Funções de listas

As listas possuem diversas funções prontas bastante úteis. Veremos algumas das mais usadas. Não se preocupe em *decorar* todas elas: sempre podemos consultar nosso material quando precisarmos de um lembrete! Com tempo e prática você irá aos poucos memorizar algumas delas.

1. Adicionando elementos

Podemos adicionar novos elementos na lista de duas maneiras. A primeira delas, mais simples, é o *append*. Ele adiciona um elemento ao final da lista. Veja o exemplo abaixo:

```
pares = [0, 2, 4, 6, 8]
pares.append(10)
print(pares) # resultado: [0, 2, 4, 6, 8, 10]
```

Outra maneira é com o *insert*: além do elemento, ele recebe a **posição** do novo elemento. O primeiro parâmetro é a posição, e a segunda é o valor.

```
pares = [0, 2, 4, 8, 10]
pares.insert(3, 6)
print(pares) #resultado: [0, 2, 4, 6, 8, 10]
```

Note que o valor que ocupava a posição anteriormente não é substituído, mas "empurrado" para a próxima posição.

2. Removendo elementos

Podemos remover o elemento de 2 jeitos: por **valor** e por **posição**. O *remove* irá remover o primeiro elemento encontrado na lista com um dado valor. Ex:

```
impares = [1, 3, 3, 5, 7, 9]
impares.remove(3)
print(impares) # resultado: [1, 3, 5, 7, 9]
```

O *pop* remove o elemento que estiver em uma dada posição, independentemente de seu valor:

```
impares = [1, 3, 5, 7, 8, 9]
impares.pop(4)
print(impares) # resultado: [1, 3, 5, 7, 9]
```

Se nenhum valor for passado no pop, ele irá remover necessariamente o último elemento da lista.

3. Ordenando a lista

Podemos ordenar a lista usando o *sort*.

```
fibonacci = [8, 1, 0, 5, 13, 1, 3, 2]
fibonacci.sort()
```

```
print(fibonacci) # resultado: [0, 1, 1, 2, 3, 5, 8, 13]
```

Caso desejássemos ordenar em ordem decrescente, podemos passar a opção `reverse = True` para o *sort*:

```
fibonacci = [8, 1, 0, 5, 13, 1, 3, 2]
fibonacci.sort(reverse = True)
print(fibonacci) # resultado: [13, 8, 5, 3, 2, 1, 1, 0]
```

> **Importante:** o *sort* só irá funcionar caso sua lista possua apenas elementos que podem ser comparados entre si (apenas *strings* ou apenas números, por exemplo). Se uma lista contém tanto *strings* quanto números, o *sort* não saberá o que vem primeiro.

Um problema do *sort* é que ele irá reordenar a própria lista. Muitas vezes precisamos preservar a lista original, e obter uma nova com a ordenação desejada. Neste caso, podemos utilizar o *sorted*:

```
fibonacci = [8, 1, 0, 5, 13, 1, 3, 2]
fibonacci_ordenada = sorted(fibonacci)
print(fibonacci_ordenada, fibonacci) # resultado: [0, 1, 1, 2, 3, 5, 8, 13] [8, 1, 0, 5, 13, 1, 3, 2]
```

O *sorted* também aceita o parâmetro *reverse* para aplicar ordem decrescente.

Falando em ordem decrescente, é possível simplesmente inverter a ordem dos elementos de uma lista utilizando o *reverse*:

```
lista = [1, 5, 'dois', 4, 3.14]
lista.reverse()
print(lista) # resultado: [3.14, 4, 'dois', 5, 1]
```

4. Buscando um elemento

Podemos buscar um elemento (descobrir sua posição) utilizando a função *index*. Ela irá informar a **primeira** posição onde um elemento for encontrado:

```
pi = [3, 1, 4, 1, 5, 9, 2, 6, 5]
posicao = pi.index(5)
print(posicao) # resultado: 4
```

> **Atenção:** caso o elemento buscado não exista, a função *index* dará erro. Considere utilizar o *in* para verificar se o elemento existe na lista antes de usar o *index*.

5. Informações sobre a lista

Podemos obter alguns dados sobre a nossa lista: seu tamanho atual (*len*), seu maior valor (*max*) e seu menor valor (*min*).

```
pi = [3, 1, 4, 1, 5, 9, 2, 6]
tamanho = len(pi)
maior = max(pi)
menor = min(pi)
print(tamanho, maior, menor) # resultado: 8 9 1
```