Criando uma lista Python

A lista pode ser criada usando o construtor de lista ou usando colchetes `[]`.

- **Usando `list()`construtor:** Em geral, o construtor de uma classe tem seu nome de classe. Da mesma forma, crie uma lista passando os valores separados por vírgulas dentro do arquivo `list()`.
- **Usando colchetes (`[]`)** : Neste método, podemos criar uma lista simplesmente colocando os itens entre colchetes.

Vamos ver exemplos para criar a lista usando os métodos acima

```
# Using list constructor
my_list1 = list((1, 2, 3))
print(my_list1)
# Output [1, 2, 3]

# Using square brackets[]
my_list2 = [1, 2, 3]
print(my_list2)
# Output [1, 2, 3]

# with heterogeneous items
my_list3 = [1.0, 'Jessa', 3]
print(my_list3)
# Output [1.0, 'Jessa', 3]

# empty list using list()
my_list4 = list()
print(my_list4)
# Output []

# empty list using []
my_list5 = []
print(my_list4)
# Output []
```

Comprimento de uma lista

Para encontrar o número de itens presentes em uma lista, podemos usar a `len()`função.

```
my_list = [1, 2, 3]
print(len(my_list))
# output 3
```

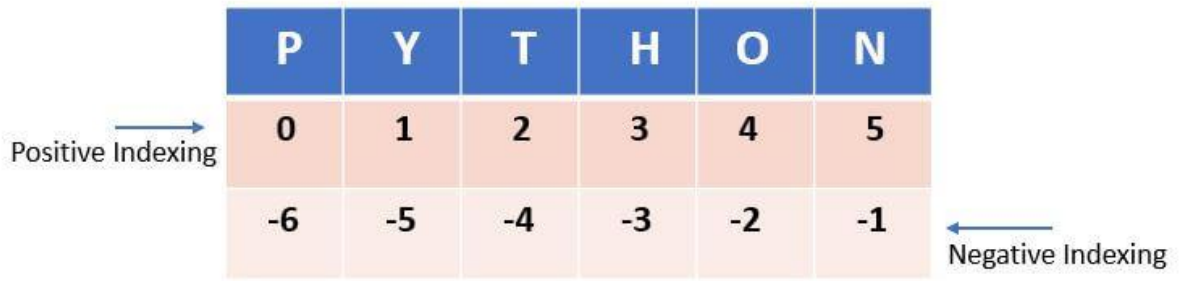
Acessando itens de uma lista

Os itens em uma lista podem ser acessados por meio de indexação e fatiamento. Esta seção irá guiá-lo acessando a lista usando as duas maneiras a seguir

- **Usando indexação** , podemos acessar qualquer item de uma lista usando seu número de índice
- **Usando Slicing** , podemos acessar uma série de itens de uma lista

Indexação

Os elementos da lista podem ser acessados usando a técnica de "indexação". As listas são coleções ordenadas com índices exclusivos para cada item. Podemos acessar os itens da lista usando este número de índice.



Indexação positiva e negativa do Python

Para acessar os elementos da lista da esquerda para a direita, o valor do índice começa de **zero** até **(comprimento da lista-1)** . Por exemplo, se quisermos acessar o 3º elemento, precisamos usar 2, pois o valor do índice começa em 0.

Nota :

- Como as Listas são sequências ordenadas de itens, os valores de índice começam de 0 até o comprimento das Listas.
- Sempre que tentamos acessar um item com um índice maior que o tamanho de Lists, ele lançará o arquivo `'Index Error'`.
- Da mesma forma, os valores de índice são sempre um número inteiro. Se dermos qualquer outro tipo, ele lançará `Type Error`.

Exemplo

```
my_list = [10, 20, 'Jessa', 12.50, 'Emma']
# accessing 2nd element of the list
print(my_list[1])  # 20
# accessing 5th element of the list
print(my_list[4])  # 'Emma'
```

Como visto no exemplo acima, acessamos o segundo elemento da lista passando o valor do índice como 1. Da mesma forma, passamos o índice 4 para acessar o 5º elemento da lista.

