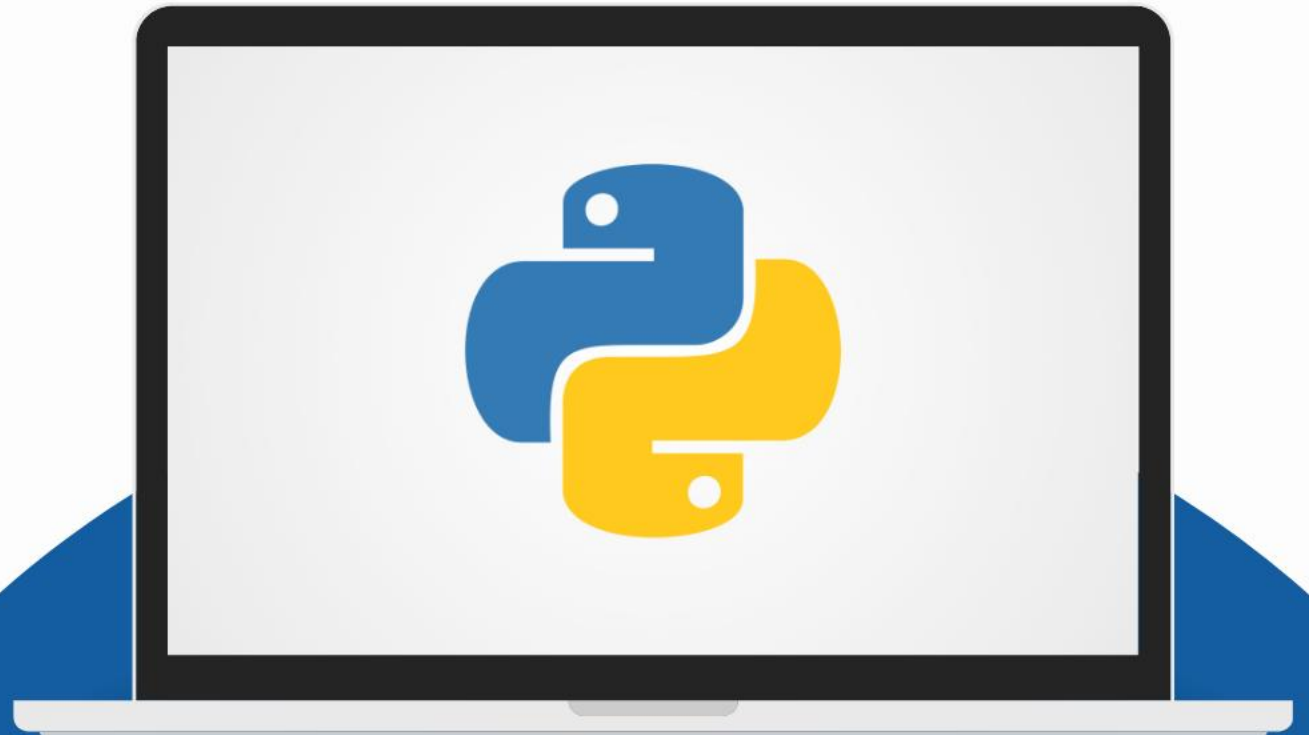




Aula #8:

Listas: funções e métodos

Prof. Dr. Luiz Álvaro de Oliveira Júnior





Objetivos desta aula

- Conceituar lista compreender sua estrutura;
- Conhecer funções e métodos para trabalhar com listas;
- Aplicar funções e métodos para resolver problemas envolvendo listas;

Sumário

▶ Listas	04
▶ Métodos para listas	06
▶ Funções para listas	17
▶ Acessando os elementos de uma lista	24
▶ Exercícios	29



Listas

Uma **lista** é uma estrutura de dados nativa do Python que permite armazenar vários valores em uma única variável. Suas características são:

Ordenadas: a ordem dos elementos é preservada.

Indexada: cada elemento tem sua posição.

Mutáveis: é possível modificar, adicionar e remover elementos após a criação da lista.

Multi-tipos: é possível armazenar elementos de tipos diferentes na mesma lista.

Listas

Usamos listas principalmente **se precisamos de flexibilidade** para **armazenar diferentes tipos de dados** e não precisamos realizar cálculos matemáticos, ainda que simples.

Listas são estruturas de dados **iteráveis**, ou seja, podem ter seus elementos percorridos por alguma estrutura de repetição.

Métodos para listas

Os principais métodos para listas são mostrados a seguir:

append(): adiciona um elemento por vez ao final da lista.

