Compléments : Travailler sur des "tableaux de tableaux"

En algorithmique ou en Python, chaque élément d'un tableau peut aussi être un tableau. On parle alors de tableau de tableau ou de tableau bi-dimensionnel.

Voici un exemple de tableau de tableau :

```
m = [[1, 3, 4], [5 ,6 ,8], [2, 1, 3], [7, 8, 15]]
```

Le premier élément du tableau ci-dessus est bien un tableau ([1, 3, 4]), le deuxième élément est aussi un tableau ([5, 6, 8])…

De manière plus lisible, on peut initialiser ces "tableaux de tableaux" comme suit :

```
m = [[1, 3, 4],
     [5, 6, 8],
     [2, 1, 3],
     [7, 8, 15]]
```

Nous obtenons ainsi quelque chose qui ressemble beaucoup à un "objet mathématique" appelé matrice.

Il est évidemment possible d'utiliser les indices de position avec ces "tableaux de tableaux". Pour cela nous allons considérer notre tableau de tableaux comme une matrice, c'est à dire en utilisant les notions de "ligne" et de "colonne".

Dans l'exemple ci-dessus, en ce qui concerne les lignes :

- 1, 3, 4 constituent la première ligne
- 5, 6, 8 constituent la deuxième ligne
- 2, 1, 3 constituent la troisième ligne
- 7, 8, 15 constituent la quatrième ligne

En ce qui concerne les colonnes :

- 1, 5, 2, 7 constituent la première colonne
- 3, 6, 1, 8 constituent la deuxième colonne
- 4, 8, 3, 15 constituent la troisième colonne

Pour cibler un élément particulier de la matrice, on utilise la notation avec "doubles crochets" : `m[ligne][colonne]`

Remarque : Ne pas oublier que la première ligne et la première colonne ont pour indice 0.

— À faire vous-même 1 —

Quelle est la valeur référencée par les variables a et b après l'exécution du programme ci-dessous ? Utilisez **trinket.io** pour vérifier votre réponse)

```
m = [[1, 3, 4],
     [5, 6, 8],
     [2, 1, 3],
     [7, 8, 15]]
a = m[1]
b = m[1][2]
```

Il est possible de parcourir l'ensemble des éléments d'une matrice à l'aide d'une "double boucle for" :

— À faire vous-même 2 —

Analysez puis testez le code suivant :

```
m = [[1, 3, 4],
     [5, 6, 8],
     [2, 1, 3],
     [7, 8, 15]]
nb_colonne = 3
nb_ligne = 4
for i in range(0, nb_ligne):
    for j in range(0, nb_colonne):
        a = m[i][j]
        print(a)
```

Mini-Projet : le carré magique

Un carré d'ordre n est un tableau carré contenant n^2 entiers strictement positifs. On dit qu'un carré d'ordre n est magique si :

- il contient tous les nombres entiers de 1 à n^2 ;
- les sommes des nombres de chaque rangée, de chaque colonne et de chaque grande diagonale sont égales.

On modélise un carré par une liste de liste de nombres.

```
carre3 = [
   [2, 7, 6],
   [9, 5, 1],
   [4, 3, 8]]
```

```
carre4 = [
   [4, 5, 11, 14],
   [15, 10, 8, 1],
   [6, 3, 13, 12],
   [9, 16, 2, 7]]
```

1. **a.** En Python, quelle est la valeur de `len(carre4)` ?
   b. Quelle est la valeur de `carre3[1]` ? de `carre3[0][2]` ?
   c. Quelle instruction permet de récupérer la valeur 3 de carre 4 ?

2. **a.** On propose le code suivant :

```
def sommeLigne(carre: list, n: int) -> int:
   """ carre est une liste de liste
```

```
    n est un nombre entier
    """
    somme = 0
    for nombre in carre[n]:
         somme = somme + nombre
    return somme
```

Que vaut `sommeLigne(carre4,2)` ?

Expliquez clairement à quoi sert cette fonction.

b. Écrire le code Python d'une fonction qui prend un carré en paramètre et qui vérifie que les sommes des nombres de chaque ligne sont égales.

c. Proposer le code Python d'une fonction qui prend un carré en paramètre, ainsi que le numéro d'une colonne et qui renvoie la somme des nombres de cette colonne.

3. **a.** En pseudo-code, écrivez une fonction estMagique() qui prend un carré en paramètre et renvoie True si le carré est magique et False sinon.

   b. Traduisez votre code en Python et testez-le sur différents carrés magiques.