6.1 *Définition*

Une liste est une suite ordonnée d'objets, qui peuvent être de types différents, qui est délimitée par des crochets et dont les éléments sont séparés par des virgules.

Par exemple, `[3.14, 'pi', 3]` est une liste contenant le flottant `3.14`, la chaîne de caractères `'pi'` et l'entier `3`.

La liste vide, qui ne contient aucun élément, se définit par `[]`.

Une liste possède un type propre en Python : le type *list* (`list`). Il s'agit d'un objet mutable ce qui signifie qu'on peut modifier ou supprimer des éléments d'une liste.

Les éléments d'une liste peuvent eux-même être des listes : on parle dans ce cas de liste de listes. Par exemple, `[[1,2,-1],[3,2,1],[6,3]]` est une liste de listes d'entiers.

Il existe différentes façons de définir une liste en Python :

- **par énumération**, comme nous l'avons fait précédemment, en écrivant explicitement les éléments de la liste : `L = [2,4,'a',2.4,3]`;
- **en compréhension**, c'est-à-dire à l'aide d'une instruction conditionnelle (if) ou d'une boucle (for ou while) permettant de décrire tous les éléments de la liste dans l'ordre.
  Par exemple, la liste des entiers pairs de 0 à 10 peut s'obtenir de la manière suivante :

  ```
  L = [2*k for k in range(6)].
  ```

- par répétition si on souhaite créer une liste dont tous les éléments sont identiques. Par exemple, `[1]*10` crée une liste contenant 10 éléments, tous égaux à 1.

6.2 *Longueur et élément d'une liste*

Comme pour les chaînes de caractères, la longueur d'une liste est son nombre d'éléments et on l'obtient à l'aide de la fonction `len`.

La notion d'indice valable et la gestion des éléments d'une liste sont identiques à celles des chaînes.

Ainsi, si `L = [2, "abc", -1.1, [3,1]]` alors `len(L)` renvoie `4`, `L[1]` renvoie la chaîne de caractères `"abc"` et `L[-1]` renvoie la liste `[3,1]`. Dès lors, `L[1][2]` renvoie le caractère `"c"` et `L[-1][0]` renvoie l'entier `3`.

6.3 *Opérations sur les listes*

La syntaxe et le fonctionnement de l'extraction de sous-liste, de la concaténation, de la répétition et du test d'appartenance sont les mêmes pour les listes que pour les chaînes de caractères. Attention, cependant, le test d'appartenance ne fonctionne que pour des éléments et pas pour des sous-listes. Ainsi, `[1,2] in [1,2,3]` ou `[2] in [1,2,3]` renvoie `False` alors que `2 in [1,2,3]` renvoie `True`.

Le type list étant mutable, on dispose, de plus, d'opérations qui modifient la liste initiale :

1) **modification d'un élément** : on peut remplacer l'élément d'indice i d'une liste `L` par la valeur `v` grâce à la syntaxe `L[i] = v`. Par exemple, si `L = [3, "c", 1.1]` alors la syntaxe `L[1] = -2` modifie la liste `L` qui devient `[3, -2, 1.1]`.
2) **ajout d'un élément à une liste** : on peut modifier une liste `L` en lui ajoutant un élément `e` en utilisant
   - `L.append(e)` qui ajoute l'élément `e` à la fin de la liste `L`. Par exemple, si `L = [2,4,6]` alors `L.append(8)` ajoute la valeur `8` à la liste `L` et ainsi la nouvelle valeur de `L` est `[2,4,6,8]`.
     Une autre syntaxe possible est `L = L + e` ou `L += e` mais il est préférable d'utiliser la méthode `append` car l'opération `+` en Python désigne déjà la concaténation des listes et cela peut être source d'erreur si l'élément `e` est lui-même une liste.
   - `L.insert(i,e)` qui ajoute l'élément `e` à l'indice i dans la liste, ce qui a pour effet de décaler les éléments suivants. Par exemple, si `L = [2,4,6]` alors `L.insert(2,8)` ajoute la valeur `8` à la liste `L` de telle sorte que `8` devienne l'élément d'indice 2 de `L`. Ainsi, la nouvelle valeur de `L` est `[2,4,8,6]`.

