



Quiz



Code : <https://wooclap.com>

KLXDIO

Les tableaux (type `list`)

Définition

En informatique, un tableau est une structure de données représentant une séquence finie d'éléments auxquels on peut accéder efficacement par leur position, ou indice, dans la séquence.

i Python implémente les tableaux dans le type `list`. De ce fait, il y a une confusion entre les listes, telle que définies en informatique et le type `list` de Python. Dans la suite de ce cours, on utilisera le mot liste et tableau indifféremment. Cependant en terminale, nous verrons la définition de liste qui est très différente de cette implémentation.

●Création d'un tableau (list)

En python, les tableaux sont contenus dans des crochets [...]
Les éléments sont séparés par des virgules ,

[élément1 , élément2 , élément3]

☛Exemples : Ainsi pour créer un tableau en python, on définit une variable en lui donnant une expression de type `list` vide ou non.

```
1 # Créer une liste vide
2 liste_1 = []
3
4 # Créer une liste à partir de valeurs
5 liste_3 = [0, "Ada Lovelace", True, 3.1416]
6
7 # Créer une liste contenant plusieurs fois le même élément ( 5 zéros dans ce
  ↳ cas).
8 liste_3 = [0]*5
```

●Accéder et modifier des éléments

Pour accéder à un élément, on utilise son index (sa position dans la liste)

```
1 ma_liste = [0, "Bob", True, 3.1416, 'new elmt']
2 print(ma_liste[1]) # "Bob"
3
4 # On peut modifier les éléments de la liste en affectant une nouvelle valeurs
5 ma_liste[1] = "Ada Lovelace"
6 print(ma_liste)
7 # [0, "Ada Lovelace", True, 3.1416, 'new elmt']
```

Exercice 1

On considère le script suivant :

```
1 tab = [2, 19, 34, 0, 8, 9, 2]
2
```

Que vaut `t[3]` ?

.....

Exercice 2

Après l'affectation suivante :

```
1 alphabet = [ 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L',
2             'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z' ]
```

Quelle instruction permet d'accéder à la lettre E ?

.....

Exercice 3

On considère le script suivant :

```
1 t = [2, 8, 9, 2]
2 t[2] = t[2] + 5
```

Quelle est la valeur de t à la fin de son exécution ?

.....

Exercice 4

Qu'affiche le programme python suivant ?

```
1 L=[3,2,2]
2 M=[1,5,1]
3 L.append(M)
4 print(L)
```

.....

●Parcourir un tableau

En python, les tableaux (list) possèdent les caractéristiques suivantes :

- un opérateur d'appartenance **in** ;
- fonction taille **len()** donnant le nombre d'éléments contenus dans la liste ;

On peut donc parcourir une liste de plusieurs façons. Nous allons en voir 3 :

1. Parcourir un tableau en utilisant les index.

```
1 for index in range(len(ma_liste)):
2     print(ma_liste[index])
```

L'index d'une liste commence à 0.

2. Parcourir un tableau en itérant les éléments de la liste.

```
1 for elmt in ma_liste:
2     print(elmt)
```

3. Parcourir un tableau en récupérant l'index et la valeur grâce à la fonction **enumerate**

```
1 for index, valeur in enumerate(ma_liste):
2     print(f"A l'index {index} de la liste, la valeur est {valeur}")
```

Exercice 5

On considère le code suivant :

```
1 def feed(t):  
2     for i in range(len(t)):  
3         t[i] = t[i] + i  
4     return t
```

Que renvoie `feed([12, 24, 32])` ?

.....

• Ajouter des éléments _____

i

Il existe plusieurs façons d'ajouter des éléments à une liste :

- En ajoutant les éléments un par un
- En étendant une liste à l'aide d'une autre liste
- En créant une nouvelle liste à partir de la concaténation de 2 listes.