Indexação negativa

Os elementos da lista podem ser acessados da direita para a esquerda usando **a indexação negativa** . O valor negativo começa de -1 a -length da lista. Indica que a lista está indexada de trás para frente.

```
my_list = [10, 20, 'Jessa', 12.50, 'Emma']
# accessing last element of the list
print(my_list[-1])
# output 'Emma'

# accessing second last element of the list
print(my_list[-2])
# output 12.5

# accessing 4th element from last
print(my_list[-4])
# output 20
```

Como visto no exemplo acima para acessar o 4º elemento do último (da direita para a esquerda) passamos '-4' no valor do índice.

Listar Fatias

Fatiar uma lista implica acessar um intervalo de elementos em uma lista. Por exemplo, se queremos obter os elementos na posição de 3 a 7, podemos usar o método de fatiamento. Podemos até modificar os valores em um intervalo usando essa técnica de fatiamento.

Abaixo está a sintaxe para fatiamento de lista.

```
listname[start_index : end_index : step]
```

- O `start_index`denota a posição do índice de onde o fatiamento deve começar e o `end_index` parâmetro indica as posições do índice até as quais o fatiamento deve ser feito.
- O `step`permite que você pegue cada enésimo elemento dentro de um `start_index:end_index`intervalo.

Exemplo

```
my_list = [10, 20, 'Jessa', 12.50, 'Emma', 25, 50]
# Extracting a portion of the list from 2nd till 5th element
print(my_list[2:5])
# Output ['Jessa', 12.5, 'Emma']
```

Vamos ver mais alguns exemplos de fatiar uma lista, como

- Extraia uma parte da lista
- Inverter uma lista
- Fatiar com um passo
- Fatia sem especificar a posição inicial ou final

Exemplo

```
my_list = [5, 8, 'Tom', 7.50, 'Emma']

# slice first four items
print(my_list[:4])
# Output [5, 8, 'Tom', 7.5]

# print every second element
```

```
# with a skip count 2
print(my_list[::2])
# Output [5, 'Tom', 'Emma']

# reversing the list
print(my_list[::-1])
# Output ['Emma', 7.5, 'Tom', 8, 5]

# Without end_value
# Stating from 3nd item to last item
print(my_list[3:])
# Output [7.5, 'Emma']
```

Iterando uma lista

Os objetos na lista podem ser iterados um por um, usando um loop for a.

```
my_list = [5, 8, 'Tom', 7.50, 'Emma']

# iterate a list
for item in my_list:
    print(item)
```

Saída

```
5

8

Tom

7,5

Emma
```

Iterar junto com um número de índice

O valor do índice começa de 0 a (comprimento da lista-1). Portanto, usar a função range() é ideal para este cenário.

A função de intervalo retorna uma sequência de números. Por padrão, ele retorna a partir de 0 até o número especificado (incrementos de 1). Os valores inicial e final podem ser passados de acordo com nossas necessidades.

Exemplo

```
my_list = [5, 8, 'Tom', 7.50, 'Emma']

# iterate a list
for i in range(0, len(my_list)):
    # print each item using index number
    print(my_list[i])
```

Saída

```
5

8

Tom

7,5

Emma
```

Adicionando elementos à lista

Podemos adicionar um novo elemento/lista de elementos à lista usando os métodos de lista, como `append()`, `insert()`e `extend()`.

Anexar item no final da lista

O método append() aceitará apenas um parâmetro e o adicionará no final da lista.

Vamos ver o exemplo para adicionar o elemento 'Emma' no final da lista.

```
my_list = list([5, 8, 'Tom', 7.50])

# Using append()
```

```
my_list.append('Emma')
print(my_list)
# Output [5, 8, 'Tom', 7.5, 'Emma']

# append the nested list at the end
my_list.append([25, 50, 75])
print(my_list)
# Output [5, 8, 'Tom', 7.5, 'Emma', [25, 50, 75]]
```

Adicionar item na posição especificada na lista

Use o `insert()`método para adicionar o objeto/item na posição especificada na lista. O método insert aceita dois parâmetros posição e objeto.

```
insert(index, object)
```

Ele irá inserir o `object`no especificado `index`. Vejamos isso com um exemplo.

```
my_list = list([5, 8, 'Tom', 7.50])

# Using insert()
# insert 25 at position 2
my_list.insert(2, 25)
print(my_list)
# Output [5, 8, 25, 'Tom', 7.5]

# insert nested list at at position 3
my_list.insert(3, [25, 50, 75])
print(my_list)
# Output [5, 8, 25, [25, 50, 75], 'Tom', 7.5]
```

Como visto no exemplo acima, o item 25 é adicionado na posição de índice 2.

