Aula 07

Trabalhando com Lista no Python

Tópicos

- Introdução às listas em Python
- Acesso a elementos de uma lista
- Operações básicas com listas (adição, remoção, alteração)
- Métodos de listas em Python (append(), extend(), insert(), remove(), etc.)
- Iteração sobre elementos de uma lista (loops for e while)
- Indexação e fatiamento de listas
- Listas aninhadas (listas dentro de listas)
- Compreensão de lista (list comprehension)
- Ordenação de listas
- Filtragem de listas com filter()

Introdução às listas em Python

Introdução às lista em Python

- Você pode criar uma lista vazia.
- Você também pode criar uma lista com valores inteiros.
- Você também pode criar uma lista com valores string.

```
[28] # Criando uma lista vazia
     lista_vazia = []

     # Criando uma lista com elementos
     numeros = [1, 2, 3, 4, 5]
     nomes = ["João", "Maria", "Pedro"]
```

Acesso a elementos de uma lista

Acesso a elementos de uma lista

- Você pode acessar qualquer elementos de uma lista, basta apenas informar o índex do elemento.

```
[33] numeros = [1, 2, 3, 4, 5]
     print(numeros[0])  # Acessando o primeiro elemento da lista (índice 0)
     print(numeros[-1])  # Acessando o último elemento da lista

     1
     5
```

Operações básicas com listas

Operações básicas com listas

- Você pode fazer operações como adicionar um elemento numa lista, remover um elemento de uma lista ou inverter a ordem de uma lista.

```
[32] numeros = [1, 2, 3, 4, 5]
     print(numeros)
     numeros.append(6)  # Adicionando um elemento ao final da lista
     print(numeros)

     numeros.remove(3)  # Removendo um elemento da lista
     print(numeros)

     numeros.reverse()  # Invertendo a ordem dos elementos na lista
     print(numeros)
```

```
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5, 6]
[1, 2, 4, 5, 6]
[6, 5, 4, 2, 1]
```

Métodos de listas em Python

Métodos de listas

- Quando você utiliza o objeto List no Python, alguns métodos já estão incorporados, como:
  - append(),
  - extend(),
  - insert(),
  - remove(),...

```
[66] numeros = [9, 20, 13, 4, 45]
     numeros.append(50)
     numeros

     [9, 20, 13, 4, 45, 50]
```

```
[67] # Lista original
     numeros1 = [1, 2, 3]
     numeros2 = [4, 5, 6]

     numeros1.extend(numeros2)

     # Imprimindo lista1 após a extensão
     print(numeros1)

     [1, 2, 3, 4, 5, 6]
```

```
[65] numeros = [9, 20, 13, 4, 45]
     numeros.insert(1,25)
     numeros

     [9, 25, 20, 13, 4, 45]
```

```
[70] numeros = [9, 20, 13, 4, 45]
     numeros.remove(13)
     numeros

     [9, 20, 4, 45]
```

Iteração sobre elementos de uma lista

Interação sobre elementos de uma lista

- Você pode fazer uma interação com os elementos de uma lista, aplicando a estrutura de repetição “for”.

```
[34] numeros = [1, 2, 3, 4, 5]
     for numero in numeros:
         print(numero)
```

```
1
2
3
4
5
```

Indexação e fatiamento de listas

Interação sobre elementos de uma lista

- Você pode utilizar o recurso de fatiamento para selecionar faixas de elementos dentro de uma lista.

```
[35] numeros = [1, 2, 3, 4, 5]
     print(numeros[1])  # Acessando o segundo elemento da lista (índice 1)
     print(numeros[2:4])  # Fatiando a lista para obter elementos do índice 2 ao 3
     print(numeros[2:])  # Fatiando a lista para obter elementos do índice 2 até o final
     print(numeros[:4])  # Fatiando a lista para obter elementos do índice 0 até o 3
     print(numeros[:])  # Fatiando a lista para obter todos os elementos

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

Listas aninhadas

Listas aninhadas

- Podemos inserir listas dentro de uma lista, assim vamos ter listas aninhadas, e para você ter acesso aos ementos da lista veja ao exemplo abaixo:

```
[26] lista_aninhada = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
     print(lista_aninhada[1][2])  # Acessando o elemento na segunda lista e terceiro elemento
     print(lista_aninhada[0][1])  # Acessando o elemento na primeira lista e segundo elemento
     print(lista_aninhada[2][0])  # Acessando o elemento na terceira lista e primeiro elemento
     print(lista_aninhada[1][0])  # Acessando o elemento na segunda lista e primeiro elemento

     6
     2
     7
     4
```

Compreensão de listas

Compreensão de listas

- Você pode utilizar o recurso de fatiamento para selecionar faixas de elementos dentro de uma lista.

```
[37] numeros = [1, 2, 3, 4, 5]
     print(numeros)
     quadrados = [x**2 for x in numeros]  # Lista com os quadrados dos números
     print(quadrados)

     [1, 2, 3, 4, 5]
     [1, 4, 9, 16, 25]
```

Ordenação de listas

Ordenação de listas

- Você pode utilizar método sort para ordenar uma lista.

```
[44] numeros = [3, 1, 4, 2, 5]
     print(numeros)
     numeros.sort()  # Ordenando a lista
     print(numeros)

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

```
[45] numeros = [3, 1, 4, 2, 5]
     print(numeros)
     numeros.sort(reverse=True)  # Ordenando a lista
     print(numeros)

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

Filtragem de listas com empressões lambda e filter()

Filtragem de listas com lambda e filter()

- A função filter() é usada para filtrar elementos de uma sequência (como uma lista, tupla ou conjunto) com base em uma função de teste. Ela retorna um iterador contendo os elementos da sequência para os quais a função de teste retorna True.
- Em Python, lambda é uma palavra-chave usada para criar funções anônimas, ou seja, funções sem um nome definido. Essas funções geralmente são usadas em situações em que você precisa de uma função temporária e simples, e criar uma função comum usando def seria excessivamente redundante.

Trabalhando com filter numa lista

- A função filter precisa de dois parâmetros, o condicional e a lista.

```
[57] def eh_negativo(x):
         return x < 0

     numeros = [-3, -2, -1, 0, 1, 2, 3]
     numeros_neg = list(filter(eh_negativo, numeros))
     print(numeros_neg)
```

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

```
[59] def eh_string_longa(s):
         return len(s) > 5

     strings = ["apple", "banana", "grape", "orange", "kiwi", "watermelon"]
     strings_longa = list(filter(eh_string_longa, strings))
     print(strings_longa)  # Saída: ['banana', 'orange', 'watermelon']
```

```
['banana', 'orange', 'watermelon']
```

Trabalhando com lambda e filter numa lista

- A função lambda é utilizada para criar uma função sem rótulo.

```
[50] numeros = [1, 2, 3, 4, 5]
     pares = list(filter(lambda x: x % 2 == 0, numeros))
     print(pares)

     [2, 4]
```

```
[60] numeros = [15, 20, 25, 30, 35, 40, 45, 50]
     divisivel_por_3_e_5 = list(filter(lambda x: x % 3 == 0 and x % 5 == 0, numeros))
     print(divisivel_por_3_e_5)

     [15, 30, 45]
```

Vamos praticar!

Link

Vamos exercitar!