



Introdução à linguagem Python

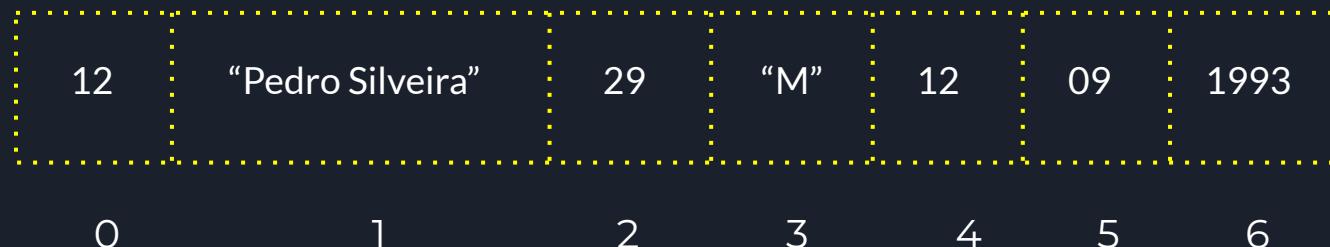
Aula 4 - Tuplas, Listas e Dicionários | Parte 1

Por: Thiago Ribeiro da Silva

Estudante de Ciência da Computação (UFAL)

Estruturas compostas

- As linguagens de programação possibilitam a criação de variáveis, mas em muitas situações, necessitamos de estruturas que possibilitem guardar um agrupamento de informações, e não apenas uma.



Estruturas compostas

- Para isso, as linguagens possuem estruturas capazes de guardar múltiplas informações, gerando variáveis compostas. Em Python, diversas estruturas exercem essa função, mas falaremos aqui sobre tuplas, listas e dicionários.



Sobre Tuplas

- A primeira estrutura que veremos se chama tupla. Ela consiste numa estrutura de dados inalterável, ou seja, ao criá-la, é impossível editar a informação dentro dela.
- Uma tupla é declarada colocando informações separadas por vírgula dentro de parênteses.

```
1 # Criando uma tupla
2
3 # modo tradicional
4 tupla1 = (12, "Pedro", 29, "123.456.789-00")
5 tupla2 = () # tupla vazia
6 # outra maneira de declaração
7 tupla3 = 1, 2, 2, 4, 2
```

Sobre Tuplas

- Tuplas são ideais para situações em que os dados são sensíveis e não devem ser alterados;
- Lembra de strings? Elas são agrupamentos de caracteres, enquanto estruturas compostas são agrupamentos de dados (no geral), o que possibilita o uso de muitas operações em comum, como fatiamento, indexação e contagem.

```
#Operações com tuplas

print(tupla1[1])
print(tupla1[0:2])
print(tupla1.index("Pedro Silveira"))
print(tupla3.count(2))

⇒ Pedro Silveira
      (12, 'Pedro Silveira')
      1
      3
```

Sobre Tuplas - Exemplo



```
"""
Tuplas, por serem agrupamentos de dados, podem ser percorridas com o for loop
"""


```

```
tupla = (12, "Pedro Silveira", 29, "123.456.789-00")

for elemento in tupla:
    print(elemento)
```

```
12
Pedro Silveira
29
123.456.789-00
```

Sobre Listas

- Listas são estruturas semelhantes às tuplas, contudo, estas podem ter seus dados alterados, o que possibilita o uso de muitas outras operações.
- A declaração de listas ocorre de maneira semelhante às tuplas, mas ao contrário dos parênteses, usa-se colchetes.

```
● ● ●  
1 #Criando uma lista  
2  
3 lista1 = [12, "Pedro", 29, "123.456.789-00"]  
4 #Listas inicializadas de forma vazia  
5 lista2 = list()  
6 lista3 = []
```

Sobre Listas

- Para alterar um valor em uma dada posição de uma lista, basta atribuir um novo valor ao índice específico que se quer alterar:
- **lista[posição] = novo_valor**
- Esse novo valor pode ser de qualquer tipo, não sendo necessário ser do mesmo tipo do valor anterior.



```
# Alterando valor
lista1 = [12, "Pedro", 29, "123.456.789-00"]

print(f"Antes: {lista1}")
lista1[0] = 15
print(f"Depois: {lista1}")
```

```
Antes: [12, 'Pedro', 29, '123.456.789-00']
Depois: [15, 'Pedro', 29, '123.456.789-00']
```

Sobre Listas

- Listas são passadas pelo seu endereço na memória, sendo assim, para gerar um cópia de uma lista, deve-se usar o comando `.copy()`.
Apenas a atribuição a uma nova variável não vai copiá-la, apenas gerar uma nova variável com o mesmo endereço.

```
[14] #Tentando criar uma cópia apenas por atribuição
lista1 = [12, "Pedro", 29, "123.456.789-00"]
lista_copia = lista1
```

```
lista1[0] = 14
print(lista1)
print(lista_copia)
```

```
[14, 'Pedro', 29, '123.456.789-00']
[14, 'Pedro', 29, '123.456.789-00']
```

```
[15] #Criando cópia com o comando .copy()
lista1 = [12, "Pedro", 29, "123.456.789-00"]
lista_copia = lista1.copy()
```

```
lista1[0] = 14
print(lista1)
print(lista_copia)
```

```
[14, 'Pedro', 29, '123.456.789-00']
[12, 'Pedro', 29, '123.456.789-00']
```