Usando estender()

O método extend aceitará a lista de elementos e os adicionará no final da lista. Podemos até adicionar outra lista usando este método.

Vamos adicionar três itens no final da lista.

```
my_list = list([5, 8, 'Tom', 7.50])
```

```
# Using extend()
my_list.extend([25, 75, 100])
print(my_list)
# Output [5, 8, 'Tom', 7.5, 25, 75, 100]
```

Como visto no exemplo acima, temos três valores inteiros de uma só vez. Todos os valores são adicionados na ordem em que foram passados e são anexados no final da lista.

Modificar os itens de uma lista

A lista é uma sequência mutável de objetos iteráveis. Isso significa que podemos modificar os itens de uma lista. Use o número de índice e o operador de atribuição (`=`) para atribuir um novo valor a um item.

Vamos ver como executar os dois cenários de modificação a seguir

- Modifique o item individual.
- Modificar o intervalo de itens

```
my_list = list([2, 4, 6, 8, 10, 12])

# modify single item
my_list[0] = 20
print(my_list)
# Output [20, 4, 6, 8, 10, 12]

# modify range of items
# modify from 1st index to 4th
my_list[1:4] = [40, 60, 80]
print(my_list)
# Output [20, 40, 60, 80, 10, 12]

# modify from 3rd index to end
my_list[3:] = [80, 100, 120]
print(my_list)
# Output [20, 40, 60, 80, 100, 120]
```

Modificar todos os itens

Use o loop for para iterar e modificar todos os itens de uma vez. Vamos ver como modificar cada item de uma lista.

```
my_list = list([2, 4, 6, 8])

# change value of all items
for i in range(len(my_list)):
    # calculate square of each number
    square = my_list[i] * my_list[i]
    my_list[i] = square

print(my_list)
# Output [4, 16, 36, 64]
```

Removendo elementos de uma lista

Os elementos da lista podem ser removidos usando os seguintes métodos de lista.

método	Descrição
`remove(item)`	Para remover a primeira ocorrência do item da lista.
`pop(index)`	Remove e retorna o item no índice fornecido da lista.
`clear()`	Para remover todos os itens da lista. A saída será uma lista vazia.
`del list_name`	Apague a lista inteira.

Métodos de lista do Python para remover o item

Remover item específico

Use o `remove()`método para remover a primeira ocorrência do item da lista.

Nota : Lança um `keyerror`se um item não estiver presente na lista original.

Exemplo

```
my_list = list([2, 4, 6, 8, 10, 12])

# remove item 6
my_list.remove(6)
# remove item 8
my_list.remove(8)

print(my_list)
# Output [2, 4, 10, 12]
```

Remover todas as ocorrências de um item específico

Use um loop para remover todas as ocorrências de um item específico

```
my_list = list([6, 4, 6, 6, 8, 12])

for item in my_list:
    my_list.remove(6)

print(my_list)
# Output [4, 8, 12]
```

Remover item presente em determinado índice

Use o `pop()`método para remover o item no índice fornecido. O `pop()`método remove e retorna o item presente no índice fornecido.

Nota : Ele removerá a última vez da lista se o número do índice não for passado.

