

# Comandos Python: Listas

# Agenda

---

- Introdução
- Listas
- Exemplos e Exercícios

# Listas

# Listas (Breve Introdução)

- Uma lista em Python é uma estrutura que armazena vários dados, que **podem ser de um mesmo tipo ou não**.
- Uma lista é criada como a construção:  $[dado_1, dado_2, \dots, dado_n]$

```
lista1 = [10, 20, 30, 40]
lista2 = ["programação", "mc102", "python"]
lista3 = ["oi", 2.0, 5, [10, 20]]
```

# Listas

- Listas são construções de linguagens de programação que servem para armazenar vários dados de forma simplificada.

# Listas

- Suponha que desejamos guardar notas de alunos.
- Com o que sabemos, como armazenaríamos 3 notas?

```
nota1 = float(input("Entre com a nota 1: "))  
nota2 = float(input("Entre com a nota 2: "))  
nota3 = float(input("Entre com a nota 3: "))
```

# Listas

- Com o que sabemos, como armazenaríamos 130 notas?

```
nota1 = float(input("Entre com a nota 1: "))
nota2 = float(input("Entre com a nota 2: "))
nota3 = float(input("Entre com a nota 3: "))
nota4 = float(input("Entre com a nota 4: "))
nota5 = float(input("Entre com a nota 5: "))

...
nota130 = float(input("Entre com a nota 130: "))
```

- Criar 130 variáveis distintas **não** é uma solução elegante.

# Listas: Definição

- Coleção de valores referenciados por um **identificador único**.

```
identificador = [dado1, dado2, ..., dadon]
```

```
notas = [8.0, 5.5, 9.3, 7.6, 3.1]
```

- Características:
  - Acesso por meio de um índice inteiro.
  - Listas podem ser modificadas.
  - Pode-se incluir e remover itens de listas.



# Exemplos de Listas

- Lista de

inteiros:

```
x = [2, 45, 12, 9, -2]
```

- Listas podem conter dados de tipos diferentes:

```
x = [2, "qwerty", 45.99087, 0, "a"]
```

- Listas podem conter outras listas:

```
x = [2, [4, 5], [9]]
```

- Ou podem não conter nada. Neste caso `[]` indica a lista vazia.

# Listas: Como Usar

- Pode-se acessar uma determinada posição da lista utilizando-se um índice de valor inteiro.
- A sintaxe para acesso de uma determinada posição é:
  - `identificador[posição]`
- Sendo  $n$  o tamanho da lista, os índices válidos para ela vão de 0 até  $n - 1$ .
  - A primeira posição da lista tem índice 0.
  - A última posição da lista tem índice  $n - 1$ .

# Listas: Como Usar

Lista `notas` : tamanho  $n = 5$ , ou seja, os índices válidos são de 0 até 4 ( $5 - 1$ ).

```
notas = [8.0, 5.5, 9.3, 7.6, 3.1]
print(notas[0])
print(notas[1])
print(notas[2])
print(notas[3])
print(notas[4])
```

```
8.0
5.5
9.3
7.6
3.1
```

# Listas: Como Usar

Lista `notas` : tamanho  $n = 5$ , ou seja, os índices válidos são de 0 até 4 ( $5 - 1$ ).

```
notas = [8.0, 5.5, 9.3, 7.6, 3.1]
```

```
print(notas[0])
```

```
print(notas[1])
```

```
print(notas[2])
```

```
print(notas[3])
```

```
print(notas[4])
```

A primeira posição da lista  
tem índice 0

A última posição da lista  
tem índice  $n - 1$

8.0

5.5

9.3

7.6

3.1

# Listas: Como Usar

- Um elemento de uma lista em uma posição específica tem o mesmo comportamento que uma variável simples.

```
notas = [8.0, 5.5, 9.3, 7.6, 3.1]
```

```
print(notas[0]+2)
```

```
10.0
```

```
notas[3] = 0.5
```

```
print(notas)
```