Sintaxe: **nome_da_lista.append(valor)**

Exemplo:



```
linguagens = ["C++", "Java"]  
cursos = [ ]  
linguagens.append("Python")  
print(linguagens)  
cursos.append(38)  
print(cursos)
```



```
["C++", "Java", "Python"]  
[38]
```

Métodos para listas

insert(): insere um elemento em uma posição especificada.

Sintaxe: **nome_da_lista.insert(posição, valor)**

Exemplo:

```
linguagens = ["C++", "Java"]
cursos = [37, 38, 118]
linguagens.insert(1, "Python")
print(linguagens)
cursos.insert(0, 25)
print(cursos)
```

Saída

```
["C++", "Python", "Java"]
[25, 37, 38, 118]
```

Observação: **posição** é sempre um número inteiro.

Métodos para listas

remove(): remove a primeira ocorrência de um elemento na lista.

Sintaxe: **nome_da_lista.remove(valor)**

Exemplo:

```
linguagens = ["C++", "Python", "Java"]
cursos = [25, 29, 37, 38, 29, 118]
linguagens.remove("Java")
print(linguagens)
cursos.remove(29)
print(cursos)
```

Saída

```
["C++", "Python"]
[25, 37, 38, 29, 118]
```

Observação: se a lista tiver **mais de uma ocorrência** do mesmo valor, **somente a primeira** será removida.

Métodos para listas

pop(): remove o elemento de uma posição indicada e o retorna.

Sintaxe: **variavel** = **nome_da_lista**.pop(**posição**)

Exemplo:

```
linguagens = ["C++", "Python", "Java"]
cursos = [25, 29, 37, 38, 29, 118]
progr = linguagens.pop(0)
print(linguagens)
cursos.pop(4)
print(cursos)
print("Programa = ", progr)
```

Saída

```
["Python", "Java"]
[25, 29, 37, 38, 118]
Programa = "C++"
```

Observação: **posição** é sempre um número inteiro e começa a contar do zero.

Métodos para listas

clear(): remove todos os elementos de uma lista, deixando-a vazia.

Sintaxe: **nome_da_lista.clear()**

Exemplo:



```
linguagens = ["C++", "Python", "Java"]
cursos = [25, 29, 37, 38, 29, 118]
progr = linguagens.clear()
print("Linguagens = ", linguagens)
cursos.clear()
print("Cursos = ", cursos)
```



Saída

```
Linguagens = [ ]
Cursos = [ ]
```

Métodos para listas

index(): retorna a posição da **primeira ocorrência** de um elemento da lista.

Sintaxe: **variavel** = **nome_da_lista.index(valor)**

Exemplo:

```
linguagens = ["C++", "Python", "Java"]
cursos = [25, 29, 37, 38, 29, 118]
x = linguagens.index("Python")
print("Posição Python = ", x)
cod_meca = cursos.index(118)
print("Cursos = ", cod_meca)
```

Saída

```
Posição Python = 1
Cursos = 5
```

Observação: se houver termos repetidos, retornará somente a posição da primeira ocorrência.

Métodos para listas

count(): conta e retorna o número de vezes que um determinado elemento aparece na lista.

Sintaxe: **variavel** = **nome_da_lista.count(valor)**

Exemplo:

```
linguagens = ["Java", "Python", "Java"]
cursos = [25, 37, 37, 38, 118, 37]
x = linguagens.count("Java")
print("Contagem = ", x)
cod_prod = cursos.count(37)
print("Cursos = ", cod_prod)
```

Saída

```
Contagem = 2
Cursos = 3
```

Métodos para listas

sort(): ordena os elementos em ordem crescente (alfabética ou numérica).

Sintaxe: **nome_da_lista.sort()**

Exemplo:



```
linguagens = ["Java", "Python", "C++"]  
cursos = [25, 37, 38, 118, 37]  
linguagens.sort()  
print(linguagens)  
cursos.sort()  
print(cursos)
```



```
["C++", "Java", "Python"]  
[25, 37, 37, 38, 118]
```

Métodos para listas

reverse(): inverte a ordem dos elementos.

Sintaxe: **nome_da_lista.reverse()**

Exemplo:



```
linguagens = ["Java", "Python", "C++"]  
cursos = [25, 37, 38, 118, 37]  
linguagens.reverse()  
print(linguagens)  
cursos.reverse()  
print(cursos)
```



```
["C++", "Python", "Java"]  
[37, 118, 38, 37, 25]
```

Observação: não ordena em ordem decrescente, apenas inverte a ordem.

Métodos para listas

copy(): cria uma cópia da lista.