Sobre Listas

- Algumas das principais funções internas para manipulação de listas, além das apresentadas com tuplas:

FUNÇÃO	DESCRIÇÃO
len(lista)	retorna a quantidade de elementos na lista
max(lista), min(lista)	retorna o maior e o menor elemento da lista
lista.append(elemento)	Adiciona um novo elemento ao final da lista
lista.clear()	limpa toda a lista



Sobre Listas

- Algumas das principais funções internas para manipulação de listas, além das apresentadas com tuplas:

FUNÇÃO	DESCRIÇÃO
lista.insert(pos, elemento)	Adiciona um elemento numa posição específica
lista.sort(key, reverse)	Ordena a lista por uma função passada. É possível ordenar de forma decrescente como parâmetro reverse = True
lista.pop(posicao)	Remove um elemento em uma dada posição
lista.remove(elemento)	Remove a primeira ocorrência de um certo elemento



Sobre Listas - Exemplos



```
# Operações em listas
```

```
lista = [1, 3, 5, 7, 9]
```

```
lista.append(11)
```

```
print(lista)
```

```
lista.insert(0, -1)
```

```
print(lista)
```

```
lista.pop(len(lista)-2)
```

```
print(lista)
```

```
[1, 3, 5, 7, 9, 11]
```

```
[-1, 1, 3, 5, 7, 9, 11]
```

```
[-1, 1, 3, 5, 7, 11]
```

Sobre Listas - Exemplos

```
#Receba 5 inteiros do usuário e então determine o maior e menor elemento

nums = []

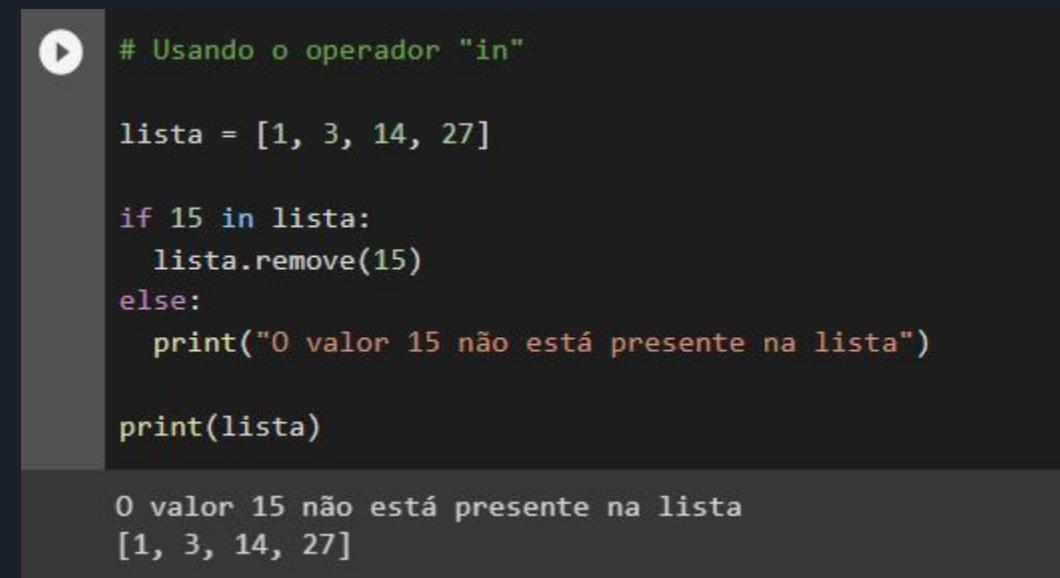
for i in range(5):
    num = int(input("Digite um número: "))
    nums.append(num)

print(f"O maior elemento é: {max(nums)}")
print(f"O menor elemento é: {min(nums)}")
```

```
↳ Digite um número: 14
Digite um número: 2
Digite um número: -100
Digite um número: 45
Digite um número: 20
O maior elemento é: 45
O menor elemento é: -100
```

Sobre Listas - Exemplos

- O operador **in** verifica se um dado elemento existe dentro de uma estrutura composta, retornando **True** ou **False**.



The screenshot shows a code editor window with a dark theme. On the left, there is a vertical toolbar with three colored bars: blue at the top, green in the middle, and red at the bottom. The main area contains Python code and its execution results.

```
# Usando o operador "in"

lista = [1, 3, 14, 27]

if 15 in lista:
    lista.remove(15)
else:
    print("O valor 15 não está presente na lista")

print(lista)
```

0 valor 15 não está presente na lista
[1, 3, 14, 27]

Sobre Listas - Exemplos

- É possível, ainda, fazer composições entre estruturas, ou seja, pode-se criar listas de listas, ou listas de tuplas



#Composição de estruturas

```
dados = [("Pedro", 14), ("Jonas", 22), ("Maria", 18)]
print(dados[0]) #conferindo o dado na posição 0
print(dados[1][1]) #conferindo dado na posição (1,1)
dados.append(("Joao", 18)) #adicionando tupla
print(dados)
```

('Pedro', 14)
22
[('Pedro', 14), ('Jonas', 22), ('Maria', 18), ('Joao', 18)]

Obrigado!

