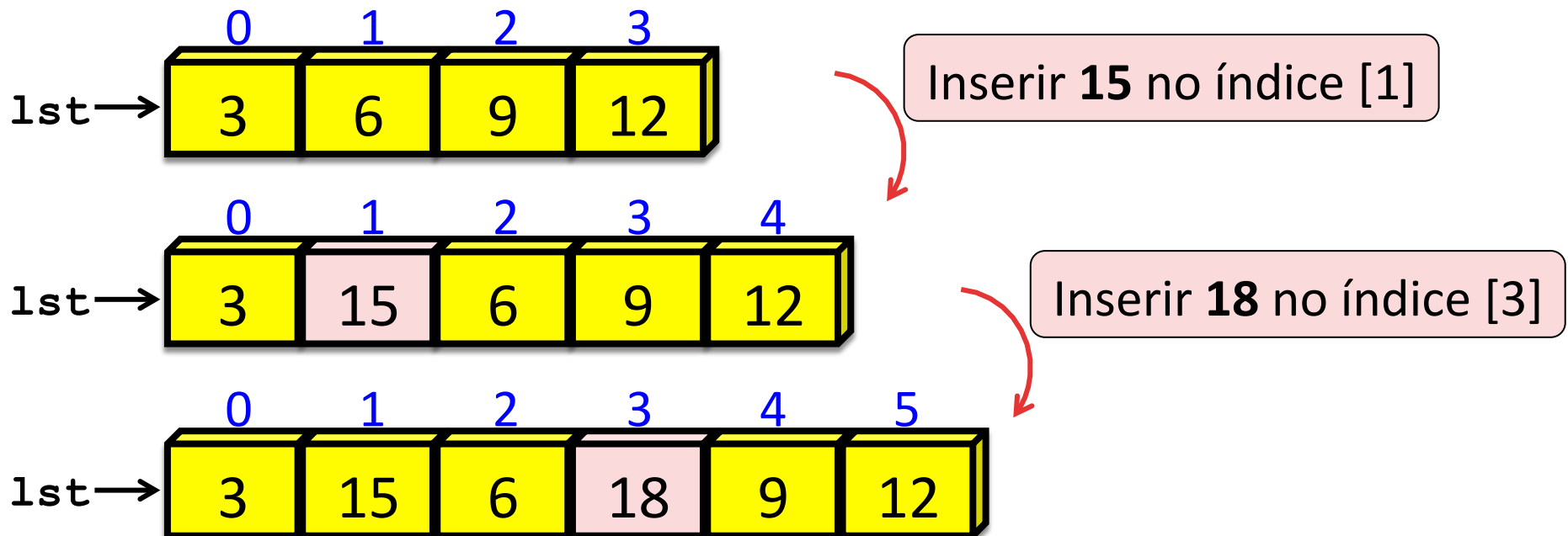
054a



# Operações com Listas

## :: Inserção

- Inserir significa introduzir um elemento em **qualquer posição** da lista.
- Portanto, além de informar **o que** desejamos inserir, devemos dizer **onde**.