Sintaxe: **nome_novo = nome_da_lista.copy()**

Exemplo:



```
linguagens = ["Java", "Python", "C++"]  
cursos = [25, 37, 38, 118]  
lista_nova = linguagens.copy()  
print("Lista nova = ", lista_nova)  
cursos_nova = cursos.copy()  
print("Cursos_nova = ", cursos_nova)
```



```
["Java", "Python", "C++"]  
[25, 37, 38, 118]
```

Métodos para listas

extend(): adiciona todos os elementos de um iterável a uma lista existente.

Sintaxe: **nome_da_lista.extend(iteravel)**

Exemplo:



```
linguagens = ["Java"]  
cursos = [25, 37]  
linguagens.extend(["Python", "C++"])  
print(linguagens)  
cursos.extend([38, 118])  
print(cursos)
```



```
["Java", "Python", "C++"]  
[25, 37, 38, 118]
```

Observação: admite apenas um iterável que pode conter um ou mais de um elemento.

Funções para listas

len(): retorna a quantidade de elementos dentro da lista

Sintaxe: **variavel** = **len**(**nome_da_lista**)

Exemplo:

```
linguagens = ["Java", "Python", "C++", "Kotlin"]  
x = len(linguagens)  
print(x)
```

Saída

4

Observação: Já vimos e usamos essa função quando estudamos as estruturas de repetição.

Funções para listas

sum(): retorna a soma dos elementos de uma lista de números.

Sintaxe: **variavel** = **sum(nome_da_lista)**

Exemplo:

```
curros = [25, 37, 38, 118]
x = sum(curros)
print(x)
```

Saída

218

Observação: Já vimos e usamos essa função quando estudamos as estruturas de repetição.

Funções para listas

max(): retorna o máximo valor de uma lista.

Sintaxe: **variavel** = **max**(**nome_da_lista**)

Exemplo:

```
linguagens = ["Java", "Python", "C++"]
cursos = [25, 37, 38, 118]
x = max(cursos)
y = max(linguagens)
print(x, y)
```

Saída

118 Python

Observação: em lista de strings, o máximo é o último elemento na ordem alfabética e em lista numérica, o máximo é o maior valor.

Funções para listas

min(): retorna o máximo valor de uma lista.

Sintaxe: **variavel** = **min**(**nome_da_lista**)

Exemplo:

```
linguagens = ["Java", "Python", "C++"]
cursos = [25, 37, 38, 118]
x = min(cursos)
y = min(linguagens)
print(x, y)
```

Saída

25 C++

Observação: em lista de strings, o mínimo é o primeiro elemento na ordem alfabética e em lista numérica, o mínimo é o menor valor.

Funções para listas

sorted(): retorna uma nova lista ordenada sem alterar a original.

Sintaxe: **variavel** = **sorted**(**nome_da_lista**)

Exemplo:



```
linguagens = ["Java", "Python", "C++"]
cursos = [37, 25, 118, 38]
nova_lista = sorted(linguagens)
cursos_novo = sorted(cursos)
print(nova_lista)
print(cursos_novo)
```



```
["C++", "Java", "Python"]
[25, 37, 38, 118]
```

Observação: a nova lista terá os elementos em ordem crescente.

Funções para listas

enumerate(): retorna uma tupla que associa as posições e os valores correspondentes a partir de uma lista.

Sintaxe: **nome_tupla** = **enumerate**(**nome_da_lista**)

Exemplo:



```
linguagens = ["Java", "Python", "C++", "Kotlin"]  
nomes = enumerate(linguagens)  
print(list(nomes))
```



Saída

```
[(0, 'Java'), (1, 'Python'), (2, 'C++'), (3, 'Kotlin')]
```

Funções para listas

enumerate(): pode também ser usada para obter a posição e valor correspondente a ela durante uma iteração. A função gera tuplas.

Sintaxe: **posicao, valor = sorted(nome_da_lista)**

Exemplo:



```
linguagens = ["Java", "Python", "C++"]  
for pos,nome in enumerate(linguagens):  
    print(f"{pos} → {nome}")
```



```
0 → Java  
1 → Python  
2 → C++
```

Observação: a função enumerate retorna tuplas, então se for usado apenas um iterador no for para assumir os pares, ao ser impresso esse iterador, veremos na tela as tuplas.

Acessando os elementos de uma lista

Os elementos de uma lista podem ser acessados pelo índice, por fatiamento e por iteração em iteráveis.