•• Ajouter un élément _____

Pour ajouter un seul élément, on utilise la méthode **append**

```
tab . append ( élément )
```

Exemple

```
1 infos = ["Ada", "Alan", "Grace"]
2 infos.append('Guido')
3 print(infos)
4 # ["Ada", "Alan", "Grace" 'Guido']
```

•• Ajouter plusieurs éléments _____

Pour ajouter tous les éléments d'un tableau à un autre tableau, on utilise la méthode **extend**

```
tab . extend ( tab2 )
```

Exemple

```
1 infos = ["Ada", "Alan", "Grace"]
2 autres_infos = ["Ian", "Bill"]
3 infos.extend(autres_infos)
4 print(infos)
5 #["Ada", "Alan", "Grace", "Ian", "Bill"]
```

•• Concaténation _____

Il est aussi possible de concaténer 2 listes pour en créer une troisième.

```
1 >>> infos1 = ["Ada", "Alan"]
2 >>> infos2 = ["Grace", "Ian",]
3 >>> tous_les_infos = infos1 + infos2
4 >>> print(tous_les_infos)
5 #["Ada", "Alan", "Grace", "Ian",]
```

● Supprimer un élément

i Dans cette partie, nous allons détailler la méthode `pop`. Cependant il existe une autre méthode permettant de supprimer des éléments : la méthode `remove` (cf : voir la partie *autres méthodes*)

- Supprimer le dernier élément d'un tableau

```
tab .pop()
```

Pour supprimer le dernier élément d'un tableau, on utilise la méthode `pop()` sans paramètre.

Exemple :

```
1 tab = ["A", "B", "C", "D", "E", "F"]
2 tab.pop()
3 print(tab)
4 # affichage ["A", "B", "C", "D", "E"]
```

- Supprimer un élément à partir de son index

```
tab .pop(index)
```

Pour supprimer un élément à partir de son index, on utilise la méthode `pop(i)` en mettant l'index `index` de l'élément à supprimer en paramètre.

Exemple :

```
1 tab = ["A", "B", "C", "D", "E", "F"]
2 tab.pop(3)
3 print(tab)
4 # affichage ["A", "B", "C", "E", "F"]
```

● Les méthodes

Dans toute cette section, on considérera la liste `nombres`. Les instructions sont considérées être effectuées les une à la suite des autres.

```
1 nombres = [17, 38, 10, 25, 72]
```

1. Ordonner la liste

```
1 # La méthode sort permet de classer les éléments de la suite
2 nombres.sort()
3 print(nombres)
4 # Affiche [10, 17, 25, 38, 72]
```

2. Inverser les éléments d'une liste

```
1 nombres.reverse()
2 print(nombres)
3 # Affiche [72, 38, 25, 17, 10]
```

3. Supprimer des éléments

```
1 nombres.remove(38)
2 print(nombres)
3 # Affiche [72, 25, 17, 10]
4 # L'instruction pop enlève et renvoi le dernier élément de la liste
5 print(nombres.pop())
6 # Affiche 10
7 print(nombres)
8 # Affiche la liste [72, 25, 17 ]
```

4. Information sur un élément

```
1 # On peut récupérer la position d'un élément
2 print(nombres.index(17))
3 # Affiche 2
4 liste = [72, 25, 17, 25 ]
5 # compte le nombre d'occurrences d'un élément
6 print(liste.count(25))
7 # Affiche : 2
8
```

Les compréhensions de listes

•Syntaxe de base

Une liste par compréhension (ou liste en intension) est une expression qui permet de générer une liste de manière très compacte. Cette notation reprend la définition mathématique d'un ensemble en intension :

$$\{x^2 | x \in [2, 10]\} \Rightarrow [x**2 \text{ for } x \text{ in range}(2, 11)]$$

Une liste en compréhension L a la syntaxe minimale suivante :

```
1 L = [expr for elmt in t]
```

la paire de crochets, les mots-clefs **for** et **in** sont obligatoires.

- **t** est, assez souvent, un objet de **list** ou un **range**
- **elmt** est l'élément courant qui parcourt l'élément **t** ; **elmt** est appelé variable de contrôle de la liste en compréhension
- **expr** est une expression qui dépend en général de **elmt**, mais pas forcément, et dont la valeur est placée dans L

Par exemple :

On veut créer une liste (L) contenant les carrés des entiers stockés dans une liste (**entiers**)

```
1 entiers = [0,1,2,3,4,5]
2 L = [elmt*2 for elmt in entiers]
3 print(L)
4 # Affiche [0,1,4,9,16,25]
```

Description : On créer une liste L contenant le double (**elmt*2**) de tous les éléments se trouvant dans la listes **entiers** (**for elmt in entiers**).

Exercice 6

Que va contenir la variable L après execution de ce code

```
1 entiers = [5, 8, 12, 9]
2 L = [ elmt**2 for elmt in entiers]
3 print(L)
```

.....
.....

Exercice 7

Soit `lettres = ["A", "b", "C", "d"]`.