# Operações com Listas

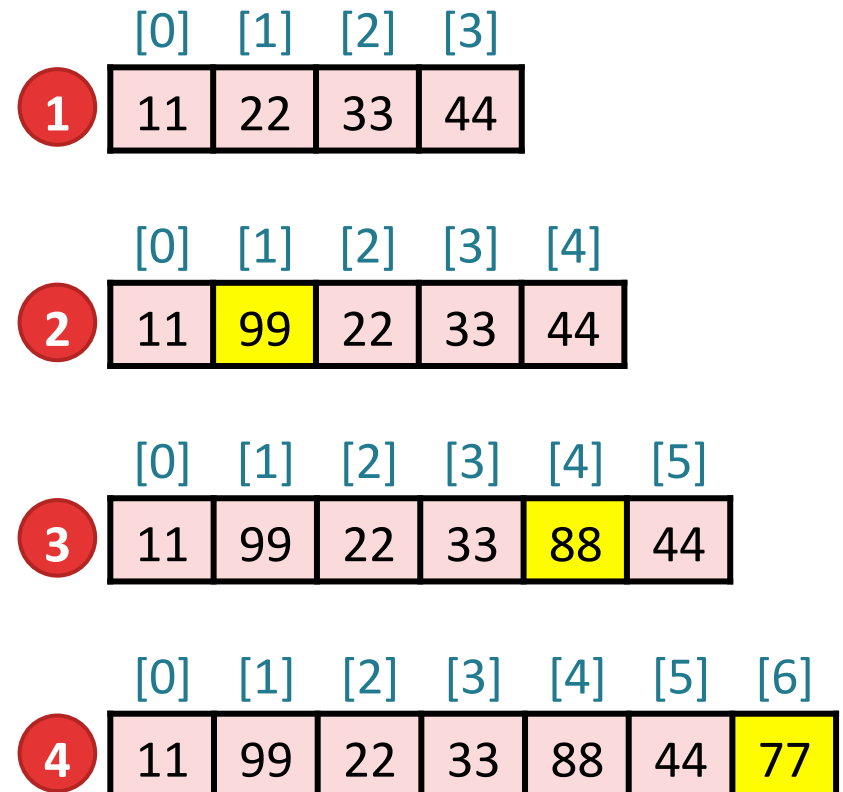
## :: Inserção – método `insert`

- Em Python, a inserção é feita pelo método `insert`, que tem dois argumentos:

```
1 lst = [11, 22, 33, 44]
2 lst.insert(1, 99)
3 lst.insert(-1, 88)
4 lst.insert(len(lst), 77)
```



054b



# Operações com Listas

## :: Anexação × Inserção

### Anexação (append)

- Insere um novo elemento no **final** da lista.
- Tem apenas **um** argumento (valor a inserir)

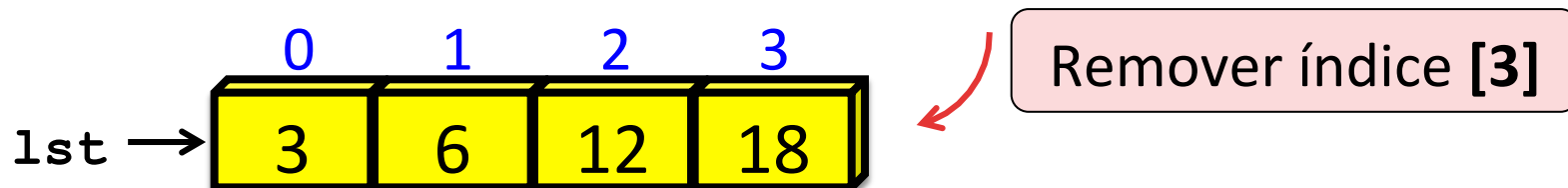
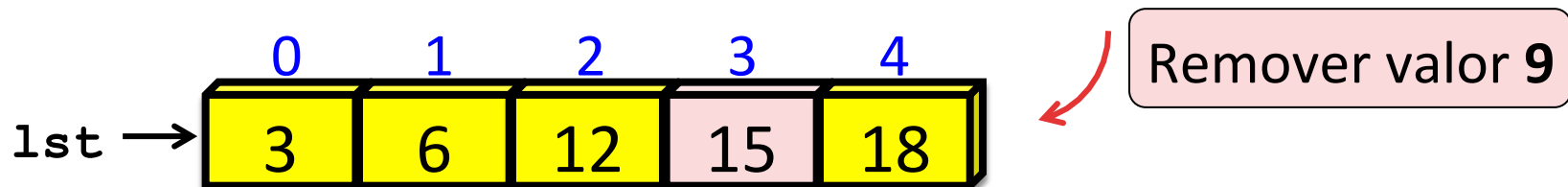
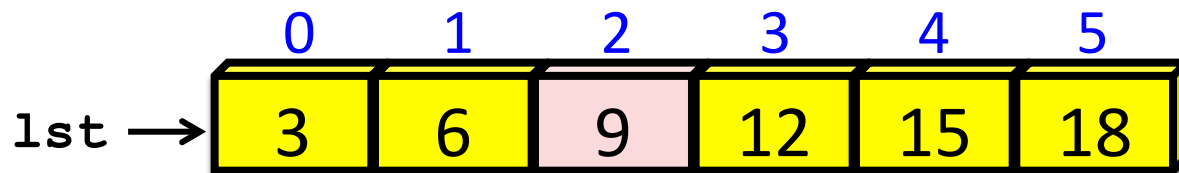
### Inserção (insert)

