



# Introdução à Programação

## Aula 09: Listas (parte 1) – Conceitos e Operações Básicas

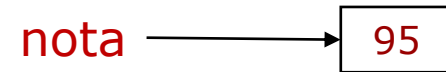
**Prof. Eduardo Corrêa**

**Data 11/04/2024**

## Variável x Lista (1/2)

- Como sabemos, uma **variável** pode armazenar apenas **um valor** de cada vez.

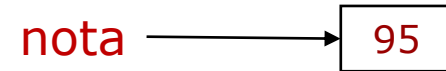
nota = 95



## Variável x Lista (2/2)

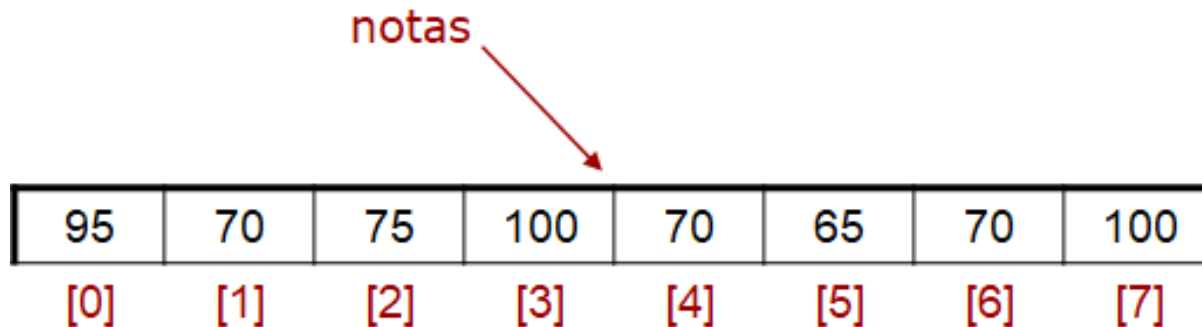
- Como sabemos, uma **variável** pode armazenar apenas **um valor** de cada vez.

nota = 95



- Hoje conheceremos a **lista**. Ela é capaz de armazenar **diversos valores** de uma vez só !!!

notas = [95, 70, 75, 100, 70, 65, 70, 100]



## Lista – Definição (1/2)

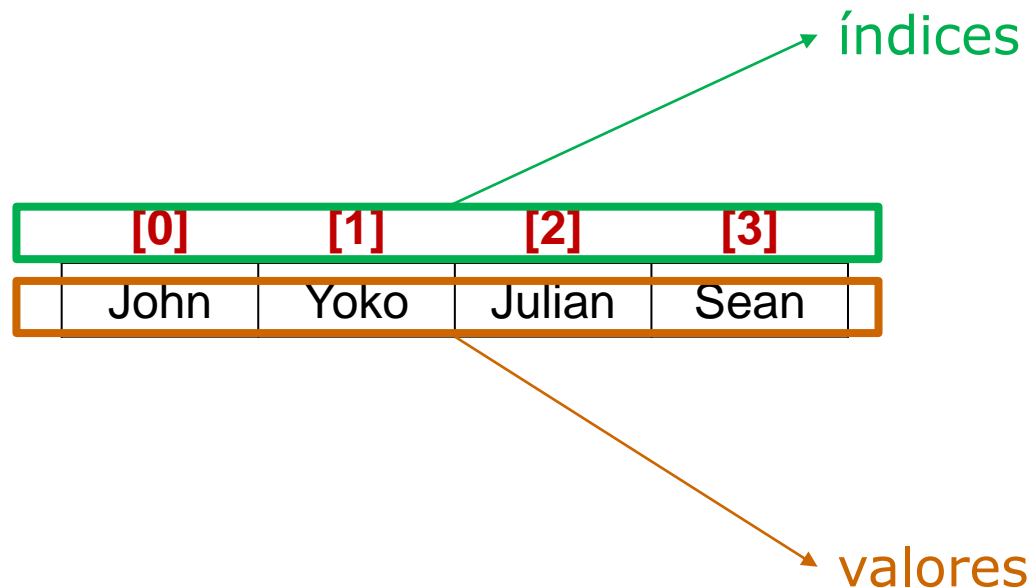
- Uma lista é uma **coleção ordenada de  $n$  elementos**.

| [0]  | [1]  | [2]    | [3]  |            |
|------|------|--------|------|------------|
| John | Yoko | Julian | Sean | <i>lst</i> |

- Cada elemento está associado a um **índice**
  - Número que indica a posição do elemento na sequência.
  - O primeiro índice é 0 e o último  $n-1$ .
  - **Ex.:** Na lista *lst*, os índices são os números 0 a 3.
- É “coleção **ordenada**” porque os elementos estão armazenados em **sequência**.

