LES LISTES PYTHON

PRESENTATION

Une **Liste Python** sont utilisées pour **stocker plusieurs éléments dans une seule variable**.

Les listes sont l'un des 4 types structurés utilisés par Python pour stocker des séries de données. Les 3 autres types sont « **Tuple »**, « **Set »** et « **Dictionnaire »**, avec des caractéristiques et usages différents.

CREATION D'UNE LISTE

Les listes sont créées avec des **crochets** :

```
maListe = ["pomme", "banane", "cerise"]
print(maListe)
```

```
['pomme', 'banane', 'cerise']
```

Création d'une liste vide:

```
maListe = []
```

Le constructeur **`list()`** permet aussi de créer une nouvelle liste.

```
maListe = list(("pomme", "banane", "cerise")) #attention : doubles parenthèses
```

ÉLEMENTS DE LA LISTE

Les éléments d'une liste sont **indexés**.

Le premier élément possède l'index 0, le deuxième élément possède l'index 1, etc.

```
maListe = ["pomme", "banane", "cerise"]
print(maListe[0]) #affiche le premier élément de la liste
```

```
pomme
```

La fonction **`len()`** permet de déterminer le nombre d'éléments d'une liste.

```
maListe = ["pomme", "banane", "cerise"]
print(len(maListe)) #affiche la longueur de la liste
```

```
3
```

Il est possible de **modifier**, **ajouter** ou **supprimer** des éléments d'une liste.

TYPES DE DONNEES

Les éléments de liste peuvent être de tout type :

```
L1 = ["pomme", "banane", "cerise"] # liste de chaines de caractères
L2 = [1, 5, 7, 9, 3] # liste d'entiers
L3 = [True, False, False] # liste de booléens
L4 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]] # liste de listes
L5 = [(1, 2, 3), (4, 5, 6), (7, 8, 9)] # liste de tuples
```

Une liste peut contenir différents types de données :

```
L4 = ["abc", 34, True, 40, "male", [1, 2, 3]]
```

REMPLACER DES VALEURS DANS UNE LISTE

On peut remplacer des éléments spécifiés par leur index :

```
maListe = ["pomme", "banane", "cerise", "orange", "kiwi", "mangue"]
maListe[1:3] = ["orange", "citron"] # "orange", "citron" remplace "banane", "cerise"
print(maListe)
                              ['pomme', 'orange', 'citron', 'orange', 'kiwi', 'mangue']
```

On peut remplacer par un nombre différent d'éléments (les éléments en place peuvent être déplacés) :

```
maListe = ["pomme", "banane", "cerise"]
maListe[1:2] = ["orange", "citron"] # "orange", "citron" remplace "banane"
print(maListe)
                                                ['pomme', 'orange', 'citron', 'cerise']

maListe = ["pomme", "banane", "cerise"]
maListe[1:3] = ["orange"] # "orange" remplace "banane", "cerise"
print(maListe)
                                                                    ['pomme', 'orange']
```

AJOUTER DES ELEMENTS A UNE LISTE

La méthode **`append()`** permet d'insérer des nouveaux éléments **en fin de liste**.

```
maListe = ["pomme", "banane", "cerise"]
maListe.append("orange")
print(maListe)
                                                ['pomme', 'banane', 'cerise', 'orange']
```

INSERER DES ELEMENTS DANS UNE LISTE

La méthode **`insert()`** permet d'insérer un nouvel élément **à une position précise**.

Insérez "pastèque" comme troisième élément :

```
maListe = ["pomme", "banane", "cerise"]
maListe.insert(2, "orange") # "orange" est inséré à la 3eme position
print(maListe)
                                                ['pomme', 'banane', 'orange', 'cerise']
```

SUPPRIMER UN ELEMENT SPECIFIE PAR SA VALEUR

La méthode **remove()**permet de supprimer un l'élément spécifié par sa valeur.

```
maListe = ["pomme", "banane", "cerise"]
maListe.remove("banane")
print(maListe)
```

```
['pomme', 'cerise']
```

Attention : seul le premier élément trouvé est supprimé.

```
maListe = ["pomme", "banane", "cerise", "banane"]
maListe.remove("banane")
print(maListe)
```

```
['pomme', 'cerise', 'banane']
```

SUPPRIMER UN ELEMENT SPECIFIE PAR SON INDEX

Remarque : le mot clé **del** permet de supprimer un élément spécifié par son index.

```
maListe = ["pomme", "banane", "cerise"]
del maListe[0]
print(maListe)
```

```
['banane', 'cerise']
```