- Insere um novo elemento em **qualquer** posição.
- Tem **dois** argumentos (onde, valor)

# Operações com Listas

## :: Remoção

- ❑ Remove significa retirar um elemento, reacomodando-se os demais.
- ❑ O elemento a ser removido pode ser indicado pelo seu **valor** ou pelo seu **índice**.



# Operações com Listas

## :: Remoção por índice – pop

- O método **pop** remove o elemento de índice **i** informado no seu argumento.

```
<nome_lista>.pop(i)
```

```
lst = [11, 22, 33, 44, 55, 66]
lst.pop(0)      # Remove 1o. elemento da lista
lst.pop(-1)     # Remove ultimo elemento da lista
lst.pop(2)      # Remove 3o. elemento da lista
```

# Operações com Listas

## :: Remoção por índice – pop

- O tamanho da lista é reduzido em **um elemento**.
- O índice deve estar em **intervalo válido**.
- O método pop devolve o **valor** do elemento removido:

```
x = lst.pop(1)
```

- Quando o argumento não é informado, remove-se o **último** elemento da lista:

```
x = lst.pop()      # x = 66
```

# Operações com Listas

## :: Remoção por valor – **remove**

- O método **remove** remove o primeiro elemento da lista correspondente ao argumento.

```
amigos.remove("Maria")
```

- O valor a ser removido deve estar na lista, caso contrário uma exceção será gerada. Assim, antes de remover o valor, verifique se ele está na lista.

```
elemento = "Maria"  
if elemento in amigos:  
    amigos.remove(elemento)
```

# Operações com Listas

:: **remove** × **pop**

## remove

- Remove da lista o **primeiro** elemento correspondente ao **valor** informado.
- Se houver outros valores iguais, eles permanecem na lista.

## pop

- Remove da lista o elemento correspondente ao **índice** informado.



# Operações com Listas

## :: Concatenação

- Para concatenar (unir) duas ou mais listas em uma só, use o operador de soma (+):

```
nova_lst = lista1 + lista2 + lista3 + ...
```

```
meusAmigos = ["Jose", "Julia"]  
seusAmigos = ["Ana", "Joao", "Marcos"]  
  
nossosAmigos = meusAmigos + seusAmigos  
# nossosAmigos = ["Jose", "Julia", "Ana",  
"Joao", "Marcos"]
```



# Operações com Listas

## :: Replicação

- Para concatenar a mesma lista múltiplas vezes, use o operador de replicação \*

```
nova_lista = lista1 * n
```

- O inteiro **n** especifica quantas cópias da lista devem ser concatenadas

```
lst = [1, 2, 3] * 2  
# lst = [1, 2, 3, 1, 2, 3]
```

# Operações com Listas

## :: Replicação

- Um uso comum para a replicação é a inicialização de uma lista com valores fixos:

```
# inicializa com 12 zeros  
pontuacaoMensal = [0] * 12
```

# Operações com Listas

## :: Busca – operador **in**

- Para saber se um elemento está presente em uma lista, usamos o operador **in**:

```
if "Ana" in amg:  
    print("Ela eh uma amiga")
```

- O operador **in** pode ser entendido como a relação de **pertinência** entre um elemento e um conjunto (símbolo  $\in$ ).

