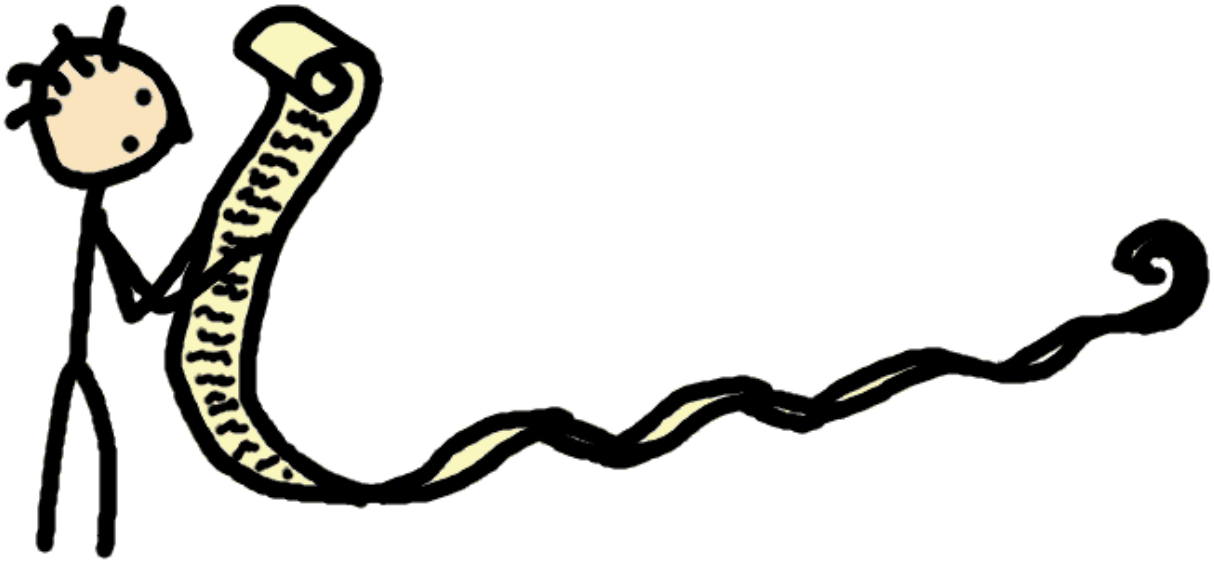

Quiz

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 implemente 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 implementation.

•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.

```
# Créer une liste vide
liste_1 = []

# Créer une liste à partir de valeurs
liste_3 = [0, "Ada Lovelace", True, 3.1416]

# Créer une liste contenant plusieurs fois le même élément ( 5 zéros dans ce cas).
liste_3 = [0]*5
```

•Ajouter des éléments

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

— Pour ajouter un seul élément, on utilise la méthode `append`

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

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

```
infos = ["Ada", "Alan", "Grace"]
autres_infos = ["Ian", "Bill"]
infos.extend(autres_infos)
print(infos)
#["Ada", "Alan", "Grace", "Ian", "Bill"]
```

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

```
>>> infos1 = ["Ada", "Alan"]
>>> infos2 = ["Grace", "Ian",]
>>> tous_les_infos = infos1 + infos2
>>> print(tous_les_infos)
#["Ada", "Alan", "Grace", "Ian",]
```

•Accéder et modifier des éléments

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

```
ma_liste = [0, "Bob", True, 3.1416, 'new elmt']
print(ma_liste[1]) # "Bob"

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

Exercice 1

On considère le script suivant :

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

```

Que vaut t[3] ?

.....

Exercice 2

Après l'affectation suivante :

```
alphabet = [ 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L',
 '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 :

```
t = [2, 8, 9, 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 ?

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

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

L'index d'une liste commence à 0.

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

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

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

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

Exercice 5

On considère le code suivant :

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

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

.....

Les méthodes

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

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

(a) Ordonner la liste

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

(b) Inverser les éléments d'une liste

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

(c) Supprimer des éléments

```
nombres.remove(38)
print(nombres)
# Affiche [72, 25, 17, 10]
# L'instruction pop enlève et renvoi le dernier élément de la liste
print(nombres.pop())
# Affiche 10
print(nombres)
# Affiche la liste [72, 25, 17 ]
```

(d) Information sur un élément

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

```