SUPPRIMER UNE LISTE

del permet de supprimer complètement une liste.

```
maListe = ["pomme", "banane", "cerise"]
del maListe
print(maListe)
```

```
NameError: name 'maListe' is not defined
```

VIDER UNE LISTE

La méthode **clear()** vide la liste (mais ne la supprime pas, contrairement à del).

```
maListe = ["pomme", "banane", "cerise"]
maListe.clear()
print(maListe)
```

```
[]
```

COMPTER DES ELEMENTS

La méthode **count(val)** compte le nombre d'éléments qui possèdent la valeur « val ».

```
L = [3,1,2,3]
print(L.count(3))
```

```
2
```

DONNER L'INDEX D'UNE VALEUR

La méthode **index(val)** donne l'index du premier élément qui possède la valeur « val ».

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

```
1
```

EXTRAIRE UN ELEMENT SPECIFIE PAR SON INDEX

La méthode **pop(index)** permet d'extraire un élément spécifié par son index.

```
maListe = ["pomme", "banane", "cerise"]
maListe.pop(1) #on extrait banane de maListe
print(maListe)
```

```
['pomme', 'cerise']
```

pop() extrait le dernier élément de la liste

```
maListe = ["pomme", "banane", "cerise"]
maListe.pop()
print(maListe)
```

```
['pomme', 'banane']
```

Important : avec la méthode **pop()** on peut récupérer l'élément qui a été extrait.

```
maListe = ["pomme", "banane", "cerise"]
x = maListe.pop()
print(maListe,x,sep=" *** ")
```

```
['pomme', 'banane'] *** cerise
```

L'utilisation simultanée de **pop()** et **append()** permet de transférer un élément d'une liste à une autre :

```
L1 = ["pomme", "banane", "cerise"]
L2 = ["orange", "citron", "poire"]
L2.append(L1.pop(0)) #le premier élément de L1 est ajouté à L2
print(L1,L2,sep=" * ")
```

```
['banane', 'cerise'] * ['orange', 'citron', 'poire', 'pomme']
```

PARCOURIR UNE LISTE AVEC UNE BOUCLE « FOR »

On peut parcourir la liste en se référant aux valeurs des éléments :

```
maListe = ["pomme", "banane", "cerise"]
for x in maListe:
  print(x)
```

```
pomme
banane
cerise
```

On peut également parcourir la liste en se référant aux index des éléments :

```
maListe = ["pomme", "banane", "cerise"]
for i in range(len(maListe)):
  print(maListe[i])
```

```
pomme
banane
cerise
```

PARCOURIR UNE LISTE AVEC UNE BOUCLE « WHILE »

```
maListe = ["pomme", "banane", "cerise"]
i = 0
while i < len(maListe):
  print(maListe[i])
  i = i + 1
```

```
pomme
banane
cerise
```

LA COMPREHENSION DE LISTE

La **compréhension de liste** offre une syntaxe plus courte pour créer de nouvelles listes ou pour parcourir des listes.

On peut utiliser la compréhension de liste pour parcourir une liste :

```
maListe = ["pomme", "banane", "cerise"]
[print(x) for x in maListe if "a" in x]
```

```
banane
```

Programme équivalent sans compréhension :

```
maListe = ["pomme", "banane", "cerise"]
for x in maListe:
    if "a" in x:
        print(x)
```

```
banane
```

Syntaxe de la création de liste par compréhension :

liste2 = [expression **for** element **in** liste1 **if condition**]

liste2 est une nouvelle liste, laissant liste1 inchangée.

```
L1 = ["pomme", "banane", "cerise", "orange", "citron"]
L2 = [x for x in L1 if "a" in x] #liste des éléments de L1 qui contiennent « a »
print(L2)
```

```
['banane', 'orange']
```

```
L3 = [x for x in L1 if x != "pomme"] #liste des éléments qui ne sont pas "pomme"
```

Remarque : La condition est facultative.

```
maListe = [x for x in range(10)]
print(maListe)
```

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

```
maListe = [x for x in range(10) if x % 2 != 0]
print(maListe)
```

```
[1, 3, 5, 7, 9]
```

```
maListe = [x**2 for x in range(10) if x % 2 != 0]
print(maListe)
```

```
[1, 9, 25, 49, 81]
```

```
maListe = [x//2 if x % 2 == 0 else 3*x+1 for x in range(1,8)]
print(maListe)
```

```
[4, 1, 10, 2, 16, 3, 22]
```