# Operações com Listas

## :: Busca – método `index`

- ❑ Para conhecer a posição em que o elemento ocorre, usamos o método `index`.
- ❑ Se o elemento procurado não estiver na lista, haverá um erro de execução.
- ❑ Por isso, deve-se testar a lista com o operador `in` antes de chamar o método `index`.

```
amg = ["Jose", "Julia", "Ana", "Joao", "Jose"]
if "Hermenegildo" in amg:
    n = amg.index("Hermenegildo")
else:
    n = -1           # Nao encontrou
```

# Operações com Listas

## :: Busca – método `count`

- O método `count(x)` retorna o número de vezes que o elemento `x` ocorre na lista.
- Se `x` não pertencer à lista, o resultado é `zero`.

```
amg = ["Jose", "Julia", "Ana", "Joao", "Jose"]

c1 = amg.count("Jose")           # c1 = 2
c2 = amg.count("Julia")         # c2 = 1
c3 = amg.count("Hermenegildo") # c3 = 0
```

# Operações com Listas

## :: Ordenação

- Para arrumar os elementos de uma lista em ordem crescente (ou alfabética), usamos o método **sort**.

```
lst = [88, 55, 99, 44, 11, 33]
```

```
lst.sort()
```

```
# lst = [11, 33, 44, 55, 88, 99]
```

# Operações com Listas

## :: Cópia

- O nome de uma lista apenas **aponta** para o início dos elementos. Portanto, a operação abaixo apenas copia a **referência**, mas não os dados:

```
lst1 = [88, 55, 99, 44, 11, 33]
lst_copy = lst1
```

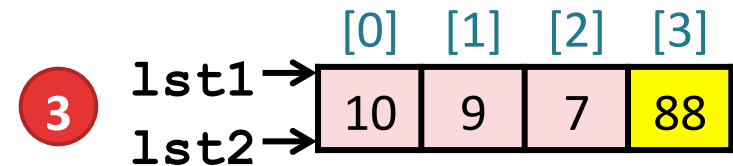
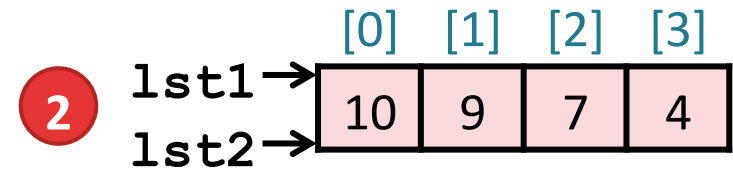
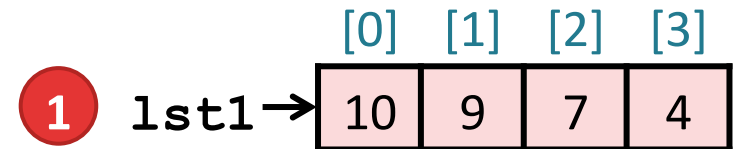
- Para copiar os dados de um lista para outra, use a função **list**:

```
lst_copy = list(lst1)
```

# Operações com Listas

## :: Cópia da referência

```
1 lst1 = [10, 9, 7, 4]
# Cópia a referencia da lista
2 lst2 = lst1
3 lst1[3] = 88
print(lst2[3]) # Imprime 88
```

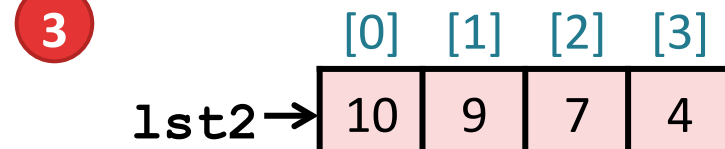
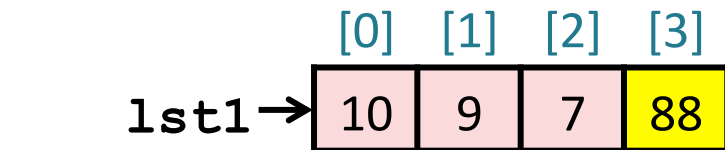
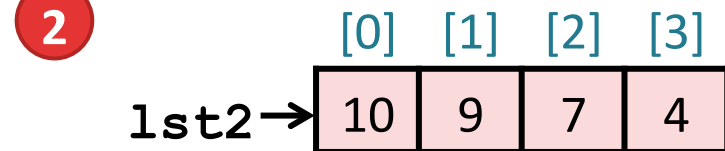
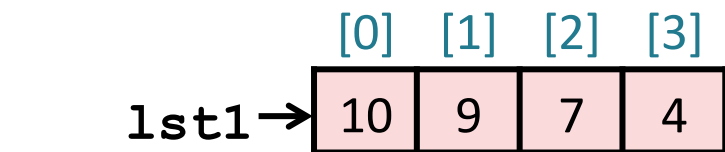
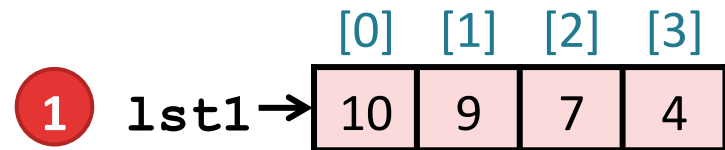




