

Programmation Python II : tuples et listes

Exercice 1

1. Définir en Python deux listes `L4` et `L6` vérifiant :
 - `L4` est la liste formée des dix premiers entiers non nuls multiples de 4 supérieurs ou égaux à 1.
 - `L6` est la liste formée des dix premiers entiers non nuls multiples de 6 supérieurs ou égaux à 1.
2. Effectuer la concaténation des deux listes : `L4+L6` .
3. Que contient la liste `2*L4` ?
4. Ecrire une fonction `sum_liste` qui renvoie la liste `Lst_som` obtenue en ajoutant terme à terme et dans l'ordre les éléments des deux listes `L4` et `L6` .

Exercice 2

On donne la liste `Points = [[2, 1], [5, 10], [1, 11], [4, 5], [4, -1]]`

Chaque élément de la liste `Points` est une liste représentant les coordonnées d'un point dans le plan muni d'un repère.

1. Ecrire une fonction `f1` qui renvoie la liste des points de la liste `Points` situés en dessous de la droite d'équation $y = x$.
2. Même question pour les points situés en dessus de la droite d'équation $y = 3x - 1$

Exercice 3

1. Ecrire une expression Python dont la valeur est la liste des entiers pairs de 2 à 22 compris.
2. Ecrire une fonction `liste_pairs(a,b)` dont les paramètres `a` et `b` sont des entiers pairs et qui renvoie la liste des entiers pairs de `a` à `b` compris.
3. Ecrire une fonction `listeEntiers` de paramètres `a,b` et `n` qui renvoie une liste de `n` entiers distincts pris au "hasard" entre les entiers `a` et `b` (`a` et `b` compris).
On supposera que $b > a + n$ et on utilisera la fonction `randint` du module `random`.

Exercice 4

Écrire une fonction `separer` qui prend une liste d'entiers en paramètre et retourne un couple (2-uplet) de listes, la première étant la sous-liste des entiers pairs, et l'autre la sous-liste des entiers impairs.

Exercice 5

Le codage par différence (delta encoding en anglais) permet de compresser un tableau d'entiers dont les valeurs sont proches les unes des autres. Le principe est de stocker la première donnée en indiquant pour chaque autre donnée sa différence avec la précédente plutôt que la donnée elle-même.

On se retrouve alors avec un tableau dont les valeurs sont plus petites, nécessitant moins de place en mémoire. Programmer la fonction `delta` qui prend en paramètre un tableau non vide de nombres entiers et qui renvoie un tableau contenant les valeurs entières compressées à l'aide cette technique.

Exemples :

```
>>> delta([1000, 800, 802, 1000, 1003])
[1000, -200, 2, 198, 3]
>>> delta([42])
[42]
```

Exercice 6

Écrire une fonction `recherche_indices_classement` qui prend en paramètres un entier `elt` et un tableau d'entiers `tab` représenté par une liste Python, et qui renvoie trois listes Python d'entiers :

- la première liste contient les indices des valeurs du tableau `tab` strictement inférieures à `elt` ;
- la deuxième liste contient les indices des valeurs du tableau `tab` égales à `elt` ;
- la troisième liste contient les indices des valeurs du tableau `tab` strictement supérieures à `elt`.

Exemples :

```
>>> recherche_indices_classement(3, [1, 3, 4, 2, 4, 6, 3, 0])
([0, 3, 7], [1, 6], [2, 4, 5])
>>> recherche_indices_classement(3, [1, 4, 2, 4, 6, 0])
([0, 2, 5], [], [1, 3, 4])
>>> recherche_indices_classement(3, [1, 1, 1, 1])
([0, 1, 2, 3], [], [])
>>> recherche_indices_classement(3, [])
([], [], [])
```

Exercice 7

On considère une piste carrée qui contient 4 cases par côté. Les cases sont numérotées de 0 inclus à 12 exclu comme ci-dessous :

0	1	2	3
11			4
10			5
9	8	7	6

L'objectif de l'exercice est d'implémenter le jeu suivant : Au départ, le joueur place son pion sur la case 0. A chaque coup, il lance un dé équilibré à six faces et avance son pion d'autant de cases que le nombre indiqué par le dé (entre 1 et 6 inclus) dans le sens des aiguilles d'une montre.

Par exemple, s'il obtient 2 au premier lancer, il pose son pion sur la case 2 puis s'il obtient 6 au deuxième lancer, il le pose sur la case 8, puis s'il obtient à nouveau 6, il pose le pion sur la case 2. Le jeu se termine lorsque le joueur a posé son pion sur toutes les cases de la piste.

Compléter la fonction `nombre_coups` ci-dessous de sorte qu'elle renvoie le nombre de lancers aléatoires nécessaires pour terminer le jeu. Proposer ensuite quelques tests pour en vérifier le fonctionnement.

```
from random import randint
def nombre_coups():
    '''Simule un jeu de plateau avec 12 cases et renvoie le nombre
       minimal de coups pour visiter toutes les cases.'''
    nombre_cases = 12
    # indique si une case a été vue
    cases_vues = [ False ] * nombre_cases
    nombre_cases_vues = 1
    cases_vues[0] = True
    case_en_cours = 0
    n = ...
    while ... < ...:
        x = randint(1, 6)
        case_en_cours = (case_en_cours + ...) % ...
        if ...:
            cases_vues[case_en_cours] = True
            nombre_cases_vues = ...
        n = ...
    return n
```