## Lista – Definição (2/2)

- Uma lista possui dois componentes:
- **ÍNDICE, POSIÇÃO** ou **SUBSCRITO**: identifica uma posição.
- **VALOR**: conteúdo armazenado numa posição.



# Conceitos Básicos (1/5)

## ■ Criação de Listas

- As listas podem ser criadas usando uma **atribuição**, de forma similar ao que fazemos com variáveis.
- Devemos especificar a sequência de valores entre colchetes, separando-os por vírgula.

```
escritores = ['Jorge Amado', 'José Saramago', 'Aldous Huxley']
```

```
sequencia_fibonacci = [0,1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
```

```
lista_vazia = []
```

```
lst_mista = ['Pen Drive', 25.90, 'Laptop', 2690] # lista mista
```

```
m = [ [1, 2, 3], [4, 5, 6] ] # lista de listas
```

## Conceitos Básicos (2/5)

### ■ Recuperando um elemento

- Para acessar um elemento, basta referenciar a posição do elemento entre colchetes []
- Essa operação é conhecida como **indexação**.
- Abaixo, a indexação é usada para recuperarmos o valor contido na posição 0 e depois o contido na posição 3

```
lst = ["John", "Yoko", "Julian", "Sean"]
```

```
primeiro = lst[0]           # 'John'  
ultimo = lst[3]             # 'Sean'
```

```
print(primeiro, ultimo)
```

Saída:

```
John Sean
```

## Conceitos Básicos (3/5)

### ■ Modificando um elemento

- Também precisamos indexar para alterar o valor armazenado em uma posição.
- Basta referenciar a posição desejada entre colchetes [] e utilizar normalmente o operador de atribuição ( = ).

```
lst = ["John", "Yoko", "Julian", "Sean"]
```

```
lst[1] = "Yoko Ono"  
print(lst)
```

```
lst[0] = "John Lennon"  
print(lst)
```

Saída:

```
['John', 'Yoko Ono', 'Julian', 'Sean']  
['John Lennon', 'Yoko Ono', 'Julian', 'Sean']
```



## Conceitos Básicos (4/5)

- Note ainda que a lista **pode ter valores repetidos**
- Na lista **notas**:
  - O valor 70 aparece nas posições 1, 4 e 6
  - O valor 100 nas posições 3 e 7.

```
notas = [95, 70, 75, 100, 70, 65, 70, 100]
```

notas

|     |     |     |     |     |     |     |     |
|-----|-----|-----|-----|-----|-----|-----|-----|
| 95  | 70  | 75  | 100 | 70  | 65  | 70  | 100 |
| [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] |

## Conceitos Básicos (5/5)

- Em resumo, uma lista funciona de forma similar a uma variável.
- No entanto, a variável é uma “caixinha” ou “gaveta” que só pode armazenar um valor de cada vez.
- Uma lista pode guardar muitos valores de uma vez porque é um “conjunto de gavetas” (cada uma dessas gavetas pode ser acessada através de seu índice).
- A seguir um exemplo de programa contendo a criação de uma lista, e depois o acesso e modificação de alguns de seus elementos.

## Lista – Exemplo (1/7)

```
a = [-45,
      6,
      0,
      72,
      1543,
      -89,
      0,
      62,
      20,
      -1]
```

```
# impressão dos elementos de índice 2 e 6
print('a[2] = ',a[2]); print('a[6] = ',a[6])
```

```
# novas atribuições
```