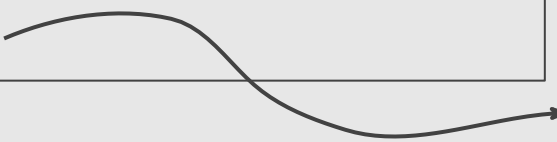
```
[8.0, 5.5, 9.3, 0.5, 3.1]
```

# Listas: Como Usar

- Você deve usar valores inteiros como índice para acessar uma posição da lista.
- O valor pode ser inclusive uma variável inteira.

```
notas = [8.0, 5.5, 9.3, 0.5, 3.1]
```

```
for i in range(5):  
    print(notas[i])
```



```
8.0  
5.5  
9.3  
0.5  
3.1
```

# Listas: Como Usar

- Quais valores serão armazenados em cada posição da lista após a execução deste código abaixo?

```
lista = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

```
for i in  
    range(10):  
    lista[i] = 5*i
```


```
print(lista)
```

```
[0, 5, 10, 15, 20, 25, 30, 35, 40, 45]
```

# Listas: Índices

- Índices negativos se referem à lista da direita para a esquerda:

```
notas = [8.0, 5.5, 9.3, 0.5, 3.1]
print(notas[-1])
print(notas[-2])
print(notas[-3])
print(notas[-4])
print(notas[-5])
print(notas[-6])
```



```
3.1
0.5
9.3
5.5
8.0
IndexError: list
index out of range
```

- Ocorre um erro se tentarmos acessar uma posição da lista que não existe.



# Listas: Índices

- Listas em Python suportam uma operação conhecida como **slicing**, que consiste em obter uma sub-lista contendo os elementos de uma posição inicial até uma posição final de uma lista.

- O **slicing** em Python é obtido como

```
identificador[ind1:ind2]
```

e o resultado é uma sub-lista com os elementos de `ind1` até `ind2-1`.

# Listas: Índices

- O **slicing** em Python é obtido como

```
identificador[ind1:ind2]
```

e o resultado é uma sub-lista com os elementos de `ind1` até `ind2-1`.

```
notas = [8.0, 5.5, 9.3, 0.5, 3.1]  
print(notas[1:4])
```

```
[5.5, 9.3, 0.5]
```

# Listas: Função `len`

- A função `len(lista)` retorna o número de itens na lista.

```
notas = [8.0, 5.5, 9.3, 0.5, 3.1]
len(notas)
```

```
5
```

- É muito comum usar a função `len` junto com o laço `for` para percorrer todas as posições de uma lista:

```
notas = [8.0, 5.5, 9.3, 0.5, 3.1]
for i in range(len(notas)):
    print(notas[i])
```

# Listas: `for`

- Lembre-se que o `for` na verdade faz a variável de controle assumir todos os valores de uma lista. Assim:

```
notas = [8.0, 5.5, 9.3, 0.5, 3.1]
for i in range(len(notas)):
    print(notas[i])
```

- E também pode ser implementado como:

```
notas = [8.0, 5.5, 9.3, 0.5, 3.1]
for i in notas:
    print(i)
```

# Listas: método `append`

- Uma operação importante é acrescentar um item no final de uma lista. Isto é feito pela função **append**.

```
lista.append(item)
```



A lista que será modificada aparece antes, seguida de um ponto, seguida do `append` com o item a ser incluído como argumento. Formalmente, este tipo de função é chamada de método.

# Listas: método `append`

- Uma operação importante é acrescentar um item no final de uma lista. Isto é feito pela função **`append`**.

```
lista.append(item)
```

```
notas = [8.0, 5.5, 9.3, 0.5, 3.1]  
notas.append(9.5)  
print(notas)
```

```
[8.0, 5.5, 9.3, 0.5, 3.1, 9.5]
```

# Listas: método `append`

- A combinação de uma lista vazia que vai sofrendo “appends” permite ler dados e preencher uma lista com estes dados:

```
notas = []
n = int(input("Entre com o número de notas: "))
for i in range(n):
    dado = float(input("Entre com a nota " + str(i) + ": "))
    notas.append(dado)
print(notas)
```

# Listas: operador +

- A operação de soma em listas gera uma nova lista que é o resultado de “grudar” `lista2` ao final da `lista1`. Isto é conhecido como **concatenação de listas**.

