Algorithmique et Programmation 1 – TD - TP 5
FONCTIONS
CORRECTION

Exercice 1 - Carré magique
L'objectif de cet exercice est de vérifier les contraintes des carrés magiques représentés par des listes de listes d'entiers. Par la même occasion, on illustre ici la décomposition d'un problème (relativement) complexe en un certain nombre de sous-problèmes plus simples.

Un carré magique de dimension n est une matrice de taille $n \times n$ contenant des entiers naturels, telle que :
- tous les entiers de 1 à $n * n$ sont présents dans la matrice (*donc tous les éléments sont distincts*),
- les sommes des entiers de chaque ligne sont toutes égales à une même valeur S,
- la somme de chaque colonne est également S,
- les sommes des deux diagonales de la matrice sont également S.

Par exemple, la matrice 3×3 suivante est un carré magique de dimension 3 :

2	7	6
9	5	1
4	3	8

En Python, nous allons représenter un carré magique par une liste de listes d'entiers : `list[list[int]]`. Pour un carré magique de taille n, chaque ligne de la matrice est représentée par une liste d'entiers. Les lignes sont elle-mêmes stockées dans une liste dans l'ordre séquentiel (la première liste correspond à la première ligne, la seconde liste correspond à la deuxième ligne, etc.).

Voici une expression permettant de construire le carré magique ci-dessus :

```
>>> CarreMagique = [[2, 7, 6],
...                 [9, 5, 1],
...                 [4, 3, 8]]
```

- Les lignes sont : `[2, 7, 6]`, `[9, 5, 1]` et `[4, 3, 8]`
- Les colonnes sont : `[2, 9, 4]`, `[7, 5, 3]` et `[6, 1, 8]`
- Les diagonales sont : `[2, 5, 8]` et `[6, 5, 4]`

1. Définir la fonction `appartient_listeliste` qui étant donné un entier `n` et une liste de listes d'entiers `ll` retourne `True` si l'entier `n` est présent dans la liste `ll`, `False` sinon.
   Par exemple :

```
>>> appartient_listeliste(5, [[1, 2, 3], [4, 5, 6]])
True
>>> appartient_listeliste(7, [[1, 2, 3], [4, 5, 6]])
False
>>> appartient_listeliste(7, CarreMagique)
True
>>> appartient_listeliste(12, CarreMagique)
False
```

```
def appartient_listeliste(n, ll) :
    """ Int x List --> Bool
    Fonction retournant True si l'entier appartient à une liste formant la liste ll"""
    app = False
    i = 0
    while i < len(ll) and not(app) :
        if n in ll[i] :
            app = True
        i = i + 1
    return(app)
```

2. Définir la fonction `verif_elems` qui, étant donné un entier naturel n non nul et une matrice `mat` d'entiers, retourne `True` si tous les entiers dans l'intervalle $[1, n*n]$ sont présents dans la matrice `mat`, `False` sinon.
   Par exemple :

```
>>> verif_elems(3, CarreMagique)
True
>>> verif_elems(3, [ [2, 7, 6], [8, 5, 1], [4, 3, 8] ])
False
```

```
def verif_elems(n, mat) :
    """ Int x List --> Bool, n!= 0
    Fonction retournant True si tous les entiers dans l'intervalle [1, n*n] sont
    présents dans la matrice mat, faux sinon"""
    assert not(n==0)
    app = True
    i = 1
    while app and i <= n*n :
        if appartient_listeliste(i, mat) :
            i = i + 1
        else :
            app = False
    return(app)
```

3. Définir la fonction `somme_liste` qui retourne la somme des éléments d'une liste d'entiers.
   Par exemple :

```
>>> somme_liste([2, 7, 6])
15
```

```
def somme_liste(liste):
    """List --> Int
    Somme des éléments d'une liste d'entiers"""
    somme = 0
    for i in range(len(liste)) :
        somme = somme + liste[i]
    return(somme)
```

