1 Réviser

— Une liste peut contenir un nombre quelconque d'éléments et ceux-ci peuvent être de type différents.
— Le nombre d'éléments d'une liste `L` est `len(L)`.
— Les éléments d'une liste `L` non vide sont numérotées à partir de `0` jusque `len(L)-1` si on lit de la gauche vers la droite ; si on lit de la droite vers la gauche on numérote de `-1` à `-L`.
Par exemple si `L = ["chat", 34, "haha", 1.23]`, `L[0]` vaut `"chat"` et `L[-2]` vaut `"haha"`.
— La liste vide est `[]`.

2 Essayer

Exécuter les instructions suivantes en essayant de comprendre leur utilité. Ne pas hésiter à faire varier les paramètres pour mieux comprendre !

2.1 Définition d'une liste

```
L1 = [1, "hello", 2.3, 47, [2, 45, "aa"], True] #une liste peut contenir n'importe quoi, y
     compris une autre liste
L2 = [1, 2] * 4
L3 = [2*x+1 for x in L2]
L4 = [[1, 2]] * 3  # noter la différence avec L4 = [1, 2] * 3
L5 = [i*i for i in range(-3, 10, 2)]
L6 = [x for x in L5 if x % 2 == 1]
L7 = list(range(-1, 5, 2))
L8 = list("abcdccdd")
```

2.2 Copie d'une liste

Attention : si vous voulez créer une liste `Lp` contenant les mêmes éléments qu'une liste `L`, il ne faut pas utiliser la commande `Lp = L` car alors `L` et `Lp` auraient la même adresse mémoire et toute modification de l'une modifie l'autre. Pour s'en convaincre, tester :

```
L = [0] * 6
Lp = L
Lp[2] = 5
print(Lp)
print(L)
```

Pour copier correctement une liste, on a le choix entre :
`Lp = L.copy()`, `Lp = [x for x in L]` ou `Lp = L[:]`. Tester :

```
L = [0] * 6
Lp = L[:]
Lp[2] = 5
print(Lp)
print(L)
```

2.3 Longueur et extraction d'un élément ou d'une sous-liste

```
len(L1)
len([])
L1[1]
L1[:]
L1[3:]
L1[:3]
L1[1:4]   # comme dans in range, la borne de droite est exclue
L1[1:4:2]
L1[4][1]  # L1[4] est une liste, on en cherche l'élément en place 1
L1[6]
L1[-1]
```

```
L1[2:8]
L1[2:8:2]
```

Remarque : Bien noter que l'accès à un élément ou à une plage nécessite des crochets et non des parenthèses. En cas d'erreur, vous verrez apparaître le message 'list' object is not callable

Remarque : pour les listes, une extraction (ou slicing) de liste avec la syntaxe `L[a:b]` créée une copie. Ainsi la suite d'instruction

```
Lp = L[2:4]
Lp[0] = 77
```

ne modifie pas la liste `L`. **Attention** : c'est différent pour les tableaux (type `array` de la bibliothèque `numpy`).

2.4 Affectation : modifier les éléments d'une liste

```
L1[4] = 789
L1
L1[6] = 5    # génère une erreur de type IndexError (message : index out of range)
L2[8] = 78
L2
```

2.5 Manipulations des listes

```
L4.append(456)  # L4 est modifiée
L4.insert(3, "haha")
L3 = L1+L2      # concaténation de deux listes
L1.extend(L2)   # rallonge L1 en "collant" L2 à sa suite (L1 est modifiée mais pas L2)
L4 = L4 + [123] # pourrait remplacer L4.append(123) (mais moins efficace)
e = L1.pop()    # supprime le dernier élément de L1 et l'affecte à la variable e
L1.pop(2)       # supprime l'élément d'indice 2 de L1
L1.remove(47)   # supprime le premier élément de L1 dont la valeur est 47
```

2.6 Autres commandes à connaître

```
a in L   # Retourne True si a est présent dans L, False sinon
sum(L)   # Retourne la somme des éléments de L si tous les éléments sont numériques
for i in range(len(L)):  # Parcours d'une liste par les indices
    print(L[i])
for e in L:              # Parcours d'une liste par les éléments
    print(e)
```

Remarques :

- Le parcours d'une liste par les éléments doit être préféré à chaque fois qu'il est possible de l'utiliser.
- Attention à une utilisation correcte de `append`, `reverse` et `sort`, qui modifient la liste en place et ne renvoient rien.
  `L.append (7)` est correct tandis que si vous faites `L = L.append(7)` vous risquez d'avoir des surprises dans la suite de votre programme ! Essayez ...puis tapez `type(L)` . Que vous répond-on ?
  De même, on utilise `L.sort()`, `L.reverse()` sans faire d'affectation.

2.7 Remarque sur les créations de listes

Dans de nombreux programmes, on veut construire une liste petit à petit.
Si on connaît à l'avance la longueur `p` de la liste que l'on veut créer mais qu'on ne peut la définir en compréhension (`L = [k*k for k in range(10)]`), on a le choix entre deux stratégies :

- Initialiser la liste avec la bonne longueur : `L = [0] * p`. Puis modifier un à un les éléments de la liste à l'aide d'une boucle.
- Initialiser la liste comme une liste vide : `L = []`. Puis ajouter un à un les éléments à l'aide de l'instruction `append`.

Chaque fois qu'elle est utilisable, la première méthode est préconisée dans les rapports de jury, et elle est plus rapide à l'exécution lorsque le nombre d'éléments est grand.
Pour s'en convaincre, tester les instructions suivantes, qui permettent toutes les deux de créer une liste de 0 de longueur 10 000 000.

```
from time import time       # permet de calculer le temps d'exécution d'un bout de
    programme

t1 = time()
L = [0] * 10**7
for k in range(10**7):
    L[k] = k
t2 = time()
print(t2-t1)

t1 = time()
L = []
for k in range(10**7):
    L.append(k)
t2 = time()
print(t2-t1)
```

3 Exercices

Exercice 1
Comment retirer l'élément d'indice `i` d'une liste sans utiliser `pop` ?

Exercice 2
Si `L` est une liste de longueur `p`, comment calculer $\sum_{i=0}^{p} i * L[i]$?

Exercice 3
Écrire un fonction permettant de « renverser » une liste `L`, sans créer de nouvelle liste (par exemple `[4, 7, 9, 1]` devient `[1, 9, 7, 4]`). Cette fonction ne renverra rien, mais modifiera la liste « sur place ».
Il est ***évidemment*** *interdit d'utiliser* `L.reverse()`.

Exercice 4
Écrire une fonction prenant en paramètres `a` et `L` (une liste) et renvoyant un booléen indiquant si `2*a` est présent dans la liste `L` sans utiliser `2*a in L`.

Exercice 5
Écrire une fonction renvoyant le minimum d'une liste numérique non vide sans utiliser la fonction `min`.