



Les listes

Samuel SORIN
contact@samuelsorin.fr

Les listes

Les listes - création ⁽¹⁾

Les listes permettent de regrouper plusieurs valeurs séparées par des virgules.

Une liste est définie par des crochets **[]**

Il est possible de mélanger dans une même liste des valeurs de type différent.

```
>> ma_liste = []
```

```
>> ma_liste2 = [1, 'a', 'b', 4]
```

```
>> ma_liste3 = [1, [2, 3], 4, 'a', ma_liste2]
```

Les listes - Ajouter une entrée ⁽²⁾

On peut ajouter des valeurs :

- lors de la création de la liste,
- avec la méthode **append**,

Les nouvelles valeurs sont toujours insérées en fin de liste.

```
>>> ma_liste = [1,2,3]
>>> ma_liste.append(1)
>>> ma_liste.append("ok")
```

```
>>> ma_liste
>>> [1, 2, 3, 1, "ok"]
```

On peut ajouter une liste dans une liste. Mais la liste ajoutée correspondra à une seule entrée de la liste de départ

```
>>> ma_liste2 = [4, 5, 6]
>>> ma_liste.append(ma_liste2)
>>> ma_liste
[1, 2, 3, 1, "ok", [4, 5, 6] ]
```

Les listes - Ajouter une entrée (2b)

On peut ajouter des valeurs :

- avec la méthode **insert(*i*, valeur)**, où *i* est l'index où insérer la valeur

```
>> ma_liste = [1,3,4]
>> ma_liste.insert(1, 'b')    # on insert 'b' à l'index 1 de la liste ma_liste

>> ma_liste
>> [1, b, 3, 4]
```

Les listes - récupérer / afficher les valeurs (3a)

- On peut récupérer une valeur de la liste par son index (*Le premier item commence toujours avec l'index 0*)

```
>> ma_liste = ['a', 'b', 'c']  
>> ma_liste[0]  
'a'  
  
>> >> ma_liste[2]  
'c'
```

Les listes - récupérer / afficher les valeurs (3b)

- en utilisant une boucle

```
>> ma_liste = ['a', 'b', 'c']  
>>> for lettre in ma_liste:  
...     print(lettre)  
  
a  
b  
c
```

Les listes - Modifier les entrées ⁽⁴⁾

en indiquant l'index de la valeur à modifier et en y assignant une nouvelle valeur

```
>> ma_liste = ['a', 'b', 'c']  
  
>> ma_liste[1] = 'BONJOUR'  
>> ma_liste  
  
['a', 'BONJOUR', 'c']
```


Les listes - Supprimer des entrées avec son index (5a)

- Supprimer une entrée avec son index avec la fonction **del**

```
>> ma_liste = ['a', 'b', 'c']  
>> del(ma_liste[1])  
>> ma_liste  
  
['a', 'c']
```

Les listes - Supprimer des entrées avec sa valeur (5b)

- Supprimer une entrée par sa valeur avec la méthode **remove()**
- Si plusieurs valeurs identiques existent, seule la première est supprimée.

```
>> ma_liste = ['a', 'b', 'c']  
>> ma_liste.remove('a')  
>> ma_liste  
  
['b', 'c']
```

Les listes - Supprimer des entrées - pop (5c)

Pour supprimer une valeur de la liste, on peut utiliser la méthode **pop()**

Par défaut, **pop** supprime la dernière valeur de la liste.

On peut indiquer un index à **pop** pour supprimer une valeur d'un index en particulier.

Pop a pour particularité de renvoyer la valeur qui a été supprimée.

```
>> ma_liste = ['a', 'b', 'c']
>> ma_liste.pop()      #supprimer la dernière valeur
'c'                    #renvoie la valeur supprimée
>> ma_liste

['a', 'b']
```

```
>> ma_liste = ['a', 'b', 'c']
>> ma_liste.pop(1)     #supprimer la valeur de l'index
1                        #renvoie la valeur supprimée
'b'
>> ma_liste

['a', 'c']
```

Les listes - vider une liste (6)

La méthode **clear()** permet de vider complètement une liste

```
>> ma_liste = ['a', 'b', 'c']  
>> ma_liste.clear()  
>> ma_liste
```

```
[]
```

Les listes - Vérifier si une valeur existe ⁽⁷⁾

- Le mot clé **in** permet de savoir si un élément est dans une liste

```
>>> ma_liste = [1, 10, 100, 250, 500]
```

```
>>> 10 in ma_liste
```

```
True
```

```
>>> 99 in ma_liste
```

```
False
```

Exemple d'utilisation :

```
ma_liste = [1, 10, 100, 250, 500]
```

```
if 10 in ma_liste:
```

```
    print("Oui le chiffre 10 est bien dans la liste")
```

Exercice 2

Créer un programme permettant de gérer une liste de course.
Le programme doit permettre les actions suivantes :

- A = Ajouter un élément à la liste
- S = Supprimer un élément à la liste
- V = Vider la liste
- E = Mettre fin au programme

- > faire la structure du programme
- > Ajouter les actions possibles (A, S, V, E) dans une list
- > tester si l'action de l'utilisateur est bien autorisée grâce à cette liste

Les listes - Méthodes et fonctions utiles ⁽⁸⁾

Récupérer l'index d'une valeur :