4. Définir la fonction `verif_lignes` telle que `verif_lignes(mat, somme)` retourne `True` si toutes les sous-listes de `mat` possèdent la même somme `somme`, `False` sinon.
   Par exemple :

```
>>> verif_lignes(CarreMagique, 15)
True
>>> verif_lignes(CarreMagique, 16)
False
>>> verif_lignes([ [2, 7, 6], [8, 5, 1], [4, 3, 8] ], 15)
False
```

```
def verif_lignes(mat, somme):
    """List x Int --> Bool
    Fonction retournant vrai si toutes les sous-listes de la liste "mat"
    ont pour somme "somme" """
    rep = True
    i = 0
    while rep and i < len(mat):
        if somme_liste(mat[i]) == somme:
            i = i+1
        else :
            rep = False
    return(rep)
```

5. Définir la fonction `colonne` qui, étant donné un entier `k` et une matrice `mat` de dimension n, retourne la `k-ième` colonne de la matrice `mat`. On fait l'hypothèse que `k` est compris entre 0 et $n-1$.
   Par exemple :

```
>>> colonne(0, CarreMagique)
[2, 9, 4]
>>> colonne(1, CarreMagique)
[7, 5, 3]
>>> colonne(2, CarreMagique)
[6, 1, 8]
```

```
def colonne(k, mat):
    """Int x List --> List
    Fonction qui retourne la kème colonne d'une matrice mat"""
    assert k >=0 and k<len(mat)
    col = []
    for i in range(len(mat)) :
        col.append(mat[i][k])
    return col
```

6. Définir la fonction `verif_colonnes` telle que `verif_colonnes(mat, somme)` retourne `True` si toutes les colonnes de `mat` possèdent la même somme `somme`, `False` sinon.
   Par exemple :

```
>>> verif_colonnes(CarreMagique, 15)
True
>>> verif_colonnes(CarreMagique, 16)
False
>>> verif_colonnes([ [2, 7, 6], [8, 5, 1], [4, 3, 8] ], 15)
False
```

```
def verif_colonnes(mat, somme):
    """List x Int --> Bool
    Fonction retournant vrai si toutes les colonnes de la matrice "mat"
    ont pour somme "somme" """
    rep = True
    i = 0
    while rep and i < len(mat):
        if somme_liste(colonne(i, mat)) == somme:
        i = i+1
    else :
        rep = False
    return(rep)
```

7. Définir la fonction `diagonale1` (resp. `diagonale2`) qui, étant donné une matrice `mat`, retourne la liste formée des éléments de la première diagonale (resp. seconde diagonale).
Par exemple :

```
>>> diagonale1(CarreMagique)
[2, 5, 8]
>>> diagonale2(CarreMagique)
[6, 5, 4]
>>> diagonale1([ [ 4, 14, 15, 1], [ 9, 7, 6, 12], [ 5, 11, 10, 8], [16, 2, 3, 13]])
[4, 7, 10, 13]
>>> diagonale2([ [ 4, 14, 15, 1], [ 9, 7, 6, 12], [ 5, 11, 10, 8], [16, 2, 3, 13]])
[1, 6, 11, 16]
```

```
def diagonale1(mat) :
    """List --> List
    Retourne la première diagonale de la matrice mat"""
    diag = []
    for i in range(len(mat)) :
        diag.append(mat[i][i])
    return(diag)

def diagonale2(mat):
    """List --> List
    Retourne la deuxième diagonale de la matrice mat"""
    diag = []
    n = len(mat)
    for i in range(n) :
        diag.append(mat[i][n-i-1])
    return(diag)
```

8. Définir (enfin !) la fonction `verif_magique` qui, étant donné une matrice `mat` de dimension n, retourne `True` si et seulement si elle représente un carré magique.
Par exemple :

```
>>> verif_magique(CarreMagique)
True
>>> verif_magique([[2, 7, 6],
...                [8, 5, 1],
...                [4, 3, 8]])
False
```

```
def verif_magique(mat):
    """List --> Bool
    Vrai si la matrice donnée en paramètre est un carré magique
    (tous les éléments de 1 à n*n sont présents, la somme des entiers de chaque ligne,
    colonne et diagonale est égale au même entier s)"""
    n = len(mat)
    s = somme_liste(mat[0])
    if not verif_elems(n, mat) :
        return(False)
    elif not verif_lignes(mat, s):
        return(False)
    elif not verif_colonnes(mat, s) :
        return(False)
    elif not (s == somme_liste(diagonale1(mat))):
        return(False)
    elif not (s == somme_liste(diagonale2(mat))) :
        return(False)
    else:
        return(True)
```