# Operações com Listas

## :: Cópia da lista

```
1 lst1 = [10, 9, 7, 4]
# Copia os dados da lista
2 lst2 = list(lst1)
3 lst1[3] = 88
print(lst2[3]) # Imprime 4
```



# Revisão

# Listas em Python

## :: Operações comuns

Operação	Descrição
<code>[val<sub>1</sub>, val<sub>2</sub>, ..., val<sub>n</sub>]</code>	Cria uma nova lista ou uma lista que contém os elementos iniciais fornecidos
<code>lst[de: para]</code>	Cria uma sublista de uma subsequência de elementos na lista <i>lst</i> começando na posição <i>de</i> até, mas não incluindo, a posição <i>para</i> . Tanto <i>de</i> quanto <i>para</i> são opcionais
<code>lst * n</code>	Cria uma nova lista replicando <i>n</i> vezes os elementos na lista <i>lst</i>
<code>lst + outraLst</code>	Cria uma nova lista concatenando os elementos de ambas as listas

# Listas em Python

## :: Funções comuns

Função	Descrição
<code>len(<i>lst</i>)</code>	Retorna o número de elementos na lista <i>lst</i>
<code>list(<i>sequencia</i>)</code>	Cria uma nova lista contendo todos os elementos da sequência
<code>sum(<i>lst</i>)</code>	Calcula a soma de valores na lista <i>lst</i>
<code>min(<i>lst</i>)</code> <code>max(<i>lst</i>)</code>	Retorna o valor mínimo e o máximo na lista <i>lst</i>

# Listas em Python

## :: Métodos comuns



Método	Descrição
<code>l.insert(pos, val)</code>	Inserir <b>val</b> na lista na posição dada. Todos os elementos na dada posição e seguintes são transferidos para baixo
<code>l.append(val)</code>	Anexa o elemento <b>val</b> ao final da lista
<code>l.remove(val)</code>	Remove o elemento <b>val</b> da lista e move todos os seguintes para cima
<code>l.pop()</code> <code>l.pop(pos)</code>	Remove o último elemento da lista ou o da posição <b>pos</b> . Todos os elementos seguintes são movidos uma posição acima.

# Listas em Python

## :: Métodos comuns

Método	Descrição
<code>l.index(val)</code>	Retorna a posição do elemento <i>val</i> na lista. O elemento deve estar na lista.
<code>l.count(val)</code>	Retorna o número de vezes que o elemento <i>val</i> ocorre na lista.
<code>l.sort()</code>	Ordena os elementos da lista em ordem crescente.

# Referências bibliográficas

-  □ Menezes, Nilo Ney Coutinho (2010). **Introdução à Programação com Python**. Editora Novatec.
-  □ Hetland, Magnus Lie (2008). **Beginning Python: From Novice to Professional**. Springer eBooks, 2ª edição. Disponível em: <http://dx.doi.org/10.1007/978-1-4302-0634-7>.
- Horstmann, Cay & Necaise, Rance D. (2013). **Python for Everyone**. John Wiley & Sons.

Dúvidas?

