PYTHON – Listes

Utilité des Listes

Imaginons que dans un programme, nous ayons besoin simultanément de 4 notes pour calculer une moyenne.

La seule solution dont nous disposons à l'heure actuelle consiste à déclarer 4 variables, appelées par exemple N1, N2, N3 et N4.

Arrivé au calcul, et après une succession de 4 instructions « input » distinctes, le calcul de la moyenne donnera obligatoirement :

```
N1 = int(input ("saisir note 1"))
N2 = int(input ("saisir note 2"))
N3 = int(input ("saisir note 3"))
N4 = int(input ("saisir note 4"))
moyenne = (N1+N2+N3+N4)/4
```

Heureusement, la programmation nous permet de **rassembler toutes ces variables en une seule**, au sein de laquelle chaque valeur sera désignée par un numéro.

- Un ensemble de valeurs portant le même nom de variable et repérées par un nombre, s'appelle une **liste**.
- Le nombre qui, au sein d'une liste, sert à repérer chaque valeur s'appelle **l'indice**.
- Chaque fois que l'on doit désigner un élément du tableau, on fait figurer le nom du tableau, suivi de l'indice de l'élément.

Création de listes

On utilise des crochets pour créer une liste :

```
ma_liste = [] # On crée une liste vide
```

On peut également créer une liste non vide, en lui indiquant directement à la création les objets qu'elle doit contenir, séparés par des virgules.

```
ma_liste = [element1, element2, elementN]
```

Exemple :

```
ma_liste = [1, 2, 3, 4, 5] # Une liste avec cinq objets de type int
print(ma_liste)
```

Résultat dans la console :

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

La liste que nous venons de créer compte cinq objets de type int

Exemple 2 :

```
ma_liste = [1, 3.5, "une chaine", []]
```

Nous avons créé ici une liste contenant quatre objets de types différents : un entier, un flottant, une chaîne de caractères et… une autre liste.

- Vous pouvez faire des listes de toute longueur.
- Les listes peuvent contenir n'importe quel type d'objet. On peut avoir une liste contenant plusieurs nombres entiers (1, 2, 50, 2000 ou plus, peu importe), une liste contenant des flottants, une liste contenant des chaînes de caractères... et une liste mélangeant ces objets de différents types
- Les objets dans une liste peuvent être mis dans un ordre quelconque.

Accéder aux éléments d'une liste de listes

Dans une liste, chaque élément a une place précise, donc l'ordre des éléments compte.

Pour accéder aux éléments d'une liste, nous allons utiliser les **indices** : les indices des listes **commencent à 0, et non à 1**, autrement dit, le plus petit indice est zéro.

On accède aux éléments d'une liste en indiquant entre **crochets** l'indice de l'élément qui nous intéresse.

```
NomListe[indice]
```

Exemple d'accès aux éléments d'une liste :

```
ma_liste = [7, -3, 'm']
print(ma_liste[0]) # On accède au 1er élément de la liste
print(ma_liste[2]) # Troisième élément
print(ma_liste)
```

indice	**0**	**1**	**2**
valeur	7	-3	'm'

Résultat dans la console :

```
7
'm'
[7, -3, 'm']
>>>
```

Modification de listes

Les listes permettent de **remplacer directement un élément par un autre**.

Les listes sont en effet des types dits **mutables**.

```
NomListe[indice]=nouvelleValeur
```

Exemple de modification d'un élément d'une liste :

```
ma_liste = [7, -3, 'm']
ma_liste[1] = 'Z' # On remplace -3 par 'Z'
print(ma_liste)
```

Résultat dans la console :

```
[7, 'Z', 'm']
>>>
```

Insérer des objets dans une liste

On dispose de plusieurs méthodes, définies dans la classe list, pour ajouter des éléments dans une liste.

Ajouter un élément à la fin de la liste

On utilise la méthode append pour ajouter un élément à la fin d'une liste.

```
ma_liste = [1, 2, 3]
ma_liste.append(56) # On ajoute 56 à la fin de la liste
print(ma_liste)
```

On passe en paramètre de la méthode append l'objet que l'on souhaite ajouter à la fin de la liste.

Résultat dans la console :

```
[1, 2, 3, 56]
>>>
```

Insérer un élément dans la liste

On peut insérer un objet dans une liste, à l'endroit voulu.

On utilise pour cela la **méthode insert**.

```
ma_liste = ['a', 'b', 'd', 'e']
ma_liste.insert(2, 'c') # On insère 'c' à l'indice 2
print(ma_liste)
```

Quand on demande d'insérer c à l'indice 2, la méthode va décaler les objets d'indice supérieur ou égal à 2. C va donc s'intercaler entre b et d.