Exemplo

```
my_list = list([2, 4, 6, 8, 10, 12])

# remove item present at index 2
my_list.pop(2)
```

```
print(my_list)
# Output [2, 4, 8, 10, 12]

# remove item without passing index number
my_list.pop()
print(my_list)
# Output [2, 4, 8, 10]
```

Remova o intervalo de itens

Use a palavra-chave del junto com o fatiamento da lista para remover o intervalo de itens

```
my_list = list([2, 4, 6, 8, 10, 12])

# remove range of items
# remove item from index 2 to 5
del my_list[2:5]
print(my_list)
# Output [2, 4, 12]

# remove all items starting from index 3
my_list = list([2, 4, 6, 8, 10, 12])
del my_list[3:]
print(my_list)
# Output [2, 4, 6]
```

Remover todos os itens

Use o `clear()`método list' para remover todos os itens da lista. O `clear()`método trunca a lista.

```
my_list = list([2, 4, 6, 8, 10, 12])

# clear list
my_list.clear()
print(my_list)
# Output []

# Delete entire list
del my_list
```

Encontrar um elemento na lista

Use a `index()`função para localizar um item em uma lista.

A `index()`função aceitará o valor do elemento como parâmetro e retornará a primeira ocorrência do elemento ou retornará `ValueError`se o elemento não existir.

```
my_list = list([2, 4, 6, 8, 10, 12])

print(my_list.index(8))
# Output 3

# returns error since the element does not exist in the list.
# my_list.index(100)
```

Concatenação de duas listas

A concatenação de duas listas significa a fusão de duas listas. Existem duas maneiras de fazer isso.

- Usando o `+`operador.
- Usando o `extend()`método. O `extend()`método anexa os itens da nova lista no final da lista de chamadas.

Exemplo

```
my_list1 = [1, 2, 3]
my_list2 = [4, 5, 6]

# Using + operator
my_list3 = my_list1 + my_list2
print(my_list3)
# Output [1, 2, 3, 4, 5, 6]

# Using extend() method
my_list1.extend(my_list2)
print(my_list1)
# Output [1, 2, 3, 4, 5, 6]
```

Copiando uma lista

Há duas maneiras pelas quais uma cópia de uma lista pode ser criada. Vejamos cada um com um exemplo.

Usando o operador de atribuição (`=`)

Esta é uma maneira direta de criar uma cópia. Nesse método, a nova lista será uma cópia profunda. As alterações que fizermos na lista original serão refletidas na nova lista.

Isso é chamado **de cópia profunda.**

```
my_list1 = [1, 2, 3]

# Using = operator
new_list = my_list1
# printing the new list
print(new_list)
# Output [1, 2, 3]

# making changes in the original list
my_list1.append(4)

# print both copies
print(my_list1)
# result [1, 2, 3, 4]
print(new_list)
# result [1, 2, 3, 4]
```

Como visto no exemplo acima, uma cópia da lista foi criada. As alterações feitas na lista original também são refletidas na lista copiada.

Nota: Ao definir `list1 = list2`, você está fazendo com que eles se refiram ao mesmo `list`objeto, portanto, quando você modifica um deles, todas as referências associadas a esse objeto refletem o estado atual do objeto. Portanto, não use o operador de atribuição para copiar o dicionário, em vez disso, use o `copy()` método.

Usando o método copy()

O método de cópia pode ser usado para criar uma cópia de uma lista. Isso criará uma nova lista e quaisquer alterações feitas na lista original não serão refletidas na nova lista. Isso é **cópia superficial** .

```
my_list1 = [1, 2, 3]

# Using copy() method
new_list = my_list1.copy()
# printing the new list
print(new_list)
# Output [1, 2, 3]

# making changes in the original list
my_list1.append(4)

# print both copies
print(my_list1)
# result [1, 2, 3, 4]
print(new_list)
# result [1, 2, 3]
```

Como visto no exemplo acima, uma cópia da lista foi criada. As alterações feitas na lista original não são refletidas na cópia.

Listar operações

Podemos realizar algumas operações sobre a lista usando certas funções como `sort()`, `reverse()`, `clear()`etc.

Lista de classificação usando sort()

A função de classificação classifica os elementos na lista em ordem crescente.

```
mylist = [3,2,1]
mylist.sort()
print(mylist)
```

Saída

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

Como visto no exemplo acima, os itens são classificados em ordem crescente.

