FICHE N°2 bis : LES LISTES

A) Accès aux éléments d'une liste

Si on veut stocker les différentes valeurs des termes d'une suite, on a besoin de pouvoir disposer d'un type de variable permettant de stocker, dans une seule variables, plusieurs valeurs. On dispose pour cela du type *list* en Python.

Définition : Une liste est une collection d'objets de même type que l'on groupe dans une seule variable afin d'éviter d'avoir plusieurs variables. Les objets sont séparés par des virgules et l'ensemble est enfermé dans des crochets.

Remarque : Le fait que les éléments d'une liste soient de même type n'est pas indispensable en Python. Cependant, cela est conseillé pour éviter les erreurs et les bugs. On pourra cependant mettre dans une même liste des nombres entiers et des nombres décimaux bien qu'ils ne soient pas de même type.

Exemple n°1 : Accéder aux éléments d'une liste.
Voici comment déclarer une liste en Python.

```
uneListe = [9, 3, 5.2, 4]
```

La liste `uneListe` contient 4 objets.
Pour accéder au premier objet, on écrit `uneListe[0]`.
Pour accéder au deuxième objet, on écrit `uneListe[1]`.

Attention, le numéro des positions des éléments dans la liste commence à 0. Ainsi, si on considère la liste `uneliste = ["a", "b", "c"]` alors :

- `uneliste[1]` renvoie `"b"` ;
- `uneliste[3]` renvoie une erreur, car il n'y a pas d'élément en position 3.

Remarques :

- Pour accéder au dernier objet, on peut écrire `uneListe[-1]`.
- On peut tester l'appartenance d'un élément à une liste à l'aide de l'instruction `in`. Ainsi l'instruction `3 in [2,3,4,5]` renverra `True` et l'instruction `3 in [7,8,9,10]` renverra `False`.

Remarque : Python permet d'additionner des listes. L'addition de deux listes revient à les concaténer. Par exemple : `[1,2,3]+[4,5,6]` renvoie `[1,2,3,4,5,6]`.

B) Différentes manières de définir une liste

Définitions :
Une liste peut être définie en Python de trois manières :

- **en extension**, c'est-à-dire en donnant la liste de tous les éléments ;
- **par ajouts successifs**, c'est-à-dire en commençant avec une liste vide et en lui ajoutant des éléments successivement (par exemple dans une boucle) ;
- **en compréhension**, c'est-à-dire en définissant les éléments de la liste par une formule.

Exemple n°2 : La liste donnant toutes les voyelles de l'alphabet, dans l'ordre alphabétique, peut être définie en extension par :

```
voyelles = ["a", "e", "i", "o", "u", "y"]
```

Exemple n°3 : Soit (u_n) la suite définie par son premier terme $u_0 = 3$ et, pour tout entier naturel n : $u_{n+1} = u_n^2 - u_n$.
Le script suivant permet d'obtenir une variable `maliste` contenant les cinq premiers termes de la suite (u_n).

```
u = 3
maliste = []
for k in range(5):
    maliste.append(u)
    u = u**2 - u
```

L'instruction `maliste.append(u)` ajoute la valeur de la variable `u` à la fin de la liste. Ainsi la variable `maliste` prend successivement les valeurs (à chaque passage de la boucle) :

- []
- [3]
- [3, 6]
- [3, 6, 30]
- [3, 6, 30, 870]
- [3, 6, 30, 870, 756030]

À la fin, la variable `maliste` est `[3, 6, 30, 870, 756030]`.

Exemple n°4 : Soit f la fonction définie sur $\mathbb{R}$ par $f(x) = 2x^2$. La commande suivante permet d'obtenir une liste des images par f de toutes les valeurs de $x \in [0;4]$ avec un pas de 1.

```
maliste = [2*k**2 for k in range(5)]
```

Cette instruction est équivalente à :

```
maliste = []
for k in range(5):
  maliste.append(2*k**2)
```

La liste obtenue est : `[0, 2, 8, 18, 32]`.

À retenir : Il existe plusieurs manières de définir une liste.

- **En extension** : On donne toute la liste dès sa création.
- **Par ajouts successifs** : On part d'une liste vide à laquelle on ajoute des éléments les uns après les autres.
- **En compréhension** : On définit les éléments de la liste comme les images par une fonction d'éléments d'un intervalle.

C) D'autres actions sur les listes

Définitions : Certaines instructions sur les listes sont possibles en Python :

- **len** : Renvoie le nombre d'éléments d'une liste.
- **min** : Renvoie le plus petit élément d'une liste.
- **max** : Renvoie le plus grand élément d'une liste.
- **index :** Renvoie la première position dans la liste d'une valeur donnée.