3) **ajout simultané d'éléments à la fin d'une liste** : si `L1` et `L2` sont deux listes, on peut modifier `L1` en lui ajoutant les éléments de `L2` à la fin grâce à la méthode `.extend` : `L1.extend(L2)`.
Ainsi, si `L1 = [2,4,6]` et `L2 = [1,3,5]` alors `L1.extend(L2)` modifie la liste `L1` qui devient `[2,4,6,1,3,5]`.

4) **suppression d'un élément d'une liste** : on peut supprimer un élément dans une liste L en utilisant :
- `L.pop()` pour supprimer le dernier élément de la liste `L`.
- `L.pop(i)` pour supprimer l'élément d'indice *i* de la liste `L`.

Par exemple, si `L = [2,4,6]` alors `L.pop()` supprime la valeur 6 de la liste `L` et ainsi la nouvelle valeur de `L` est `[2,4]` et `L.pop(1)` supprime la valeur 4 de la liste `L` et ainsi la nouvelle valeur de `L` est `[2,6]`.

6.4 *Itération sur les éléments d'une liste*

Comme pour les chaînes de caractères, il est possible d'itérer une action sur les éléments d'une liste à l'aide d'une instruction conditionnelle et/ou d'une boucle. Par exemple, si on souhaite écrire une fonction `somme_liste` qui prend en argument une liste `L` dont les éléments sont des nombres et qui renvoie la somme des éléments de `L`, plutôt que d'écrire

```
def somme_liste (L):
    n = len(L)
    S = 0
    for k in range(n):
        S += L[k]
    return S
```

il est préférable d'écrire :

```
def somme_liste (L):
    S = 0
    for e in L:
        S += e
    return S
```

6.5 *Exercices*

♦ **PYT.1** On considère la liste suivante :

`L = [1,2.3,4.1,4,2.55,5,23].`

Les instructions suivantes sont-elles correctes et, si oui, que renvoient-elles ?

1) `L[1]`
2) `L[-2]`
3) `L[7]`
4) `len(L)`
5) `L[1.5]`
6) `L[1,5]`
7) `L[1:5]`
8) `L[0:4:2]`
9) `L[0]+L[3]`
10) `[L[0]]+[L[3]]`
11) `3*L[0]`
12) `3*[L[0]]`
13) `L[2] == 2.3`
14) `L[-3] == 2.55`
15) `4 in L`
16) `[4.1,4] in L`

♦ **PYT.2** On considère la liste suivante :

`M = [[1,-2], [0.5,6], [-1,3.45]].`

Les instructions suivantes sont-elles correctes et, si oui, que renvoient-elles ?

1) `M[0]`
2) `M[0][1]`
3) `M[0][1][2]`
4) `len(M)`
5) `len(M[0])`
6) `M[0]+M[2]`

♦ **PYT.3** On souhaite écrire une fonction affiche qui affiche en colonne les entiers de 0 à 3. Parmi les quatre fonctions suivantes, laquelle ou lesquelles convien(nen)t ?

```
def affiche1():
    for i in [0, 3]:
        print(i)
```

```
def affiche2():
    for j in [0, 1, 2, 3]:
        print(j)
```

```
def affiche3():
    for k in range (3) :
        print(k)
```

```
def affiche4():
    for m in range (4) :
        print(m)
```

♦ PYT.4

1) Compléter les programmes suivants pour qu'ils affichent la liste L de tous les entiers de 1 à 100.

```
# Programme 1
L = []
for k in range (...) :
    L += ...
print(L)
```

```
# Programme 2
L = []
for k in range (...) :
    L. append (...)
print(L)
```

```
# Programme 3
L = []
L = 100*[0]
for k in range (...) :
    L[k] = ...
print(L)
```

2) On considère la liste `L` de la question précédente. Écrire un programme qui multiplie par 2 tous les éléments de la liste `L` et affiche la nouvelle liste obtenue.

3) Écrire une fonction carre prenant en argument un entier $n > 0$ et qui renvoie la liste des carrés des entiers de 1 à n. Par exemple, `carre(5)` doit renvoyer `[1, 4, 9, 16, 25]`.