Inverta uma lista usando reverse()

A função reverse é usada para inverter os elementos na lista.

```
mylist = [3, 4, 5, 6, 1]
mylist.reverse()
print(mylist)
```

Saída

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

Como visto no exemplo acima, os itens na lista são impressos na ordem inversa aqui.

Funções internas do Python com lista

Além dos métodos internos disponíveis na lista, podemos usar as funções internas também na lista. Vejamos alguns deles, por exemplo.

Usando max() e min()

A função max retorna o valor máximo na lista enquanto a função min retorna o valor mínimo na lista.

```
mylist = [3, 4, 5, 6, 1]
print(max(mylist)) #returns the maximum number in the list.
print(min(mylist)) #returns the minimum number in the list.
```

Saída

```
6

1
```

Como visto no exemplo acima, a `max` função retorna 6 e a `min`função retorna 1.

Usando soma()

A função soma retorna a soma de todos os elementos da lista.

```
mylist = [3, 4, 5, 6, 1]
print(sum(mylist))
```

Saída

```
19
```

Como visto no exemplo acima, a função sum retorna a soma de todos os elementos da lista.

all()

No caso da função all(), o valor de retorno será verdadeiro somente quando todos os valores dentro da lista forem verdadeiros. Vejamos os diferentes valores de item e os valores de retorno.

Valores do item na lista	**Valor de retorno**
Todos os valores são verdadeiros	Verdadeiro
Um ou mais valores falsos	Falso

Valores do item na lista	Valor de retorno
Todos os valores falsos	Falso
Lista vazia	Verdadeiro

```
#with all true values
samplelist1 = [1,1,True]
print("all() All True values::",all(samplelist1))

#with one false
samplelist2 = [0,1,True,1]
print("all() with One false value ::",all(samplelist2))

#with all false
samplelist3 = [0,0,False]
print("all() with all false values ::",all(samplelist3))

#empty list
samplelist4 = []
```

Saída

```
all() Todos os valores True:: True

all() com um valor falso :: False

all() com todos os valores falsos :: False

all() Lista vazia :: True
```

algum()

O método any() retornará true se houver pelo menos um valor true. No caso de Lista Vazia, retornará false.
Vamos ver a mesma combinação possível de valores para a função any() em uma lista e seus valores de retorno.

Valores do item na lista	Valor de retorno
Todos os valores são verdadeiros	Verdadeiro
Um ou mais valores falsos	Verdadeiro
Todos os valores falsos	Falso
Lista vazia	Falso

Da mesma forma, vamos ver cada um dos cenários acima com um pequeno exemplo.

```
#with all true values
samplelist1 = [1,1,True]
print("any() True values::",any(samplelist1))

#with one false
samplelist2 = [0,1,True,1]
print("any() One false value ::",any(samplelist2))


#with all false
samplelist3 = [0,0,False]
print("any() all false values ::",any(samplelist3))

#empty list
samplelist4 = []
print("any() Empty list ::",any(samplelist4))
```

Saída

```
any() Valores verdadeiros:: Verdadeiro

any() Um valor falso :: True

any() todos os valores falsos :: False

any() Lista vazia :: False
```

Lista aninhada

A lista pode conter outra lista (sub-lista), que por sua vez contém outra lista e assim por diante. Isso é chamado de lista aninhada.

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

Para recuperar os elementos da lista interna, precisamos de um arquivo `For-Loop`.

```
nestedlist = [[2,4,6,8,10],[1,3,5,7,9]]

print("Accessing the third element of the second list",nestedlist[1][2])
for i in nestedlist:
  print("list",i,"elements")
  for j in i:
    print(j)
```

Saída

```
Acessando o terceiro elemento da segunda lista 5

lista [2, 4, 6, 8, 10] elementos

2

4

6

8

10

lista [1, 3, 5, 7, 9] elementos

1

3

5

7
```

```
9
```

Como podemos ver na saída acima, a indexação das listas aninhadas com o valor de índice do loop externo primeiro seguido pela lista interna. Podemos imprimir valores das listas internas por meio de um arquivo `for-loop`.