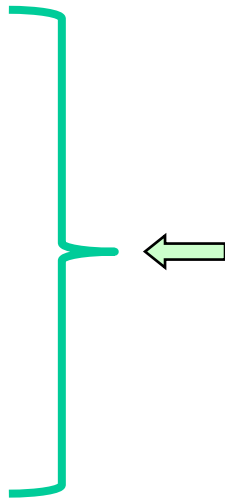
```
a[2] = a[2] + 1    # agora a[2] vale 1
a[6] = a[4]        # agora a[6] vale 1543
```

```
# impressão dos elementos de índice 2 e 6
print('a[2] = ',a[2]); print('a[6] = ',a[6])
```

- O programa Python ao lado:
- Declara uma lista de inteiros de 10 posições chamada **a**.
- Imprime o conteúdo de algumas posições da lista na tela.
- Modifica alguns dos valores inicialmente atribuídos.
- Imprime o conteúdo das posições modificadas.
- Vamos examinar a sua execução!

## Lista – Exemplo (2/7)

```
a = [-45,  
     6,  
     0,  
     72,  
     1543,  
     -89,  
     0,  
     62,  
     20,  
     -1]
```



```
# impressão dos elementos de índice 2 e 6  
print('a[2] = ',a[2]); print('a[6] = ',a[6])
```

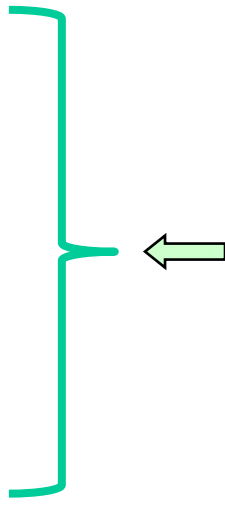
```
# novas atribuições  
a[2] = a[2] + 1   # agora a[2] vale 1  
a[6] = a[4]       # agora a[6] vale 1543
```

```
# impressão dos elementos de índice 2 e 6  
print('a[2] = ',a[2]); print('a[6] = ',a[6])
```

- Neste ponto o computador **cria em memória** a lista **a**, que possui 10 posições/valores.
- Na hora de criar a lista você pode:
  - Colocar os elementos em linhas separadas.
  - Ou todos os mesma linha, como vimos anteriormente.
- O importante é abrir colchetes, separar os elementos por vírgula e, no fim, fechar colchetes.

## Lista – Exemplo (3/7)

```
a = [-45,  
     6,  
     0,  
     72,  
     1543,  
     -89,  
     0,  
     62,  
     20,  
     -1]
```



```
# impressão dos elementos de índice 2 e 6  
print('a[2] = ',a[2]); print('a[6] = ',a[6])
```

```
# novas atribuições
```

```
a[2] = a[2] + 1   # agora a[2] vale 1  
a[6] = a[4]       # agora a[6] vale 1543
```

```
# impressão dos elementos de índice 2 e 6  
print('a[2] = ',a[2]); print('a[6] = ',a[6])
```

- Nesse caso, todas as 10 posições receberam valores inteiros.
- Mas uma lista pode ter elementos de tipos diferentes.

a →

|   |      |
|---|------|
| 0 | -45  |
| 1 | 6    |
| 2 | 0    |
| 3 | 72   |
| 4 | 1543 |
| 5 | -89  |
| 6 | 0    |
| 7 | 62   |
| 8 | 20   |
| 9 | -1   |

## Lista – Exemplo (4/7)

```
a = [-45,  
     6,  
     0,  
     72,  
     1543,  
     -89,  
     0,  
     62,  
     20,  
     -1]
```

```
# impressão dos elementos de índice 2 e 6
```

```
print('a[2] = ',a[2]); print('a[6] = ',a[6])
```

```
# novas atribuições
```

```
a[2] = a[2] + 1 # agora a[2] vale 1
```

```
a[6] = a[4]     # agora a[6] vale 1543
```

```
# impressão dos elementos de índice 2 e 6
```

```
print('a[2] = ',a[2]); print('a[6] = ',a[6])
```

- Os dois comandos destacados imprimem na tela:

a[2] que tem valor 0  
a[6] que tem valor 0

a →

|   |      |
|---|------|
| 0 | -45  |
| 1 | 6    |
| 2 | 0    |
| 3 | 72   |
| 4 | 1543 |
| 5 | -89  |
| 6 | 0    |
| 7 | 62   |
| 8 | 20   |
| 9 | -1   |