Résultat dans la console :

```
['a', 'b', 'c', 'd', 'e']
>>>
```

Concaténation de listes

On peut agrandir des listes en les concaténant avec d'autres. On utilise la **méthode extend** ou l'opérateur +

```
ma_liste1 = [3, 4, 5]
ma_liste2 = [8, 9, 10]
ma_liste1.extend(ma_liste2) # On insère ma_liste2 à la fin de ma_liste1
print(ma_liste1)
```

Résultat dans la console :

```
[3, 4, 5, 8, 9, 10]
>>>
```

```
ma_liste1 = [3, 4, 5]
ma_liste2 = [8, 9, 10]
print(ma_liste1 + ma_liste2) # l'opérateur + concatène 2 listes entre elles
```

Résultat dans la console :

```
[3, 4, 5, 8, 9, 10]
>>>
```

```
ma_liste1 = [3, 4, 5]
ma_liste2 = [8, 9, 10]
ma_liste1 += ma_liste2 # On utilise += pour étendre une liste
print(ma_liste1)
```

Résultat dans la console :

```
[3, 4, 5, 8, 9, 10]
>>>
```

Suppression d'éléments d'une liste

Le mot-clé del

On peut également utiliser del pour supprimer des éléments d'une séquence, comme une liste.

```
ma_liste = [-5, -2, 1, 4, 7, 10]
del ma_liste[0] # On supprime le premier élément de la liste
print(ma_liste)
```

Résultat dans la console :

```
[-2, 1, 4, 7, 10]
>>>
```

```
ma_liste = [-5, -2, 1, 4, 7, 10]
del ma_liste[2] # On supprime le troisième élément de la liste
print(ma_liste)
```

Résultat dans la console :

```
[-2, 1, 7, 10]
>>>
```

Notez que le mot-clé del n'est pas une méthode de liste. Il s'agit d'une fonctionnalité de Python qu'on retrouve dans la plupart des objets.

La méthode remove

On peut aussi supprimer des éléments de la liste grâce à la méthode remove qui prend en paramètre la valeur de l'élément lui-même.

```
ma_liste = [31, 32, 33, 34, 35]
ma_liste.remove(32)
print(ma_liste)
```

Résultat dans la console :

```
[31, 33, 34, 35]
>>>
```

La méthode remove parcourt la liste et en retire l'élément que vous lui passez en paramètre.

La méthode remove ne retire que la première occurrence de la valeur trouvée dans la liste

Autres méthodes

Vous pouvez en voir la liste exhaustive des méthodes dans l'aide, en tapant, dans l'interpréteur : help(list)

len(list)

Renvoie le nombre d'éléments dans la liste.

```
fruits = ['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
print(len(fruits))
```

```
7
```

list.clear()

Supprime tous les éléments de la liste.

list.count(*x*)

Renvoie le nombre d'éléments ayant la valeur *x* dans la liste.

```
fruits = ['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
print(fruits.count('apple'))
```

```
2
```

list.reverse()

Inverse l'ordre des éléments dans la liste.

```
fruits = ['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
fruits.reverse()
print(fruits)
```

```
['banana', 'apple', 'kiwi', 'banana', 'pear', 'apple', 'orange']
```

list.sort()

Ordonne les éléments dans la liste.

```
fruits = ['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
fruits.sort()
print(fruits)
```

['apple', 'apple', 'banana', 'banana', 'kiwi', 'orange', 'pear']

max(liste)

Renvoie l'élément avec la plus grande valeur

min(liste)

Renvoie l'élément avec la plus petite valeur

Vérifier si un élément existe

Pour déterminer si un élément spécifié est présent dans une liste, utilisez le mot-clé *in* :

```
thisliste = ["apple", "banana", "cherry"]
if "apple" in thisliste :
    print("Oui, 'apple' est dans la liste")
```

Oui, 'apple' est dans la liste

Parcourir les listes

L'énorme avantage des listes, c'est qu'on va pouvoir les traiter en faisant des boucles.

Boucle while

on utilise la méthode **len()** pour connaitre la longueur de la liste

```
ma_liste = ['a', 'b', 'c','d', 'e', 'f']
i = 0 # Notre indice pour la boucle while
while i < len(ma_liste):
    print(ma_liste[i])
    i += 1 # On incrémente i, ne pas oublier !
```

Résultat dans la console :

```
a
b
c
d
>>>
```

Boucle for

On utilise la structure **for xxx in…**

xxx est une variable créée par le for, ce n'est pas à vous de l'instancier.

xxx prend successivement chacune des valeurs figurant dans la liste

```
ma_liste = ['a', 'b', 'c','d', 'e', 'f']
for element in ma_liste:
    print(element)
```

Résultat dans la console :

```
a
b
c
d
>>>
```

la méthode for se contente de parcourir la liste en capturant les éléments dans une variable, sans qu'on puisse savoir où ils sont dans la liste

Exemple de code pour effectuer un calcul de moyenne sur 4 notes:

```
#Programme moyenne.py
note = []
somme, moyenne = 0,0
for i in range (0,4):
        print("saisir la note n°",i+1)
        saisie = int(input())
        note.append(saisie)
for i in range (0,4):
    somme = somme + note[i]
moyenne = somme / 4
print("la moyenne des notes est",moyenne)
```

NB : On a fait deux boucles successives pour plus de lisibilité, mais on aurait pu n'en écrire qu'une seule dans laquelle on aurait tout fait d'un seul coup.

Résultat dans la console :

```
saisir la note n° 1
12
saisir la note n° 2
7
saisir la note n° 3
15
saisir la note n° 4
9
la moyenne des notes est 10.75
>>>
```