- La méthode **index** vous permet de connaître la position de l'item recherché.
- Attention, l'index renvoyé est celui de la 1ere valeur trouvée
- Il est possible de donner l'index de départ à partir duquel chercher l'item

```
>>> ma_liste = ["a","a","a","b","c","c", "a"]
>>> ma_liste.index("b")
3

>>> ma_liste.index("a")
0

>>> ma_liste.index("a", 4)
>> 6
```

Les listes - Méthodes et fonctions utiles ⁽⁹⁾

- La fonction **len** permet de compter le nombre d'éléments

```
>>> ma_liste = ["a","b","c","d"]  
>>> len(ma_liste)
```

```
4
```


Les listes - Méthodes et fonctions utiles (10)

- La méthode **count** permet de connaître le nombre d'occurrences d'une valeur donnée

```
>>> ma_liste = ["a", "a", "a", "a", "b", "c", "d", "d"]  
>>> ma_liste.count("a")
```

```
4
```

Les listes - Méthodes et fonctions utiles

- La méthode **reverse** permet d'inverser les éléments d'une liste

```
>>> ma_liste = ["a", "b", "c"]  
>>> ma_liste.reverse()  
>>> ma_liste  
  
['c', 'b', 'a']
```

Les listes - Méthodes et fonctions utiles

- Une liste ne peut pas être copiée directement
- Il faut utiliser la méthode **copy**

```
>>> ma_liste = ["a", "b", "c", "d"]  
>>> ma_liste2 = ma_liste.copy()
```

Exercice 3

Reprendre le programme de liste de course et ajouter l'action :

- L = Lister les éléments de la liste et les compter
- C = Check qui permet de vérifier si un item n'est pas présent plusieurs fois. Si c'est le cas, supprimer l'item en trop

Les listes - Méthodes et fonctions utiles (11)

- Pour concaténer 2 listes, on peut utiliser l'addition **+** ou la méthode **extend**

```
>>> x = [1, 2, 3]
>>> y = [4, 5, 6]
>>> x + y

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

```
>>> x = [1, 2, 3]
>>> y = [4, 5, 6]
>>> x.extend(y)

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

Les listes - Méthodes et fonctions utiles

- on peut multiplier une liste
- utilisé pour initialiser une nouvelle liste

```
>>> ma_liste = [0] * 5
```

```
>>> ma_liste
```

```
[0, 0, 0, 0, 0]
```

Les listes - récupérer / afficher les valeurs - 2eme partie ⁽¹²⁾

- les listes peuvent être indicées et découpées
- Toutes les opérations par tranches `[i:i2]` renvoient une nouvelle liste
- `ma_liste[i:i]` se lit : “de l’index *i* à l’index *i2* non compris”
- Si on ne précise pas le 1er index, l’affichage se fait automatiquement depuis l’index 0
- Si on n’affiche pas le dernier index (i2), l’affichage se fait jusqu’à la fin de la liste

Les listes - récupérer / afficher les valeurs - 2eme partie (4b)

```
>>> ma_liste = [1, 10, 100, 250, 500]

>>> ma_liste[2:4]          # Affiche les valeurs à partir de l'index 2 jusqu'à l'index 4 Attention, l'index 4 n'est pas compris dans le résultat
[100, 250]

>>> ma_liste[-1]          # Affiche la 1ère occurrence à partir de la droite
500

>>> ma_liste[-4]          # Affiche les 4eme occurrence à partir de la droite
10

>>> ma_liste[-4:]         # affiche une liste avec les 4 dernières occurrences de droite
[10, 100, 250, 500]

>>> ma_liste[:]           # Affiche toutes les occurrences
[1, 10, 100, 250, 500]

>>> ma_liste = [1, 10, 100, 250, 500, 1000, 2000, 3000]
>>> ma_liste[2:-2]        # Affiche les occurrences à partir de l'index 2 jusqu'à la 2eme occurrence(non comprise) à partir de la droite
[100, 250, 500, 1000]
```


Les listes - Méthodes et fonctions utiles

- La fonction **range** est un itérable, elle permet de générer automatiquement une liste arithmétique
- Par exemple, `range(10)` génère une liste qui va de 0 à 9
- **Attention**, l'objet renvoyé par `range()` se comporte presque comme une liste, mais ce n'en est pas une.
- Il est possible de spécifier une valeur de début et/ou une valeur d'incrément
exemple : `range(3, 10, 2)`
- `range(start, stop, step)`
- Afin d'afficher une liste, il est nécessaire d'utiliser *list()* (cf exemple ci-dessous)

Les listes - Méthodes et fonctions utiles

```
>>> for i in range(4):
```

```
...     print(i)
```

```
0
```

```
1
```

```
2
```

```
3
```

```
>>> list(range(10))
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
>>> list(range(3, 10))
```

```
[3, 4, 5, 6, 7, 8, 9]
```

```
>>> list(range(3, 10, 2))
```

```
[3, 5, 7, 9]
```

```
>>> list(range(0, -10, -1))
```

```
[0, -1, -2, -3, -4, -5, -6, -7, -8, -9]
```

Les listes - string et liste (13a / 13b)

- Transformer une string en liste avec la méthode **split**

```
>>> ma_chaine = "Banane; Kiwi; pomme"
>>> ma_chaine.split(";")
['Banane', 'Kiwi', 'pomme']
```

- Transformer une liste en string avec la méthode **join**

```
>>> ma_liste = ['Banane', 'Kiwi', 'pomme']
>>> "-".join(ma_liste)
'Banane - Kiwi - pomme'
```

Exercice 2

Reprendre la liste de course et ajouter la possibilité d'ajouter plusieurs items à la fois dans la liste