## Lista – Exemplo (5/7)

```
a = [-45,
      6,
      0,
      72,
      1543,
      -89,
      0,
      62,
      20,
      -1]
```

```
# impressão dos elementos de índice 2 e 6
print('a[2] = ',a[2]); print('a[6] = ',a[6])
```

```
# novas atribuições
```

```
a[2] = a[2] + 1  # agora a[2] vale 1
a[6] = a[4]      # agora a[6] vale 1543
```

```
# impressão dos elementos de índice 2 e 6
print('a[2] = ',a[2]); print('a[6] = ',a[6])
```

- O valor do elemento de índice 2 foi modificado.
- Antes valia 0 e agora vale 1
- Note que os valores das outras posições são preservados.

a →

|   |                |
|---|----------------|
| 0 | -45            |
| 1 | 6              |
| 2 | <del>0</del> 1 |
| 3 | 72             |
| 4 | 1543           |
| 5 | -89            |
| 6 | 0              |
| 7 | 62             |
| 8 | 20             |
| 9 | -1             |

## Lista – Exemplo (6/7)

```
a = [-45,  
     6,  
     0,  
     72,  
     1543,  
     -89,  
     0,  
     62,  
     20,  
     -1]
```

```
# impressão dos elementos de índice 2 e 6  
print('a[2] = ',a[2]); print('a[6] = ',a[6])
```

```
# novas atribuições
```

```
a[2] = a[2] + 1 # agora a[2] vale 1
```

```
a[6] = a[4] ← # agora a[6] vale 1543
```

```
# impressão dos elementos de índice 2 e 6  
print('a[2] = ',a[2]); print('a[6] = ',a[6])
```

- Este comando faz com que o valor do elemento 6 de **a**, passe a ser igual ao valor de seu elemento 4.

a →

|   |                   |
|---|-------------------|
| 0 | -45               |
| 1 | 6                 |
| 2 | 1                 |
| 3 | 72                |
| 4 | 1543              |
| 5 | -89               |
| 6 | <del>0</del> 1543 |
| 7 | 62                |
| 8 | 20                |
| 9 | -1                |



## Lista – Exemplo (7/7)

```
a = [-45,
      6,
      0,
      72,
      1543,
      -89,
      0,
      62,
      20,
      -1]
```

```
# impressão dos elementos de índice 2 e 6
print('a[2] = ',a[2]); print('a[6] = ',a[6])
```

```
# novas atribuições
```

```
a[2] = a[2] + 1   # agora a[2] vale 1
```

```
a[6] = a[4]       # agora a[6] vale 1543
```

```
# impressão dos elementos de índice 2 e 6
```

```
print('a[2] = ',a[2]); print('a[6] = ',a[6])
```

- Os dois comandos destacados imprimem na tela:

a[2] que tem valor 1  
a[6] que tem valor 1543

a →

|   |      |
|---|------|
| 0 | -45  |
| 1 | 6    |
| 2 | 1    |
| 3 | 72   |
| 4 | 1543 |
| 5 | -89  |
| 6 | 1543 |
| 7 | 62   |
| 8 | 20   |
| 9 | -1   |

## Lista – Exemplo (7/7)

```
1 a = [-45,  
2     6,  
3     0,  
4     72,  
5     1543,  
6     -89,  
7     0,  
8     62,  
9     20,  
10    -1]  
11  
12 # impressão dos elementos de índice 2 e 6  
13 print('a[2] = ',a[2]); print('a[6] = ',a[6])  
14  
15 # novas atribuições  
16 a[2] = a[2] + 1    # agora a[2] vale 1  
17 a[6] = a[4]        # agora a[6] vale 1543  
18  
19 # impressão dos elementos de índice 2 e 6  
20 print('a[2] = ',a[2]); print('a[6] = ',a[6])  
21
```

```
Shell ×  
Python 3.10.4 (C:\Pyt  
>>> %Run listas.py  
  
a[2] = 0  
a[6] = 0  
a[2] = 1  
a[6] = 1543
```

# Operações Básicas (1/11)

## ■ INDEXAÇÃO

- A operação que consiste em acessar um elemento de uma lista, seja para consulta-lo ou modifica-lo, é conhecida como **indexação**.
- `carros[0]`: indexa o primeiro elemento da lista, que, conforme ilustra a figura abaixo, é "HB20".
- `carros[4]`: indexa o quinto elemento, cujo valor é igual a "UNO".
- `carros[6]`: esta indexação é um **erro !! (IndexError)**  
A lista tem apenas 6 posições (0 a 5). Então não existe o elemento de índice 6.

