

CC1612

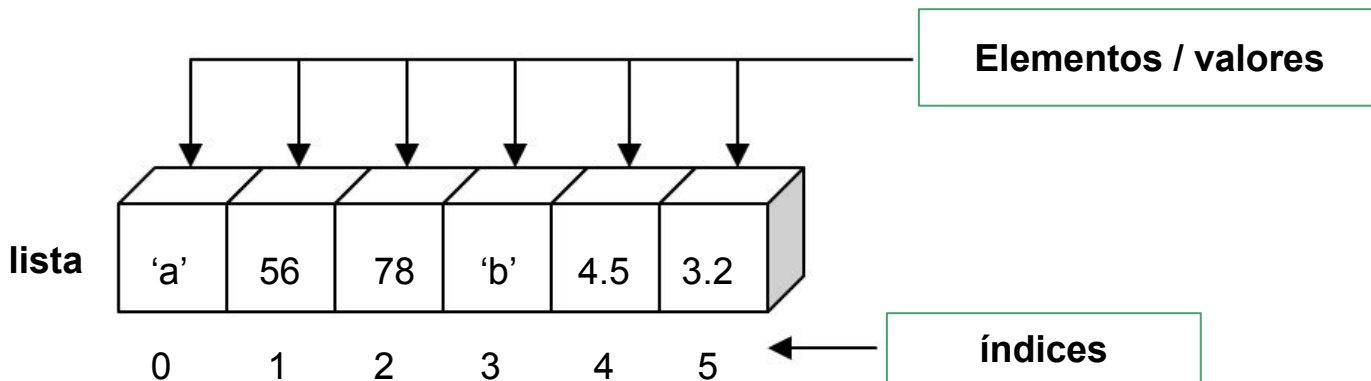
Fundamentos de Algoritmos

Prof. Danilo H. Perico

Listas

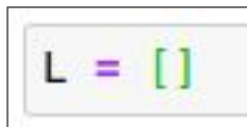
Lista

- **Uma lista é uma variável que armazena um conjunto de valores**
- Lista é um tipo de **variável** que permite o armazenamento de valores com tipos homogêneos ou heterogêneos (do mesmo tipo ou de tipos diferentes)
- Os valores armazenados em uma lista são acessados por um **índice**



Lista

- Para indicar que uma variável é uma lista, o símbolo de colchetes `[]` é utilizado para delimitar o conjunto
- Sintaxe - criando uma lista chamada `L`:



```
L = []
```

A code snippet showing the creation of an empty list. The variable 'L' is followed by an equals sign and a pair of empty square brackets, indicating an empty list.

- `L` é uma lista vazia

Lista

- Exemplo:
 - Criando uma lista chamada **z** com 3 números inteiros

```
z = [5, 7, 1]
print(z)
```

[5, 7, 1]

- Dizemos que **z** tem tamanho **3**
- Podemos utilizar o *Python Tutor* para verificar melhor a lista:
 - <http://pythontutor.com/visualize.html#mode=edit>

Lista - Acesso aos elementos

- Exemplo:

```
z = [5, 7, 1]
print(z)
```

[5, 7, 1]

- Para acessarmos o primeiro número da lista z, utilizamos a notação: **z[0]**
- Ou seja, da lista z queremos pegar o valor armazenado no índice 0.

```
z = [5, 7, 1]
print(z[0])
print(z[1])
print(z[2])
```

5
7
1

Lista

- Utilizando o nome de uma lista com o índice desejado, podemos também modificar o conteúdo armazenado.
- Exemplo: Alterando o valor do primeiro elemento da lista **z**

```
z = [5, 7, 1]
```

```
z[0] = 32
```

```
print(z)
```

```
[32, 7, 1]
```

Lista - Fatiamento (*Slicing*)

- No Python, podemos também fatiar as listas
- Ou seja, pegar somente partes de uma lista
- Exemplo:

```
p = [1,2,3,4,5,6]  
print(p[0:5])
```

```
[1, 2, 3, 4, 5]
```

```
print(p[:4])
```

```
[1, 2, 3, 4]
```

```
print(p[1:3])
```

```
[2, 3]
```

```
print(p[-1])
```

```
6
```


Lista - Adicionando elementos no fim da lista

- Podemos ainda adicionar novos elementos no fim da lista
- Para isto, utilizamos o método *append(item)*
- Exemplo:

```
z = [32, 7, 1]  
print(z)
```

```
[32, 7, 1]
```

```
z.append("oi")  
print(z)
```

```
[32, 7, 1, 'oi']
```

