3 - Les listes (tableaux)

https://www.lumni.fr/video/notion-de-listes-en-informatique-et-application-aux-images-numeriques#containerType=serie&containerSlug=la-maison-lumni-lycee

Définition:

Un objet de type list, que nous appelons une liste, ressemble à un p-uplet : un ensemble ordonné d'éléments avec des indices pour les repérer. Les éléments d'une liste sont séparés par des virgules et entourés de crochets et peuvent être de n'importe quel type, organisés séquentiellement, c'est à dire, les uns à la suite des autres.

Dans beaucoup de langage, les listes sont appelée "tableaux", mais en Python, un tableau ce nomme liste.

Contrairement au tuple, une liste peut être modifié par affectation, on dit quelle est muable ou mutable.

Création d'une liste:

La création d'une liste est assez similaire à celle d'un tuple, à une exception près, il faut mettre des "[]" à la place des "()".

Ainsi, pour créer une liste, la syntaxe sera la suivante:

liste1 = [1, 3, 5, 7, 9]

Cette instruction va créer la liste "liste1", est insérer dedans les variables 1, 3, 5, 7 et 9.
Il est aussi possible de créer une liste vide:

liste2 = []

Il est possible de créer une liste composée de plusieurs listes:

liste3 = [[1, 2], [3, 4]]

Cette instruction va créer une liste "liste3", composée de la sous-liste [1, 2] et de la sous-liste [3, 4]

Il est aussi possible d'utiliser les fonctions list() et range():

liste4 = list(range(2, 10, 3))

Va créer une liste de type list, contenant les entiers de 2 à 10 par pas de 3, c'est à dire (2, 5, 8)

Indéxation des éléments d'une liste:

Comme pour les tuples, chaques élément qui compose la liste à un indice attribué en fonction de sa place dans la liste. Une liste commence à 0, puis 1, puis 2, etc...

Ainsi si l'on reprend notre liste1:

liste1 = [1, 3, 5, 7, 9]

1 est à l'indice 0, 3 à l'indice 1, 5 à l'indice 2, 7 à l'indice 3 et 9 à l'indice 4.

Nous avons bien 5 indices, pour 5 nombres dans notre liste.

Si l'on reprend notre liste3:

liste3 = [[1, 2], [3, 4]]

La sous-liste [1, 2] est à l'indice 0 et la sous-liste [3, 4] est à l'indice 1

La même chose s'applique pour nos sous-liste...

Pour accéder à un élément de notre liste, il faudra le faire via son indice, par exemple, dans notre liste1, si je souhaite accéder à l'élément 2:

liste1[2] -> 5

etc...

Cet "indice" est aussi appelé "index" de postion.

Modification de la liste par affectation:

Contrairement aux tuples, une liste est modifiable, via la syntaxe suivante:

liste1[1] = 6

Va remplacer le nombre 3 (indice 1 de notre liste), par le nombre 6. Elle devient donc:

liste1 -> [1, 6, 5, 7, 9]

Ajouter un élément dans une liste:

Il est possible d'ajouter un élément à la fin d'une liste via la syntaxe suivante:

liste1.append(11) -> liste1 -> (1, 6, 5, 7, 9, 11)

Création d'une liste par compréhension:

Il est possible, et certain trouve cela plus élégant, de créer une liste en compréhension, avec le langage Python, et cela s'avère très pratique pour transformer ou encore filtrer une liste.

Pour ce faire, on utilise une boucle "for" avec la syntaxe suivante:

for ... in range()

Il est aussi possible de créer un filtre grâce à une condition de filtrage.

Ainsi, si l'on prend l'exemple suivant:

liste6 = [i**2 for i in range(5) if i**2 % 2 == 0]

Cela peut parraitre un peu compliqué, mais si l'on décortique cette ligne, la liste6 va contenir les valeurs de i^2, pour i allant de 0 à 4 (range(5)), si le reste de la division de i par 2 vaut 0.
Ainsi, notre liste6 contiendra les valeurs [0, 4, 16].

Tout comme les tuples, les listes sont des itérables, ainsi que les chaines de caractères ou encore les dictionnaire. Un itérable est un objet dont les éléments sont accessible via une boucle "for".

Exemple:

```
table_de_3 = [3*i for i in range(1, 11)]
table_de_6 = [2*n for n in multiple_de_3]
```

On pourrait aussi imaginer disposer d'une fonction f et d'une liste d'abscisses:

```
images = [f(x) for x in abscisses]
```

Avec des listes emboîtées:

```
liste = [[[i, j] for i in range(3)] for j in range(2)]
print(liste)
[[[0, 0], [1, 0], [2, 0]], [[0, 1], [1, 1], [2, 1]]]
```

Ajouter un filtre (ou condition de filtrage), permet de ne selectionner que certains éléments de notre liste, et ceux, grâce a une expression booléenne. Si la condition est Vraie (True), les éléments seront ajouté à notre liste.