carros

|      |       |     |       |     |       |
|------|-------|-----|-------|-----|-------|
| HB20 | FUSCA | UP  | LOGAN | UNO | YARIS |
| [0]  | [1]   | [2] | [3]   | [4] | [5]   |

# Operações Básicas (2/11)

## ■ INDEXAÇÃO NEGATIVA

- A indexação também pode ser feita usando índices negativos. Neste caso, **-1** serve para indexar o **último índice** da lista, **-2** o **penúltimo** e assim sucessivamente
- `carros[-1]`: indexa o último elemento da lista, cujo valor é "YARIS". Também é possível indexar esse elemento fazendo `carros[5]`.
- `carros[-2]`: indexa o penúltimo elemento da lista, cujo valor é igual a "UNO". Também é possível indexar esse elemento por `carros[4]`.
- `carros[-7]`: esta indexação é um **erro!!**  
A lista tem apenas 6 posições. O primeiro é indexado por `carros[-6]` ou `carros[0]`.

carros

|      |       |      |       |      |       |
|------|-------|------|-------|------|-------|
| HB20 | FUSCA | UP   | LOGAN | UNO  | YARIS |
| [0]  | [1]   | [2]  | [3]   | [4]  | [5]   |
| [-6] | [-5]  | [-4] | [-3]  | [-2] | [-1]  |

- O operador **in** pode ser utilizado para verificar se um valor pertence à lista (se está armazenado em algum índice)
- Basta indicar o valor e depois escrever **in** e o nome da lista.
- É como se estivéssemos perguntando “O valor x está nessa lista?”.
- O resultado será True ou False.
- **'FUSCA' in carros** retorna True
- **'BMW' in carros** retorna False

|      |       |     |       |     |       |
|------|-------|-----|-------|-----|-------|
| HB20 | FUSCA | UP  | LOGAN | UNO | YARIS |
| [0]  | [1]   | [2] | [3]   | [4] | [5]   |

## Operações Básicas (4/11)

- **NÃO PERTENCE**
- Já o **not in** faz o contrário
- Nesse caso, a pergunta é: "O valor x não está nessa lista?".
  - O resultado será True ou False.
- 'FUSCA' **not in** carros                      retorna False
- 'BMW' **not in** carros                         retorna True

carros

|      |       |     |       |     |       |
|------|-------|-----|-------|-----|-------|
| HB20 | FUSCA | UP  | LOGAN | UNO | YARIS |
| [0]  | [1]   | [2] | [3]   | [4] | [5]   |

# Operações Básicas (5/11)

- **Tamanho, Soma, Mínimo, Máximo** - as **funções pré-definidas** (*built-in functions*) são aquelas que podem ser utilizadas sem que seja preciso importar um módulo. Alguns exemplos: **int()**, **str()** e **round()**.
- Agora vamos conhecer **4 funções** pré-definidas especialmente **úteis** para **listas** e outras coleções.

**len(lst)**: retorna o número de elementos atualmente armazenados na lista *lst*. Muito importante, pois é usada na iteração por índice, como veremos daqui a pouco. O número de elementos também é chamado de **comprimento** da lista.

**sum(lst)**: retorna a soma dos elementos de *lst* (serve apenas para listas com elementos numéricos)

**min(lst)**: retorna o menor valor de *lst*

**max(lst)**: retorna o maior valor de *lst*.

## Operações Básicas (6/11)

- **EXERCÍCIO RESOLVIDO:** abaixo são apresentados os resultados do teste de QI de oito participantes de um estudo. Faça um programa para estruturar os valores de QI em uma lista e retornar o QI máximo, mínimo e médio.

| Participante | QI  |
|--------------|-----|
| Asif         | 126 |
| Bill         | 100 |
| Bob          | 130 |
| Jim          | 92  |
| Liu          | 120 |
| Joan         | 99  |
| Rakesh       | 125 |
| Zangh        | 72  |



