TP N°4: Les listes

Une *liste* est une séquence ordonnée et modifiable d'objets homogènes ou hétérogènes séparés par des virgules et délimités par des crochets [ ]. Elle supporte la répétition et appartient au type `list`.

1. Les valeurs des listes peuvent être de types différents.
2. Les listes sont *indexées* à partir de 0. On accède à l'élément d'indice i d'une liste L par la syntaxe $L[i]$. Noter également que $L[-1]$ désigne le dernier élément de L, $L[-2]$ l'avant-dernier, etc.
3. Les listes sont *modifiables* : `L[i] = x` remplace l'élément d'indice i de L par x.

Création de listes

```
L1 = [] ; L2 = list() # listes vides
L3 = [1,2,3]
L4 = ["a" , 2.5] # types hétérogènes
L5 = [L3, L4] # liste de listes
L6 = list(range(0,1000,200)) # créer une liste à partir d'un intervalle
L7 = [for i in range(0,1000,200)] #Définition par compréhension
L8 = [i ** 2 for i in range(1, 11) if i % 2 == 0]#Définition par compréhension avec une condition
L9 = [[ i for i in range(11)] for j in  range(11)]#Définition par compréhension: liste de listes
```

Exercice 1

1. Créer la liste `liste = [2, 0, 5, [7, 6, 17], 4]`, remplacer le 0 par 8 puis remplacer le 17 par 22.
2. Construire par compréhension la liste des entiers multiples de 7 compris entre 1 et 100.

Fonctions et méthodes associées aux listes

```
L = [2, 0, 5, [7, 6, 17], 4] # liste initiale
L.pop() # permet d'enlever (et de récupérer) le dernier élément d'une liste
L.append(8) # permet d'ajouter un élément à la fin d'une liste.
L.insert(3,15) # Ajouter un élément à une position donnée dans une liste
len(L) # renvoie le nombre d'éléments d'une liste.
del (L) # Supprimer définitivement la liste
L1,L2 = [1, 2, 3] , [4, 5, 6]
L1.extend([1, 2, 3]) # Agrandir la liste L1 par la liste L2
L1 + L2 # concaténer deux listes
[1] * 5 # concaténer plusieurs copies d'une liste avec l'opérateur *.
L1 == L2 # Comparer le contenu de deux listes
2 in [1, 2, 3] # tester avec in et not in si un élément appartient ou non à une liste.
max(L), min(L), sum(L) # Retourner le maximum, le minimum et la somme des éléments d'une liste
L = [2, 1, 4, 3]
sorted(L) # renvoie une copie de L triée. L n'est pas modifiée
L.sort() # ne renvoie rien. L est triée.
```

Exercice 2

1. Construire, à partie d'une nombre entier `n`, la liste `L` composée de ses chiffres.
   Exemple : `n = 1234` → `L = [1,2,3,4]`
2. Ecrire une instruction qui modifie chacune des listes préremplies pour parvenir à la liste `L=[1,2,3,4,5]` (plusieurs réponses possibles):
   a) `L=[1,2,3]` b) `L=[-1,0,1,2,3,4,5]` c) `L=[1,2,3,3.5,4,5]` d) `L=[5,4,3,2,1]`
3. Ecrire un programme python qui permet de saisir une liste de 5 entiers strictement positifs. Afficher la liste triée en ordre descendant.

Itération sur les listes

```
for x in  L: # On peut itérer directement sur une liste  : x représente un élément de L
    print(x)

for i in range(len(L)): # i représente un indice variant de 0 à len(L) - 1
    print(L[i])
```

Exercice 3

1. Ecrire un programme python qui permet de saisir une liste de n nombres et renvoie le produit de ses éléments.
2. Ecrire un programme python qui à partir d'une liste d'entiers L, renvoie un liste de deux listes, la première sous-liste contenant les éléments pairs de L et la seconde ses éléments impairs.
   Exemple : `[1, 2, 4, 7, 3, 2, 1]` → `[ [2, 4, 2], [1, 7, 3, 1] ]`

Slicing

Le *slicing* (découpage en tranches en français) permet de récupérer ou de modifier un morceau d'une liste : `L[i:j]` représente la sous-liste de `L` formée des éléments dont les indices sont compris entre `i` et `j-1`. On peut spécifier un `pas` égal à `k` en écrivant `L[i:j:k]`.

```
L = [2, 8, 1, 3, 5]
L[2:4] # éléments d'indices 2 et 3
L[2:] # éléments d'indices 2 et plus (jusqu'à la fin de la liste)
L[:2] # éléments d'indices strictement inférieurs à 2
L[:] # toute la liste (copie de la liste)
L[1:5:2] # éléments d'indices 1 et 3
L[::-1] # toute la liste mais à l'envers
```

Exercice 4

1. Ecrire un programme python qui permet de saisir une liste de n nombres et renvoie le produit de ses éléments.
2. Ecrire un programme python qui à partir d'une liste d'entiers L, renvoie un liste de deux listes, la première sous-liste contenant les éléments pairs de L et la seconde ses éléments impairs.
   Exemple : `[1, 2, 4, 7, 3, 2, 1]` → `[ [2, 4, 2], [1, 7, 3, 1] ]`

Copier une liste

```
L1 = [1, 2, 3] # on crée en mémoire l'objet [1, 2, 3] et on lui associe le nom L1
L2 = L1; id(L1); id(L2) # on crée une copie de L1. L1 et L2 pointent vers le même objet.
L1[0] = 4; L1 ; L2 # mais si on modifie un élément de L1, L2 est modifiée aussi
L3 = L1[:] ; id(L3) # on crée un nouvel objet en mémoire. L3 pointe vers le nouvel objet créé.
L4 = [1,2,[3,4]]
L5 = L4[:] ; id(L4[2]); id(L5[2]) # copier la référence de la sous liste L4[2]
L5[0] = 5; L5[2][0] = 55; L5; L4
import copy
L6 = copy.copy(L4) # copie superficielle (shallow copy) : ne crée pas une copie de sous listes
L6[0] = 6; L6[2][0] = 66; L6; L4
L7 = copy.deepcopy(L4) # copie approfondie : crée une copie indépendante de sous listes
L7[0] = 7; L7[2][0] = 77; L7; L4
```

Exercice 5

Ecrire un programme python qui à partir d'une liste d'entiers `L` et deux objets `x1` et `x2`, renvoie une copie de `L` où on a remplacé toutes les occurrences de `x1` par `x2`.
Exemple : `x1 = 4; x2 = 1; L = [4, 5, 3, 4, 2, 1]` → `copie = [1, 5, 3, 1, 2, 1]`