Para acessar um determinado elemento **pelo índice**, usamos a seguinte sintaxe:



```
linguagens = ["Java", "Python", "C++"]  
print(linguagens[1])  
print(linguagens[2])
```



```
Python  
C++
```

Observação: o acesso se dá pelo nome da lista seguindo-se de um número inteiro representativo da posição desejada entre colchetes. Tentar acessar um índice maior que o limite da lista gera erro, pois a lista tem 3 elementos

Acessando os elementos de uma lista

Também é possível usar **índices negativos** quando se deseja acessar os elementos a partir do final da lista. No exemplo abaixo, -1 significa o último elemento (o primeiro de trás para frente) e -2 significa o penúltimo elemento (o segundo de trás para frente).



```
linguagens = ["Java", "Python", "C++"]  
print(linguagens[-1])  
print(linguagens[-2])
```



```
C++  
Python
```

Observação: o acesso se dá pelo nome da lista seguindo-se de um número inteiro representativo da posição desejada entre colchetes. Se o inteiro for maior do que o número de elementos que a lista tem, ocorrerá um erro.

Acessando os elementos de uma lista

Podemos usar ainda as funções `range` e `len` para acessar os índices dos elementos da lista da seguinte forma:

```
linguagens = ["Java", "Python", "C++"]  
for i in range(len(linguagens)):  
    elemento = linguagens[i]  
    print(elemento)
```

Saída

```
Java  
Python  
C++
```

Observação: o acesso se dá pelo nome da lista seguindo-se de um número inteiro representativo da posição desejada entre colchetes. Se o inteiro for maior do que o número de elementos que a lista tem, ocorrerá o erro “*list index out of range*”.

Acessando os elementos de uma lista

Já a estratégia chamada de **fatiamento** (*slicing*) permite acessar um intervalo de valores da lista informando os **índices do início (inclusivo)** e **fim do intervalo (exclusivo)**. **Omitir o primeiro** faz com que o **fatiamento ocorra do começo da lista até o índice indicado**. **Omitir o segundo** faz com que o **fatiamento ocorra do índice indicado até o final da lista**.



```
cursos = [16,25,29,35,37,38,49,118]
print(cursos[2:5])    #da posição 2 a 5
print(cursos[:4])     #até a posição 4
print(cursos[6:])     #após a posição 6
```



```
[29, 35, 37]
[16, 25, 29, 35]
[49, 118]
```

Acessando os elementos de uma lista

Por fim, o acesso por iteração em iterável nós já vimos quando estudamos as estruturas de repetição. Vamos relembrar:



```
linguagens = ["Java", "Python", "C++"]  
for curso in cursos:  
    print(cursos)
```

 Saída

```
Java  
Python  
C++
```



```
linguagens = ["Java", "Python", "C++"]  
for pos,nome in enumerate(linguagens):  
    print(f"{pos}: {nome}")
```

 Saída

```
0: Java  
1: Python  
2: C++
```

Exercícios

Exercício #1

Usando os métodos que vimos para listas, escreva um programa em Python para criar uma lista de compras que permita que:

- o usuário cadastre os itens que deseja comprar;
- coloque os itens em ordem alfabética;
- verifique e remova itens repetidos;
- após as compras estarem feitas, apague a lista.

```
mercado = []
resp = ""
i = 1

while resp != "0":
    resp = input(f"Digite o item {i}: ou 0 para sair: ")
    mercado.append(resp)
    i += 1
mercado.remove("0")
mercado.sort()

for item in mercado.copy():
    while mercado.count(item) > 1:
        mercado.remove(item)
print(mercado)

conclusao = input("Se compras feitas, digite ok: ")
if conclusao == "ok":
    mercado.clear()
    print("Lista encerrada.")
```

Exercício #2

Usando funções que vimos para as listas, escreva um programa em Python para ler e armazenar as notas de um aluno. O número de avaliações deve ser perguntado ao usuário. Em seguida, o programa deve:

- calcular e exibir a média das notas;
- identificar e exibir a menor nota;
- identificar e exibir a maior nota;
- calcular e exibir a média desconsiderando a menor nota.

```
notas = []
provas = int(input(f"Digite o número de provas: "))

for i in range(provas):
    notas.append(float(input(f"Digite a nota da prova {i}: ")))

media = sum(notas) / len(notas)
maior = max(notas)
menor = min(notas)

notas.remove(menor)
media_nova = sum(notas) / len(notas)

print(f"Média considerando todas a notas: {media:.2f}")
print(f"Maior nota: {maior:.2f}")
print(f"Menor nota: {menor:.2f}")
print(f"Média desconsiderando a menor nota: {media_nova:.2f}")
```