Lista - Adicionando elemento em qualquer lugar

- Podemos ainda adicionar novos elementos em qualquer lugar da lista
- Para isto, utilizamos o método *insert(índice, item)*
- Exemplo:

```
z = [32, 7, 1]  
print(z)
```

```
[32, 7, 1]
```

```
z.insert(1, "oi")  
print(z)
```

```
[32, 'oi', 7, 1]
```

Lista - Removendo da lista pelo índice

- Podemos remover um elemento da lista
- Para isto, utilizamos o método *pop(índice)*
- Exemplo:

```
z = ["a", "b", "c", "d", "e"]  
print(z)
```

```
['a', 'b', 'c', 'd', 'e']
```

```
z.pop(1)  
print(z)
```

```
['a', 'c', 'd', 'e']
```

Lista - Removendo da lista pelo elemento

- Podemos remover um elemento da lista
- Para isto, utilizamos o método *remove(item)*
- Exemplo:

```
z = ["a", "b", "c", "d", "e"]  
print(z)
```

```
['a', 'b', 'c', 'd', 'e']
```

```
z.remove("d")  
print(z)
```

```
['a', 'b', 'c', 'e']
```

```
z = [1,2,3,1,4,5,1]  
z.remove(1)  
print(z)
```

```
[2, 3, 1, 4, 5, 1]
```

Lista - Cópia

- A cópia de uma lista para uma nova variável requer alguma atenção!
- Por exemplo, se quisermos copiar a lista `z` para uma nova variável chamada `z1`, o mais natural seria o seguinte:

`z1 = z`

- Porém, quando fazemos isso no Python, criamos duas variáveis que referenciam a mesma lista!
- É como se déssemos dois nomes para a mesma lista

Lista - Cópia

- Exemplo:
- Quando alteramos o elemento na lista z, a alteração ocorre também na lista z1

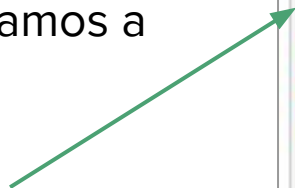
```
z = [4,5,3,6]
z1 = z
print("antes da alteração na lista z")
print(z)
print(z1)
z[1] = 98
print("depois da alteração na lista z")
print(z)
print(z1)
```

```
antes da alteração na lista z
[4, 5, 3, 6]
[4, 5, 3, 6]
depois da alteração na lista z
[4, 98, 3, 6]
[4, 98, 3, 6]
```

Lista - Cópia

- Para criarmos uma cópia independente, utilizamos a sintaxe:

z1 = z[:]



```
z = [4,5,3,6]
z1 = z[:]
print("antes da alteração na lista z")
print(z)
print(z1)
z[1] = 98
print("depois da alteração na lista z")
print(z)
print(z1)
```

antes da alteração na lista z

[4, 5, 3, 6]

[4, 5, 3, 6]

depois da alteração na lista z

[4, 98, 3, 6]

[4, 5, 3, 6]

Lista - Tamanho da lista

- Como temos os métodos para incluir e remover dados das listas, nem sempre sabemos qual é o tamanho exato que a lista tem
- Para descobrirmos o tamanho da lista, utilizamos o método *len(lista)*
- Exemplo:

```
a = [3, 4, 5]  
print(len(a))
```

```
a.append(9)  
a.append(11)  
print(len(a))
```

3

5

Lista - Pesquisando na lista

- Podemos pesquisar se um elemento está na lista
- Para isso, verificamos do primeiro ao último comparando com o que queremos encontrar.
- Para percorrer listas, utilizamos uma estrutura de repetição: *while* ou *for*
- A estrutura *for* é otimizada para trabalhar com listas

Lista - Pesquisando na lista

- Exemplo: Procurar o elemento “c” na lista z

```
z = ["a", "b", "c", "d", "e"]
for elemento in z:
    if elemento == "c":
        print("Elemento encontrado!")
        break
    else:
        print("Elemento não encontrado!")
```

Elemento encontrado!

Lista - Pesquisando na lista

- Porém, se a ideia é somente falar se o elemento está ou não na lista, podemos utilizar uma estrutura mais simples:

```
z = ["a", "b", "c", "d", "e"]  
if "c" in z:  
    print("Encontrado!")  
else:  
    print("Não encontrado!")
```

Encontrado!

Lista - Pesquisando na lista

- Contudo, nem sempre encontrar o elemento é suficiente.
- Muitas vezes, precisamos saber qual é a sua posição na lista.
- Exemplo:

```
z = ["a", "b", "c", "d", "e"]
for indice in range(len(z)):
    if z[indice] == "c":
        print("Elemento encontrado no índice %d" % indice)
        break
else:
    print("Elemento não encontrado!")
```

Elemento encontrado no índice 2