Définition

Les listes permettent de **représenter des collections d'objets** dans Python. Elles sont appelées `array` ou tableau dans PHP, ou Javascript. Elles vont pouvoir contenir des types d'objets différents ou non et on va pouvoir faire des opérations d'ajout, de suppression, de modification.

Les listes reposent sur un système d'index ce qui en fait des objets **ordonnés**. Les fonctions qui sont appliquées à une liste vont changer la valeur de la liste en mémoire, c'est un objet muable.

```
liste_vide    = []
liste_ints    = [1, 2, 3, 4, 5]
liste_strings = ["coucou", "ca", "va", '?']
liste_mix     = [1, "Super", True, [1, 2, "Bonjour"]]
```

Ajouter ou enlever des éléments

`append()` et `extend()`

Append permet d'ajouter un élément à une liste en bout d'une liste, alors qu'extend permet d'en ajouter plusieurs.

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

ma_liste.append(4)
print(ma_liste) #  [1, 2, 3, 4]

ma_liste.extend([5, 6, 7])
print(ma_liste) #  [1, 2, 3, 4, 5, 6, 7]
```

`remove()`

Supprime la première occurrence d'un élément à une liste

```
ma_liste = [1, 2, 3, 4, 3]

ma_liste.remove(3)
print(ma_liste) # [1, 2, 4, 3]
```

pop()

Enlève l'élément à un index donné. Il permet aussi de supprimer le dernier élément si aucun index n'est fournis.

```
liste = [1, 2, 3, 4, 3]

liste.pop(2) # Suppression de l'élément à l'index 2
print(liste) # [1, 2, 4, 3]

liste.pop() # Suppression du dernier élément
print(liste) # [1, 2, 4]
```

Récupérer des éléments d'une liste

Pour accéder à un élément d'une liste on utilise des index qui représentent l'ordre dans lequel l'élément est disposé.

```
ma_liste = [1, 2, 3, 4, 5, [10, 11]]

print(ma_liste[3])     # 4
print(ma_liste[5][1])  # 11
print(ma_liste[-1])    # Retourne le dernier élément de la liste
print(ma_liste[-2])    # Retourne l'avant-dernier élément de la liste
```

> ⚠️ Index
>
> L'indice du premier élément d'une liste est toujours `0`.

Les slices

Les slices permettent de sélectionner une sous-partie d'une liste. Celle ci peut être vide. Il est composé de 2 indexes, le premier inclusif qui définis le debut, le second exclusif qui définis le premier élément a ne pas prendre. Ces indexes sont séparées par des `:`.

```
ma_liste = [1, 2, 3, 4, 5, 6]
print(ma_liste[3:5])   # [4, 5] - exlusion de l'index 5
```

Pour nous faciliter l'écriture, les slicers permettent d'omettre les index : - de début, pour récupérer tout jusqu'à l'index de fin - de fin, pour récupérer toute la fin de la liste après l'index de debut

```
print(ma_liste[:2])    # [1, 2] - depuis le début jusqu'à l'index 2 exclu
print(ma_liste[2:])    # [3, 4, 5, 6] - depuis l'index 2 jusqu'à la fin
print(ma_liste[:])     # [1, 2, 3, 4, 5, 6] - toute la liste ce qui est equivalent
à afficher la liste sans slicer
```

Pour accroitre encore cette facilité d'écriture, on a accès aux chiffres négatifs. Cela nous permet facilement d'écarter N-nombres sans devoir faire `len(ma_liste) - N`

```
print(ma_liste[2:-1])  # [3, 4, 5] - depuis l'index 2 jusqu'au dernier élément
exclus (-1 représente le dernier élément)
print(ma_liste[-5:5])  # [2, 3, 4, 5] - depuis l'index -5 jusque l'index 5.
```

Il existe un troisième index qui permet de définir le pas. Cette index est optionnel et est définis à 1 par défaut

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

print(ma_liste[::1])   # [1, 2, 3, 4, 5, 6] - entièreté de la liste avec un pas de
1, donc affichage normal
print(ma_liste[::2])   # [1, 3, 5] - entièreté de la liste avec un pas de 2 (1
chiffre sur 2)
print(ma_liste[2:5:2]) # [3, 5] depuis l'index 2 jusqu'à l'index 5 avec un pas de
2
print(ma_liste[::-1])  # [6, 5, 4, 3, 2, 1] - entièreté de la liste en sens
inverse
```

Diverses méthodes de liste

`len()`

Retourne la longueur d'une liste ou d'une string

```
longueur = len([1, 2, 3, 4, 5])
print(longueur) # 5
```

`index()` Retourne l'index d'un élément dans une liste

```
liste = [1, 2, 3]
print(liste.index(2)) # 1
```

`count()` Retourne le nombre de fois qu'un élément est dans une liste

```
liste = ["coucou", "pepito", "coucou"]
print(liste.count("coucou")) # 2
```

`sort()` Trie une liste en fonction du type d'éléments

```
liste_str = ['a', 'b', 'z', 'c', 'r', 'm', 'y']
liste_int = [23, 1, 2, 75, 54, 2,42]
liste_str.sort()
liste_int.sort()

print(liste_str) # ['a', 'b', 'c', 'm', 'r', 'y', 'z']
print(liste_int) # [1, 2, 2, 23, 42, 54, 75]
```

> 📌 Fonctionnement de la fonction
>
> La fonction `sort()` ne retourne rien, quand elle est appelée, elle va trier la liste "in place". Pour stocker le résultat d'un tri d'une liste, utiliser la fonction `sorted()`.

`filter()` Filtre une liste en utilisant une fonction

```
ma_liste = [1, 2, 33, 477, 5, 61, 73, 8]

# Filtre de la liste en ne gardant que les valeur supérieure à 100
filtre = filter(lambda x: x > 100, ma_liste)

# Retourne un objet filter, doit être reconverti en liste
print(list(filtre))
```

`join() et split()`

Join permet de joindre les éléments d'une liste dans une même string, tandis que split permet de séparer une string en une liste. (cf. 8. Les méthodes de chaînes de caractères).

Les ranges

Retourne un objet de type `range` qui, converti en liste, vous donne une suite de nombres avec un incrément de 1, commençant à la valeur `x` jusqu'à la valeur `y` exclue. Un troisième paramètre peut être ajouté pour paramétrer le pas.

```
range_pas_1 = range(1, 10)
print(list(range_pas_1)) # [1, 2, 3, 4, 5, 6, 7, 8, 9]

range_pas_2 = range(1, 10, 2)
print(list(range_pas_2)) # [1, 3, 5, 7, 9]
```

⚠️ Il faut transformer le résultat de `range` en liste avec la fonction `list()`, sinon l'affichage vous surprendra !

Exercice: "14. Les listes"