# Operações Básicas (7/11)

## ■ Solução

```
lst_qi = [126, 100, 130, 92, 120, 99, 125, 72]

print("resultados do teste de QI: ", lst_qi)
print("maior: ", max(lst_qi))
print("menor: ", min(lst_qi))
print("soma: ", sum(lst_qi))
print("media: ", sum(lst_qi) / len(lst_qi))
```

### Saída:

```
resultados do teste de QI: [125, 92, 72, 126, 120, 99, 130, 100]
maior: 130
menor: 72
soma: 864
media: 108.0
```

# Operações Básicas (8/11)

- **ITERAÇÃO**
- Iteração é a operação que consiste em **percorrer** todos ou parte dos elementos de uma lista, **um por um**, em sequência.
- É implementada através da construção de um laço com o comando **for** (*também é possível usar o while, porém quase sempre é mais simples iterar com o for*).
- Há 2 formas de fazer:
  - Iterar pelos elementos
  - Iterar pelos índices

# Operações Básicas (9/11)

- **ITERAÇÃO FORMA 1 – PELOS ELEMENTOS**
- A forma mais simples de iterar é utilizando a estrutura **for-coleção**, para realizar a iteração diretamente sobre os elementos.

```
for v in coleção:  
    comando1  
    comando2  
    ...  
    comandon
```

- O comando **for** vai percorrer sequencialmente cada elemento da lista que for especificada como “coleção”
- A cada iteração, a variável **v** receberá como valor um dos elementos da coleção.

# Operações Básicas (10/11)

## ■ ITERAÇÃO FORMA 1 – PELOS ELEMENTOS

```
lst = ["John", "Yoko", "Julian", "Sean"]
```

```
for pessoa in lst:  
    print(pessoa)
```



```
John  
Yoko  
Julian  
Sean
```

# Operações Básicas (11/11)

- **ITERAÇÃO FORMA 2 – COM BASE NOS ÍNDICES**
- Uma segunda maneira de iterar por uma lista e fazer não diretamente sobre os seus elementos, mas sim com base nos índices da lista.
- Para alguns problemas, pode ser necessário fazer assim!
- Isto pode ser implementado como o auxílio das funções `range()` e `len()`. Veja o exemplo a seguir:

```
for k in range(len(lst)):  
    print("elemento {} = {}".format(k, lst[k]))
```



```
elemento 0 = John  
elemento 1 = Yoko  
elemento 2 = Julian  
elemento 3 = Sean
```

# Métodos de Lista (1/6)

- **O que são métodos?**
- Quando você cria uma lista em um programa, você “ganha de presente” uma série de **métodos** que podem ser utilizados para executar operações sobre a lista.
- Na realidade, toda lista é uma coleção que contém **dados** e **métodos**.
- **Dados** → valores dos elementos armazenados.
- **Métodos** → funções presentes “dentro” da lista e que sempre estão disponíveis para serem utilizadas pelo programador.

## Métodos de Lista (2/6)

- **O que são métodos?**
- Assim como ocorre com qualquer função (como `len()`, `math.cos()` etc.), um método pode receber zero ou mais argumentos e retornará um valor.
- A diferença é que, para chamar um método, devemos adicionar um ponto (".") e o nome do método ao final do nome de uma lista.
  - Isto é chamado de "invocar" o método... invocar significa simplesmente "executar"
  - A sintaxe para invocar um método é resumida abaixo.

**lista**.método (*argumentos*)

## Métodos de Lista (3/6)

- **Na sintaxe:** `lista.método(argumentos)`
- - “lista” corresponde ao nome da lista;
  - “método” ao nome do método que desejamos executar;
  - “argumentos” são os argumentos exigidos por tal método.
- Cada método executa uma **ação diferente**, portanto exige parâmetros diferentes.
  - Alguns métodos servem para **modificar** uma lista (ex.: inserir ou remover elementos).
  - Outros para **explorá-la** de alguma forma (ex.: contar o número de ocorrências de um determinado valor).
- **Nesta aula**, apresentaremos apenas os **métodos básicos para modificar listas**, deixando os demais para outras aulas



# Métodos de Lista (4/6)

## ■ Métodos Básicos para **Modificar** uma Lista

- `lista.append(x)`: insere elemento `x` no final da lista.
- `lista.insert(i, x)`: insere elemento `x` na posição `i`.
- `lista.pop(i)`: remove um elemento no final (caso `i` não seja especificado) ou da posição `i` da lista (caso `i` seja especificado). O método retorna o elemento removido.
- `lista.remove(x)`: remove o primeiro elemento que tiver o valor `x`.
- `lista.clear()`: esvazia a lista
- `lista.extend(lista2)`: concatena os elementos da `lista2` (ou outro iterável) ao final de `lista`. Também pode-se usar `lista += lista2`.

