

# ASIMOV

---

## Listas

Anteriormente, ao discutir strings, introduzimos o conceito de *sequência* em Python. As listas podem ser pensadas na versão mais geral de uma *sequência* em Python. Ao contrário das strings, eles são mutáveis, o que significa que os elementos dentro de uma lista podem ser alterados!

Nesta seção, aprenderemos sobre:      1.) Criação de listas      2.) Índice e corte de listas  
3.) Métodos básicos da lista      4.) Listas aninhadas      5.) Introdução ao método de Compreensão em listas

As listas são construídas com colchetes [] e vírgulas que separam cada elemento da lista.

Avançemos e vejamos como podemos construir listas!

```
In [1]: # Atribui uma lista a uma variável chamada my_list
        my_list = [1,2,3]
```

Acabamos de criar uma lista de números inteiros, mas as listas podem realmente armazenar diferentes tipos de objeto. Por exemplo:

```
In [2]: my_list = ['A string',23,100.232,'o']
```

Assim como as strings, a função len() irá dizer-lhe quantos itens estão na sequência da lista.

```
In [3]: len(my_list)
```

```
Out[3]: 4
```

## Indexação e corte

Indexar e cortar funciona exatamente como em strings. Vamos fazer uma nova lista para nos lembrar de como isso funciona:

```
In [7]: my_list = ['one', 'two', 'three', 4, 5]
```

```
In [10]: # Pega o elemento de índice 0
         my_list[0]
```

```
Out[10]: 'one'
```

```
In [11]: # Pegue o índice 1 e tudo depois
```

```
my_list[1:]
```

```
Out[11]: ['two', 'three', 4, 5]
```

```
In [13]: # Pega tudo até o elemento de índice 3  
my_list[:3]
```

```
Out[13]: ['one', 'two', 'three']
```

Nós também podemos usar + para concatenar listas, assim como fizemos por strings.

```
In [14]: my_list + ['new item']
```

```
Out[14]: ['one', 'two', 'three', 4, 5, 'new item']
```

Nota: Isso realmente não altera a lista original!

```
In [15]: my_list
```

```
Out[15]: ['one', 'two', 'three', 4, 5]
```

Você teria que reatribuir a lista para tornar a mudança permanente.

```
In [16]: # Reassign  
my_list = my_list + ['add new item permanently']
```

```
In [18]: my_list
```

```
Out[18]: ['one', 'two', 'three', 4, 5, 'add new item permanently']
```

Nós também podemos usar o \* para um método de duplicação semelhante às strings:

```
In [20]: # Make the list double  
my_list * 2
```

```
Out[20]: ['one',  
          'two',  
          'three',  
          4,  
          5,  
          'add new item permanently',  
          'one',  
          'two',  
          'three',  
          4,  
          5,  
          'add new item permanently']
```

```
In [4]: my_list
```

```
Out[4]: ['A string', 23, 100.232, 'o']
```

## Métodos de Lista Básica

Se você está familiarizado com outra linguagem de programação, você pode começar a comprar arrays em outro idioma e listas em Python. As listas em Python, no entanto, tendem a ser mais flexíveis do que arrays em outras línguas por dois bons motivos: eles não têm tamanho fixo (o que significa que não precisamos especificar o tamanho de uma lista), e eles não têm restrição de tipo fixo (como já vimos acima).

Vamos prosseguir e explore alguns métodos mais especiais para listas:

```
In [6]: # Cria a lista  
l = [1,2,3]
```

Use o método **append** para adicionar permanentemente um item ao final de uma lista:

```
In [7]: # Append  
l.append('append me!')
```

```
In [8]: # Mostra  
l
```

```
Out[8]: [1, 2, 3, 'append me!']
```

Use **pop** para "retirar" um item da lista. Por padrão, pop tira o último índice, mas também pode especificar qual índice aparecer. Vamos ver um exemplo:

```
In [9]: # Retira o item de índice 0  
l.pop(0)
```

```
Out[9]: 1
```

```
In [46]: # Mostra  
l
```

```
Out[46]: [2, 3, 'append me!']
```

```
In [10]: # Atribui o elemento retirado, lembre-se de que o índice padrão é -1  
popped_item = l.pop()
```

```
In [48]: popped_item
```

```
Out[48]: 'append me!'
```

```
In [49]: # Mostra a lista restante  
l
```

```
Out[49]: [2, 3]
```

Também deve notar-se que a indexação das listas retornará um erro se não houver nenhum elemento nesse índice. Por exemplo:

```
In [11]: l[100]
```

```
-----  
IndexError                                Traceback (most recent call last)  
<ipython-input-11-e2a0c2623844> in <module>()  
----> 1 l[100]
```

**IndexError:** list index out of range

Podemos usar o método **sort** e os métodos **reverse** para alterar suas listas:

```
In [13]: new_list = ['a', 'e', 'x', 'b', 'c']
```

```
In [14]: new_list
```

```
Out[14]: ['a', 'e', 'x', 'b', 'c']
```

```
In [15]: # Use o reverso para reverter a ordem (isto é permanente!)  
new_list.reverse()
```

```
In [16]: new_list
```

```
Out[16]: ['c', 'b', 'x', 'e', 'a']
```

```
In [55]: # Use ordenar para classificar a lista (neste caso, ordem alfabética)  
new_list.sort()
```

```
In [56]: new_list
```

```
Out[56]: ['a', 'b', 'c', 'e', 'x']
```

## Listas aninhadas

Uma ótima característica das estruturas de dados do Python é que eles suportam *aninhamento*. Isso significa que podemos ter estruturas de dados dentro das estruturas de dados. Por exemplo: uma lista dentro de uma lista.

Vamos ver como isso funciona!

```
In [57]: # Começamos com 3 listas  
lst_1=[1,2,3]  
lst_2=[4,5,6]  
lst_3=[7,8,9]  
  
# Faça uma lista de listas para formar uma matriz  
matrix = [lst_1,lst_2,lst_3]
```

```
In [60]: # Mostra  
matrix
```

```
Out[60]: [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Agora, podemos usar novamente a indexação para pegar elementos, mas agora existem dois

níveis para o índice. Os itens no objeto matriz e, em seguida, os itens dentro dessa lista!

```
In [61]: # Pegue o primeiro item no objeto da matriz  
matrix[0]
```

```
Out[61]: [1, 2, 3]
```

```
In [62]: # Pegue o primeiro item do primeiro item no objeto da matriz  
matrix[0][0]
```

```
Out[62]: 1
```

## Compreensão de listas

Python possui um recurso avançado chamado compreensões de lista. Eles permitem a construção rápida de listas. Para entender completamente as compreensões da lista, precisamos entender os loops. Portanto, não se preocupe se você não entender completamente esta seção e sinta-se à vontade para ignorá-la, pois retornaremos a esse assunto mais tarde.

Mas no caso de você querer saber agora, aqui estão alguns exemplos!

```
In [63]: # Crie uma lista de compreensão desconstruindo um loop for dentro de um []  
first_col = [row[0] for row in matrix]
```

```
In [64]: first_col
```

```
Out[64]: [1, 4, 7]
```

Usamos a compreensão da lista aqui para pegar o primeiro elemento de cada linha no objeto da matriz. Vamos abordar isso com muito mais detalhes mais tarde!

Para obter métodos e recursos mais avançados das listas em Python, consulte a seção de lista avançada mais adiante neste curso!