```
lista1 + lista2
```

```
lista1 = [1, 2, 4]
lista2 = [27, 28, 29, 30, 33]
x = lista1 + lista2
print(x)
```



# Listas: operador +

- A operação de soma em listas gera uma nova lista que é o resultado de “grudar” `lista2` ao final da `lista1`. Isto é conhecido como **concatenação de listas**.

```
lista1 + lista2
```

```
lista1 = [1, 2, 4]
lista2 = [27, 28, 29, 30, 33]
x = lista1 + lista2
print(x)
```

```
[1, 2, 4, 27, 28, 29, 30, 33]
```

# Listas: operador \*

- O operador “\*” faz repetições da concatenação:

```
x = [1, 2, 3]  
y = 4*x  
print(y)
```

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

- O resultado da operação do exemplo é o mesmo que somar (concatenar) 4 vezes a lista `x`.

# Outros Métodos em Listas

- `lista.insert(índice, dado)` insere na lista o dado antes da posição índice.

```
x = [40, 30, 10, 20]
x.insert(1, 99)
print(x)
```

# Outros Métodos em Listas

- `lista.insert(índice, dado)` insere na lista o dado antes da posição índice.

```
x = [40, 30, 10, 20]
x.insert(1, 99)
print(x)
```

```
[40, 99, 30, 10, 20]
```

# Outros Métodos em Listas

- `del lista[posição]` remove da lista o item da posição especificada.

```
x = [40, 99, 30, 10, 20]
del x[2]
print(x)
```

```
[40, 99, 10, 20]
```

# Outros Métodos em Listas

- Também podemos remover um item da lista pelo valor utilizando o método `remove`.

```
x = [40, 99, 10, 20]
x.remove(10)
print(x)
```

```
[40, 99, 20]
```

# Outros Métodos em Listas

- Podemos contar o número de elementos na lista com um certo valor usando o método `count`.

```
x = [40, 99, 10, 20, 13, 20, 14]
N = x.count(20)
print(N)
```

2

# Informações Extras: Inicialização de uma Lista

- Em algumas situações é necessário declarar e já atribuir um conjunto de valores constantes para uma lista.
- Dentro da lista incluímos uma construção com um laço que gerará valores iniciais para a lista.

```
x = [0 for i in range(5)]
```

```
print (x)
```

```
[0, 0, 0, 0, 0]
```

```
x = [2*i for i in range(5)]
```

```
print (x)
```

```
[0, 2, 4, 6, 8]
```



# Exemplos & Exercícios

# Exercício

- Faça um programa que leia um número  $n$  e imprima  $n$  linhas na tela com o seguinte formato (exemplo se  $n = 5$ ):

Entrada	Saída
5	1 1 2 1 2 3 1 2 3 4 1 2 3 4 5

# Exercício

- Faça um programa que leia  $n$  notas, mostre as notas e a média.

Entrada	Saída
5	[ 8 . 0 , 5 . 5 , 9 . 3 , 0 . 5 , 3 . 1 ]
8 . 0	5 . 3
5 . 5	
9 . 3	
0 . 5	
3 . 1	

# Exercício

- Faça um programa que:
  - Lê duas listas com 5 inteiros cada.
  - Checa quais elementos da segunda lista são iguais a algum elemento da primeira lista.
  - Se não houver elementos em comum, o programa deve informar isso.

Entrada	Saída
[1, 2, 3, 4, 5] [0, 7, 6, 10, 3]	3

Entrada	Saída
[1, 2, 3, 4, 5] [0, 7, 6, 10, 8]	Não tem.

# Referências & Exercícios

- <https://wiki.python.org.br/ExerciciosListas>: 24 exercícios  
=)

# Créditos

---

Os *slides* deste curso foram baseados nos slides produzidos e cedidos gentilmente pela Professora Sandra Ávila, do Instituto de Computação da Unicamp.