## Métodos de Lista (5/6)

- **Exemplo - métodos básicos para **modificar** uma lista**
  - O programa do slide a seguir exemplifica a utilização prática dos métodos para modificar listas.
  - Analise-o com calma.
  - Para facilitar a sua compreensão, colocamos comentários ao lado de cada linha que apresentam o conteúdo da lista após a invocação do método.

# Métodos de Lista (6/6)

**# IMPORTANTE:** os *métodos modificam a lista* sem que seja preciso  
# realizar uma atribuição com o operador “=”

`numeros = [5, 10]`

`numeros.append(20)` # insere 20 no final: [5, 10, 20]

`numeros.append(10)` # insere 10 no final: [5, 10, 20, 10]

`numeros.insert(2,15)` # insere 15 na posição 2: [5, 10, 15, 20, 10]

`numeros.insert(0,10)` # insere 10 na posição 0: [10, 5, 10, 15, 20, 10]

`numeros.pop()` # remove o último elemento: [10, 5, 10, 15, 20]

`numeros.pop(3)` # remove o quarto elemento: [10, 5, 10, 20]

`numeros.remove(10)` # remove o primeiro 10: [5, 10, 20]

`numeros.extend([40, 50])` # estende a lista: [5, 10, 20, 40, 50]

`numeros.clear()` # esvazia a lista: [ ]

# Exercícios

- Os exercícios propostos devem ser feitos considerando que **a(s) lista(s)** a serem processadas **já existe(m) em memória**.
- Nas primeiras linhas de código, defina uma ou mais listas que sejam necessárias para testar seu programa.
- Entretanto, tenha em mente que **o programa deve** ser feito de modo a **funcionar para qualquer lista** (não apenas a do exemplo que você bolou).
- Ou seja, para qualquer lista independente do conteúdo
- Quando o enunciado disser “dada uma lista com  $n$  elementos”, o programa terá que funcionar para qualquer  $n$ , ou seja, para qualquer lista independente de seu comprimento.

# Exercícios

- (1) Considere uma lista em memória com dez números reais. Faça um programa que calcule e imprima:
  - A média dos valores.
  - A quantidade de números negativos
  - A soma dos números positivos dessa lista.
  
- (2) Dada uma lista com 3 elementos, faça um programa que utilize as funções `max()`, `min()` e `sum()` para dizer qual é o menor, o maior e o do meio.
  
- (3)- Faça um programa que imprima uma lista com 7 elementos antes e depois dos elementos de índice 5, 4 e 0 serem removidos (nesta ordem). Veja o exemplo abaixo:

`letras = ['A', 'B', 'C', 'D', 'E', 'F', 'G']` → `letras = ['B', 'C', 'D', 'G']`  
(lista original) (lista depois da remoção)

## Exercícios

- (4) Crie um programa que altere uma lista  $w$  de  $n$  elementos da seguinte forma:
  - Os elementos de índice par deverão ser multiplicados por 3
  - Os elementos de índice ímpar deverão ser divididos por 2

|   |   |   |   |     |
|---|---|---|---|-----|
| 3 | 8 | 4 | 2 | ... |
|---|---|---|---|-----|

(lista  $w$  original)

|   |   |    |   |     |
|---|---|----|---|-----|
| 9 | 4 | 12 | 1 | ... |
|---|---|----|---|-----|

(lista  $w$  depois da alteração)

- (5)- Dada a lista mista  $a$  com  $n$  elementos, **crie uma nova lista  $b$**  onde os elementos do tipo string tenham sido substituídos pelo valor -999. Exemplo:

$a = ["abcd", "acme", 1000, "xyz", 45.50, -1] \rightarrow b = [-999, -999, 1000, -999, 45.50, -1]$

**Obs.:** para testar o tipo de um valor  $x$  você pode usar: **if type(x)==tipo**, onde tipo é str, float, int, etc. Exemplo: **if type(x)==str**