UniCEUB - Bacharelado em Ciência da Computação
Professor: Antônio **Barbosa** Júnior
Python - lista - list

Lista é um conjunto sequencial de valores, onde cada valor é identificado através de um índice.
O primeiro valor armazenado na lista tem índice 0.

Uma lista pode ter valores de qualquer tipo, incluindo outras listas.
Exemplo:

```
lista1 = [3 , 'abacate' , 9.7 , [5 , 6 , 3] , "Python" , (3 , 'j')]
print(lista1[2])
```
9.7

```
print(lista1[3])
```
[5,6,3]

```
print(lista1[3][1])
```
6

1. Uma lista em Python é declarada da seguinte forma.

1.1. Cria lista vazia:

```
lista_um = []
lista_um = list()
```

1.2. Cria lista não vazia:

```
lista_dois = [10, 20, 30, 40]
              0   1   2   3     - Índice ou posição da lista, positivos.
             -4  -3  -2  -1     - Índice ou posição da lista, negativos.
```

1.3. Cria lista com a função range ()
Sintaxe: range([i,] n [, p])
i – valor inicial. Valor default é 0
n – valor final. Obrigatório
p – passo ou incremento. Valor default é 1
Os parâmetros i e p são opcionais.
A função range () define um intervalo de valores inteiros.
Associada a list (), cria uma lista com os valores do intervalo.
A função range () pode ter de 1 a 3 parâmetros:

- range (n) - gera uma sequência de 0 a n-1 de 1 em 1;
- range (i , n) - gera uma sequência de i a n-1 de 1 em 1;
- range (i , n, p) - gera uma sequência de i a n-1 com intervalo p entre os números.

Exemplos:

```
L1 = list(range(5))
print(L1)
```
[0, 1, 2, 3, 4]

```
L2 =list(range(3,8))
print(L2)
```
[3, 4, 5, 6, 7]

```
L3 = list(range(2, 11, 3))
print(L3)
[2, 5, 8]

L1 = range(5)      # Precisa do list
print(L1)          # Não criou a lista
```

2. Métodos de lista:

2.1. Insere

```
x = 123
```

Sintaxe: lista.append(*x*)

Adiciona um elemento ao final da lista.

Ex.: antes da utilização do método:

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

depois:

```
lista.append(9)
print(lista)
[1, 2, 3, 4, 5, 9]
```

Sintaxe; lista.insert(*i*, *x*)

Insere um item (x) numa determinada posição ou índice (i).

O primeiro argumento é o índice do elemento (i) antes do qual será feita a inserção.

Ex.: antes da utilização do método:

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

depois:

```
lista.insert(2, 6)
print(lista)
[1, 2, 6 ,3 ,4, 5]
```

Obs.: se a posição não existir, o valor será inserido no final da lista.

```
lista.insert(12,6)
print(lista)
```

2.2. Consulta

Sintaxe: lista.index(*x* [, *start* [, *end*]])

Retorna o índice baseado em zero na lista do primeiro item cujo valor é igual a x.

Gera um ValueError se não houver tal item (x).

Os argumentos opcionais start (início) e end (fim) são interpretados como na *slice notation* e são usados para limitar a pesquisa a uma subsequência específica da lista.

O índice retornado é calculado em relação ao início da sequência completa em vez do argumento inicial.

Ex.: antes da utilização do método:

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

depois:

```
print(lista.index(3))
2
```

Sintaxe: lista.count(*x*)

Retorna o número de vezes que x aparece na lista.
Ex.: antes da utilização do método:
lista = [1, 2, 2, 2, 3, 4, 5]
depois:
print(lista.count(2))
3
print(lista.count(9))
0

Sintaxe: x in lista:
in verifica se um valor pertence à lista. Retorna True ou False

```
lista = [1, 2 , 3, 4]
if 3 in lista:
   print('True')
```

2.3. Atualiza

lista1 = [3 , 'abacate' , 9.7 , [5 , 6 , 3] , "Python" , (3 , 'j')]
Sintaxe: lista [i] = 'valor'
Notação de vetor.
Exemplos:
Lista1 [3]= 'morango'
print(lista1)
[3 , 'abacate' , 9.7 , 'morango', "Python" , (3 , 'j')]
lista [7]= 'banana'
A tentativa de acesso a um índice inexistente resultará em erro.

```
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    L[7]='banana'
IndexError: list assignment index out of range
```

2.4. Remove

Sintaxe: lista.remove(*x*)
Remove o primeiro item encontrado na lista cujo valor é igual a *x*.
Se não existir valor igual, uma exceção ValueError é levantada.
Ex.: antes da utilização do método:
lista = [1, 2, 3, 4, 5]
depois:
lista.remove(4)
print(lista)
[1, 2, 3, 5]

Sintaxe: lista.pop([i])
Remove o item na índice (posição) dada.
Se nenhum índice for especificado, a pop() remove e devolve o último item na lista.
Ex.: antes da utilização do método:
lista = [1, 2, 3, 4, 5]
depois:

```
print(lista.pop(1))
2
print(lista)
lista = [1, 3, 4, 5]
```

Sintaxe: del lista[i]
del remove um elemento da lista, dado seu índice.
lista = [1, 2, 3, 4, 5]
del lista[1]
lisat: [1, 3, 4, 5]

Sintaxe: lista.clear()

Remove todos os itens da lista.