4) En utilisant la définition en compréhension, écrire un programme qui affiche une liste dont les éléments sont les doubles de tous les entiers de 1 à 100 puis une fonction `carre_compr` prenant en argument un entier $n > 0$ et qui renvoie la liste des carrés des entiers de 1 à n.

♦ PYT.5

Pour chacune des listes suivantes, donner une syntaxe Python permettant de la définir en compréhension.

1) `[1,3,5,7,9,11,13,15,17,19]`

2) `[0,1,4,9,16,25,36,49,64,81,100]`

3) `[1,2,4,8,16,32,64,128,256,512,1024]`

4) `[3,6,12,15,21,24,30,33,39,42,48,51]`

♦ PYT.6

Écrire une fonction `indices` qui prend en argument une chaîne de caractères `ch` et un caractère `c` et qui renvoie la liste, éventuellement vide, des indices de la chaîne `ch` où apparaît le caractère `c`. Ainsi, `indices('abbacccaa','a')` renvoie `[0,3,7,8]` et `indices('abbacccaa','d')` renvoie [].

♦ PYT.7

Écrire une fonction `sous_liste` prenant en argument deux listes `L1` et `L2` et qui renvoie `True` si `L2` est une sous-liste de `L1` (c'est-à-dire si tous les éléments de `L2` apparaissent consécutivement et dans le même ordre dans `L1`) et qui renvoie `False` sinon.

♦ PYT.8

Écrire une fonction `tous_positifs` qui prend en argument une liste non vide de nombres `L` et qui renvoie `True` si tous les nombres de la liste sont positifs ou nuls et `False` sinon.

♦ PYT.9

1) Écrire une fonction positifs prenant en argument une liste non vide d'entiers `L` et qui renvoie le nombre d'éléments de `L` qui sont positifs ou nuls.

Par exemple, `positifs([2, -1, 3, 4, -6, 0, 6])` doit renvoyer 5.

2) Écrire une fonction `rangs_negatifs` prenant en argument une liste non vide d'entiers `L` et qui renvoie la liste des indices des éléments strictement négatifs de `L`.

Par exemple, `rangs_negatifs([2, -1, 3, 4, -6, 0, 6])` doit renvoyer `[1, 4]`.

3) Écrire une fonction `motif` prenant en argument une liste non vide d'entiers `L` et qui renvoie `True` si `[0, 0]` est une sous-liste de `L` et `False` sinon.

4) Écrire une fonction `dix_vingt` prenant en argument une liste non vide d'entiers `L` et qui renvoie `True` si tous les éléments de `L` sont compris entre 10 et 20 (avec égalité possible) et `False` sinon.

♦ PYT.10

1) On veut écrire une fonction `croissante` prenant en argument une liste de réels `L` et qui renvoie `True` si

chaque élément de la liste est inférieur ou égal au suivant et `False` sinon. Recopier, en la corrigeant, la fonction proposée ci-dessous qui comporte des erreurs.

```
def croissante (L):
    n = len(L)
    for k in range (0, n):
        if L[k] > L[k +1]:
            return False
        else:
            return True
```

2) Écrire une fonction `monotone` prenant en argument une liste de réels `L` et qui renvoie `True` si la suite des termes de `L` est monotone (c'est-à-dire croissante ou décroissante) et False sinon.

♦ **PYT.11** Expliquer le rôle de la fonction `nb2` suivante :

```
from random import *

def nb2(n):
    c=0
    L=[randint(1,6) for k in range(n)]
    for j in L:
        if (j==2):
            c=c+1
    return c
```

♦ **PYT.12** Écrire une fonction suite qui prend en argument un entier naturel n et qui renvoie la liste des n premières valeurs de la suite (u_n) définie par $u_0 = 1$ et, pour tout $n \in \mathbf{N}$, $u_{n+1} = u_n^2 + n$.

♦ **PYT.13** Écrire une fonction `valeur_absolue_liste` qui prend en argument une liste d'entiers `L` et remplace chaque élément de la liste par sa valeur absolue.

♦ **PYT.14** Écrire une fonction `E_et_sigma` qui prend en arguments deux listes : la liste des valeurs d'une variable aléatoire X et la liste des probabilités associées (dans le même ordre, évidemment) et qui renvoie l'espérance et l'écart-type de X.