Exemple n°5 : On considère la liste `maliste = [3, 2, 5, 7, 5, 4]`.

- `len(maliste)` renvoie 6.
- `min(maliste)` renvoie 2.
- `max(maliste)` renvoie 7.
- `maliste.index(5)` renvoie 2 (on rappelle que l'indice de la première position est 0).

Définitions : Certaines instructions sont également possibles sur les listes avec Python :

- **remove** : Enlève un élément de la liste.
- **sort** : Trie les éléments d'une liste par ordre croissant.
- **reverse** : Inverse l'ordre des termes d'une liste.

Complément : Il existe de nombreuses autres instructions sur les listes. Les plus intéressantes sont :

- **pop :** Enlève un élément d'une liste et le renvoie.
- **mean** (nécessite la librairie `numpy`) **:** Renvoie la moyenne des éléments d'une liste contenant des nombres.

Exemple n°6 :

```
from numpy import*
A = [1,2,3,4]
print(mean(A))
premier = A.pop(1)
print(A)
```

Ce programme affiche `2,5` (la moyenne des éléments de la liste A).
Il affiche ensuite `[1, 3, 4]` car l'instruction `pop(1)` a enlevé l'élément en position 1 de la liste (et l'a stocké dans la variable premier).

D) Liste de listes (maths expertes)

Définition : Les listes permettent de stocker, dans une même variable, plusieurs données. En particulier, une liste peut contenir des listes.
On peut ainsi représenter une matrice comme une liste contenant des listes, chacune de ces listes représentant une ligne de la matrice.

Exemple n°7 : La matrice $A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$ peut se modéliser par la liste de listes `A = [[1,2,3],[4,5,6],[7,8,9]]`.
Cette instruction permet de définir une liste de listes. On peut alors remarquer que :

- `A[1]` est l'élément d'indice 1 de `A` c'est-à-dire la liste `[4, 5, 6]` ;
- `A[1][0]` est l'élément d'indice 0 de la liste `A[1]` c'est-à-dire 4.

Exemple n°8 : Le programme suivant permet de générer par ajout successifs la matrice $A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$.

```
A = []
for i in range(2):
    ligne = []
    for j in range(3):
        ligne.append(3*i + j + 1)
    A.append(ligne)
```

À retenir : Une matrice peut être modélisée par une liste de listes.

E) Exercices

✓ **Exercice 1 :**

On considère les listes définies par les instructions suivantes :

- `maliste1 = [12,15,16,22];`
- `maliste2 = [[1,11],[2,22],[33,333]].`

1. Que renvoie chacune des instructions suivantes ?
   a. `maliste1[2]`
   b. `maliste1[4]`
   c. `maliste2[1]`
   d. `maliste2[2][1]`
2. Comment les listes sont-elles modifiées par les instructions suivantes ? On considérera que les instructions sont indépendantes les unes des autres et que la liste est remise à son état initial entre chaque question.
   a. `maliste1.append(11)`
   b. `maliste1.remove(16)`
   c. `maliste1.sort()`
   d. `maliste2[0].append(111)`

✓ **Exercice 2 :**

Proposer une fonction `mult` prenant en paramètres une liste de nombres `liste1` ainsi qu'un nombre réel λ et renvoyant une autre liste contenant les éléments de `liste1` multipliés par λ.

Exemple : `mult([1, 2, 3], 4)` renvoie `[4, 8, 12]`.

✓ **Exercice 3 :**

Que renvoie la fonction suivante ?

```
def mystery():
  A = [k-7 for k in range(8, 11)]
  A.append(0)
  A.append(22)
  A.reverse()
  A = A + [3, 5, 11, 9]
  A.sort()
  return A
```

✓ **Exercice 4 :**

On suppose que l'on dispose d'une fonction `test_premier` qui prend en paramètre un nombre entier strictement positif et qui renvoie True si ce nombre est premier et False sinon.

1. Proposer une instruction permettant de définir la liste des nombres premiers inférieurs à 20 en extension.
2. Proposer une instruction permettant de définir la liste des nombres premiers inférieurs à 20 par ajouts successifs.
3. Proposer une instruction permettant de définir la liste des nombres premiers inférieurs à 20 en compréhension (difficile).

✓ **Exercice 5 :**

Compléter la fonction `tri` suivante qui prend en paramètres deux listes de nombres et renvoie la liste obtenue en rangeant dans l'ordre décroissant la concaténation des deux listes.

```
def tri(A,B):
      out = … # Concaténation des deux listes
      out = … # Tri de la liste
      out = …
      return out
```