Ex.: antes da utilização do método:

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

depois:

```
lista.clear()
print(lista)
[]
```

2.5 Ordena

Sintaxe: lista.sort(*key=None*, *reverse=False*)

Ordena os itens da lista no lugar (os argumentos podem ser usados para uma organização personalizada, veja sorted() para a sua explicação).

Altera a lista original e retorna None.

Ex.: antes da utilização do método:

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

depois:

```
lista.sort()
print(lista)
[1, 2, 3, 4, 5]
```

Sintaxe: sorted(lista)

Retorna os elementos da lista em uma ordem crescente e não altera a lista original:

Ex.: antes da utilização do método:

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

depois:

```
nova_lista = sorted(lista)
print(nova_lista)
[1, 2, 3, 4, 5]
```

Sintaxe: lista.reverse()

Inverta os elementos da lista.

Altera a lista original e retorna None.

Ex.: antes da utilização do método:

```
lista = [1, 2, 7, 3, 4, 5]
```

depois:

```
lista.reverse()
print(lista)
[5, 4, 3, 7, 2, 1]
```

Sintaxe: reversed(lista) # builtins

```
lista = [1, 2, 3, 4, 5]
print(lista)
# depois:
print( list(reversed(lista)))
lista = [1, 2, 3, 4, 5]
print(lista)
l = list(reversed(lista))
print(l)
print(lista)
```

2.6. Numérico

Sintaxe: len(lista)

retorna o número de elementos da lista:

Ex.: antes da utilização do método:

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

depois:

```
l = print(len(lista))
print(l)
5
```

max(lista)

retorna o maior elemento da lista:

Ex.: antes da utilização do método:

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

depois:

```
m = max(lista)
print(m)
5
```

min(lista)

retorna o menor elemento da lista:

Ex.: antes da utilização do método:

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

depois:

```
m = min(lista)
print(m)
1
```

sum(lista)

retorna a soma dos elementos da lista:

Ex.:

antes da utilização do método:

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

depois:

```
soma = sum(lista)
print(soma)
15
```

2.7. Miscelânea

Sintaxe: lista.copy()

Retorna uma cópia superficial da lista. Equivalente a = a[:].

Ex.: antes da utilização do método:

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

depois:

```
copia = lista.copy()
print(copia)
[1,2,3,4,5]
```

Sintaxe: tuple(lista)

retorna os elementos da lista em formato de tupla:

Ex.: antes da utilização do método:

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

depois:

```
t1 = tuple(lista)
print(t1)
(1,2,3,4,5)
```

Sintaxe: map(função. lista)

É uma função builtin de Python, isto é, uma função que é implementada diretamente no interpretador Python, podendo ser utilizada sem a importação de um módulo específico.
Aplica uma função a cada elemento de uma lista, retornando uma nova lista contendo os elementos resultantes da aplicação da função. Considere o exemplo abaixo:
Exemplo:

```
import math
lista1 = [1, 4, 9, 16, 25]
lista2 = map(math.sqrt, lista1)
print lista2
[1.0, 2.0, 3.0, 4.0, 5.0]
```

2.8 Operações com listas

Concatenação (+)
Ex. 1:

```
a = [0,1,2]
b = [3,4,5]
c = a + b
print(c)
[0, 1, 2, 3, 4, 5]
```

Ex. 2:

```
a = [0,1,2]
b = [2, 3,4,5]
c = a + b
print(c)
[0, 1, 2, 3, 4, 5]
```

Sintaxe: lista.extend(*iterable*)

Estende a lista, adicionando todos os elementos do *iterable*.

Ex.: antes da utilização do método:

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

depois:

```
                    lista.extend([6,7,8])
                    print(lista)
                    [1,2,3,4,5,6,7,8]
        # Ex.:  antes da utilização do método:
                lista = [1,2,3,4,5]
                lista2 = [9, 8, 7, 6]
        # depois:
                lista.extend(lista2)
                print(lista)
```

Repetição (*)

```
L = [1,2]
R = L * 4
print(R)
[1, 2, 1, 2, 1, 2, 1, 2]
print(L)
[1, 2]
```