TRI ALPHANUMERIQUE OU NUMERIQUE

La méthode **`sort()`** permet de trier une liste par ordre alphabétique ou croissant :

```
maListe = ["orange", "pomme", "poire", "banane", "citron"]
maListe.sort()
print(maListe)
```

```
['banane', 'citron', 'orange', 'poire', 'pomme']
```

```
maListe = [412, 73, 36, 5, 59]
maListe.sort()
print(maListe)
```

```
[5, 36, 59, 73, 412]
```

Pour inverser l'ordre de tri, il faut ajouter l'argument **`reverse = True`** :

```
maListe = ["orange", "pomme", "poire", "banane", "citron"]
maListe.sort(reverse = True)
print(maListe)
```

```
['pomme', 'poire', 'orange', 'citron', 'banane']
```

```
maListe = [412, 73, 36, 5, 59]
maListe.sort(reverse = True)
print(maListe)
```

```
[412, 73, 59, 36, 5]
```

Par défaut, la méthode **`sort()`** est sensible à la casse,
ce qui entraîne le tri de toutes les lettres majuscules avant les lettres minuscules :

```
maListe = ["orange", "Pomme", "poire"]
maListe.sort()
print(maListe)
```

```
['Pomme', 'orange', 'poire']
```

On peut utiliser la clé de tri **`str.lower`** pour corriger le problème :

```
maListe = ["orange", "Pomme", "poire"]
maListe.sort(key = str.lower)
print(maListe)
```

```
['orange', 'poire', 'Pomme']
```

La fonction **`sorted(Liste)`** permet de créer une liste triée à partir d'une autre liste :

```
L1 = [73, 5, 59]
L2 = sorted(L1) #L1 n'est pas modifiée
print(L1,L2,sep = " * ")
```

```
[73, 5, 59] * [5, 59, 73]
```

INVERSION DE LISTE

La méthode **`reverse()`** permet d'inverser l'ordre des éléments.

```
maListe = ["pomme", "banane", "cerise"]
maListe.reverse()
print(maListe)
```

```
['cerise', 'banane', 'pomme']
```

COPIER UNE LISTE

Important : on ne doit pas copier une liste en tapant Liste1 = Liste2, car les modifications apportées à l'une des deux listes seront automatiquement apportées dans l'autre.

Il existe plusieurs façons de faire une copie.

La méthode **`copy()`** fait une copie superficielle.

```
L1 = ["pomme", "banane", "cerise"]
L2 = L1.copy()
print(L2)
```

```
['pomme', 'banane', 'cerise']
```

Le constructeur **`list()`** fait une copie superficielle.

```
L1 = ["pomme", "banane", "cerise"]
L2 = list(L1)
print(L2)
```

```
['pomme', 'banane', 'cerise']
```

La fonction **`deepcopy()`** du **module copy** est la seule à faire une copie profonde.

```
L1 = ["pomme", "banane", "cerise"]
L2 = deepcopy(L1)
print(L2)
```

```
['pomme', 'banane', 'cerise']
```

FUSIONNER DEUX LISTES

Il existe plusieurs façons de fusionner ou de concaténer deux ou plusieurs listes en Python.

Première façon : on utilise l'opérateur « **+** ».

```
L1 = ["a", "b", "c"]
L2 = [1, 2, 3]
L1 += L2
print(L1)
```

```
['a', 'b', 'c', 1, 2, 3]
```

Deuxième façon : on ajoute les éléments un par un.

```
L1 = ["a", "b", "c"]
L2 = [1, 2, 3]
for x in L2:
  L1.append(x)
print(L1)
```

```
['a', 'b', 'c', 1, 2, 3]
```

Troisième façon : on utilise la méthode **`extend()`**

```
L1 = ["a", "b", "c"]
L2 = [1, 2, 3]
L1.extend(L2)
print(L1)
```

```
['a', 'b', 'c', 1, 2, 3]
```

AJOUTER LES ELEMENTS D'UN AUTRE TYPE (TUPLE, SET, DICTIONNAIRE)

La méthode **`extend()`** permet d'ajouter les éléments d'un tuple à une liste :

```
maListe = ["pomme", "banane", "cerise"]
monTuple = ("orange", "citron") # ("orange", "citron") est un tuple
maListe.extend(monTuple) # ajoute à maListe les éléments de monTuple
print(maListe)
```

```
['pomme', 'banane', 'cerise', 'orange', 'citron']
```