Copie:

Si l'on prend l'exemple suivant:

```
liste1 = ["a", "b", "c"]
liste2 = liste1
liste1[1] = "d"
print(liste2)
```

-> ['a', 'd', 'c']

Pourtant, on pourrait croire que liste2 contient ['a', 'b', 'c']

En fait, dans cet exemple, une même liste a 2 noms et liste2 n'est pas une copie de liste1.
Pour obtenir une copie, il faut créer une nouvelle liste, par exemple:

liste2 = list(liste1)

Il s'agit d'une copie superficielle, disons de niveau 1. Pour comprendre le fonctionnement, il faut considérer qu'une liste est juste une adresse (en mémoire) qui contient les adresses de ses éléments. Avec cette copie superficielle, une nouvelle liste est créée avec une nouvelle adresse. Les éléments gardent eux la même adresse.

Examinons un autre exemple avec liste = 2 * [["a", "b"]]. Cela revient à écrire liste1 = ["a", "b"], puis liste = 2 * [liste1]. Le résultat est [['a', 'b'], ['a', 'b']]. Les deux éléments ont ici la même adresse, celle de liste1.

Donc une modification d'un élément de liste1 concerne les deux éléments de liste. Que ce soit liste1 [1] = "c" ou liste[0][1] = "c", le résultat est le même, [['a', 'c'], ['a', 'c']].

```
liste = 2 * [["a", "b"]]
print(liste) -> [['a', 'b'], ['a', 'b']]
liste1 = ["a", "b"]
liste2 = 2 * [liste1]
print(liste2) -> [['a', 'b'], ['a', 'b']]
liste1[1] = "c"
print(liste2) -> [['a', 'c'], ['a', 'c']]
liste[0][1] = "c"
print(liste) -> [['a', 'c'], ['a', 'c']]
```

Par contre avec l'instruction liste = [["a", "b"] for i in range(2)], une première liste ["a", "b"] est créée quand i prend la valeur 0, puis une seconde liste quand i prend la valeur 1. Ces deux listes sont distinctes, (les adresses sont différentes), donc modifier l'une n'a aucune conséquence sur l'autre.

Considérons le cas d'une liste dont les éléments sont des listes. Une copie superficielle comme ci-dessus crée une nouvelle liste avec une nouvelle adresse et les éléments de cette copie ont eux la même adresse que les éléments de la liste initiale. Par exemple avec liste1 = [['a', 'b'], ['c', 'd']] puis liste2 = list(liste1), les adresses de liste1 et liste2 sont distinctes mais les adresses de liste1[0] et liste2[0] sont identiques. Donc la modification d'un élément de liste1 [0] se répercute sur liste2[0]. Pour obtenir une copie "en profondeur", passer au niveau 2, il faut donner de nouvelles adresses pour les listes éléments d'une liste

Des fonctions de copie sont disponibles à partir du module copy. En particulier, dans le cas d'une liste de listes, pour effectuer une copie en profondeur, nous disposons de la fonction deepcopy.

```
from copy import deepcopy

liste1 = [["a", "b"], ["c", "d"]]
liste2 = deepcopy(liste1)
liste2[1][0] = "e"
print(f"liste2 => {liste2}") -> liste2 => [['a', 'b'], ['e', 'd']]
print(f"liste1 => {liste1}") -> liste1 => [['a', 'b'], ['c', 'd']]
```

Applications:

Calcul d'une moyenne:

```
# Calcul d'une moyenne
def moyenne(liste):
        s = 0
```

```
        for u in liste:
            s += u
        return s/len(liste)
```

- Créez une liste de notes
- Testez si la moyenne renvoyé est juste

Représentation graphique d'une fonction:

```
# Représentation graphique d'une fonction
import matplotlib.pyplot as plt


def f(x):
   return x ** 2 + x - 4

liste_x = [0.1 * n for n in range(-30, 31)]
liste_y = [f(x) for x in liste_x]
plt.plot(liste_x, liste_y)
plt.show()
```

- Testez le code précédent

Exos:

Se renseigner sur les différentes opérations et méthodes disponible pour les listes et créer un tableau (excel ou word) de la forme:

type opération / méthode | exemple concret | résultat exemple

créer une liste | liste_vide = [] | print(liste_vide) -> []
créer une liste | liste_int = [1, 2] | print(liste_int) -> [1, 2]
etc...
méthode .append() | liste_int.append(3) | print(list_int) -> [1, 2, 3]
etc...

Est-il possible de créer la liste suivante:

- L = [1, 2.8, '3', True, ("f", "a", "c", "e")]

Soit la liste suivante généré en compréhension, que contient-elle ?

- L1 = [i**2 for i in range(5)]

Il est également possible d'utiliser la même fonction pour tous les éléments d'une liste, par exemple la fonction sqrt() du module math. Soit l'instruction suivante, combien de terme la liste L2 contient ?

- L2 = [math.sqrt(element) for element in L1]

Ecrire les 2 listes précédentes et expliquer la différence entre les listes L1 et L2

Ecrire l'instruction permettant de créer la liste (L3) de listes (L1 et L2)

On souhaite stocker dans un tableau de tableaux, les déplacements d'un cavalier sur l'échiquier. Ecrire le tableau en fonction des déplacements suivant:

Le cavalier part de B1, puis vas en C3, puis en B5, puis en D6, puis en E4, puis en F6

Il est possible d'écrire une liste en compréhension de la même manière qu'a l'exercice précédent, en utilisant une double boucle for. Ecrire la liste généré par l'instruction suivante:

liste = [[i * j for i in range(5)] for j in range(3)]

Combien de lignes et de colonnes possède la liste ?