1 Compréhension du cours

1.1 Grilles et listes de listes

Soit le tableau ci-dessous :

1	2	3
4	5	6

1. On représente ce tableau par des listes de listes. Définir des variables qui représentent respectivement :

- `g_ligne` : la liste des lignes du tableau
- `g_col` : la liste des colonnes du tableau

2. Quelle est la longueur de `g_ligne` et `g_col` ? D'un élément de `g_ligne` et `g_col` ?

3. Donner une suite d'instructions qui ajoute une colonne contenant les entiers `7` et `8` à la droite du tableau :

- dans le cas d'une liste de lignes
- dans le cas d'une liste de colonnes

1.2 Aliasing

4. En représentant l'état de la mémoire de la manière qui vous semble la plus appropriée, indiquer les valeurs prises / référencées par les variables lors de l'exécution des codes suivants. Vous devez préciser quelles sont les **valeurs finales** référencées par chaque variable.

a.
```
x = 1
y = x
x = x + 1
```

b.
```
liste1 = [1, 2, 3]
liste2 = liste1
liste1.append(4)
```

c.
```
liste1 = [1, 2, 3]
liste2 = liste1
liste2 = liste2 + [4]
```

On considère le code suivant :

```
# crée une grille de 0 avec 1 au centre
ligne = [0, 0, 0]
g = [None] * 3
for i in range(3):
    g[i] = ligne
g[1][1] = 1
```

5. À votre avis, quelle était l'intention de la personne qui a écrit ce code ? Quelle est la valeur référencée par `g` en fin d'exécution ? Que changer pour revenir à l'intention première ?

On considère la fonction suivante :

```
def positionne(grille:list[list[int]], i:int, j:int, elem:int) -> list[list[int]]:
    g = grille.copy()
    g[i][j] = elem
    return g
```

et l'appel suivant :

```
>>> g1 = [[1, 2], [3, 4]]
>>> g2 = positionne(g1, 1, 1, 5)
```

6. Donner la valeur de `g1` et `g2` à la fin de l'éxécution.

7. Comment modifier le code de `positionne` pour être sûr qu'il n'y aura aucun problème d'aliasing ?

2 Grilles et listes de listes, exercices de programmation.

On s'intéresse dans un premier temps à des grilles carrées contenant des caractères `'*'` et `'+'` , représentées par la liste de leurs **lignes**.

8. Écrire une fonction `init_grille` qui prend en paramètre un entier strictement positif `n` et qui renvoie une liste de liste de caractères représentant une grille carrée de taille `n` et contenant des `'*'`.
9. Écrire une fonction `afficher_grille` qui prend en paramètre une liste de liste de caractères représentant une grille carrée et l'affiche en intercalant un caractère espace entre 2 éléments consécutifs d'une même ligne.
10. Écrire une fonction `grille_triangle_inf` qui prend en paramètre un entier strictement positif `n` et qui renvoie une liste de liste de caractères représentant une grille carrée de taille `n` et contenant des `'*'` sauf sur et sous la première diagonale (sens \) qui contient des `'+'`.

Par exemple `grille_triangle_inf(3)` vaut `[['+', '*', '*'], ['+', '+', '*'], ['+', '+', '+']]`.

11. Voyez-vous comment modifier le code de `grille_triangle_inf` pour générer la liste des paires des indices des cases contenant un `'+'` ?
12. De même écrire une fonction `grille_triangle_sup` qui prend en paramètre un entier strictement positif `n` et qui renvoie une liste de liste de caractères représentant une grille carrée de taille `n` et contenant des `'*'` sauf sur et au dessus de la première diagonale (sens \) qui contient des `'+'`.

Par exemple `grille_triangle_sup(3)` vaut `[['+', '+', '+'], ['*', '+', '+'], ['*', '*', '+']]`

13. Écrire une fonction `complemente_case` qui prend en paramètre une liste de liste de caractères représentant une grille carrée et des indices `i` et `j` entiers dans cette grille, et renvoie une nouvelle grille équivalente au paramètre sauf la case d'indices (i, j) remplacée par un `'+'` si elle contenait un `'*'`, ou par un `'*'` si elle contenait un `'+'`.
14. Écrire une fonction `grille_croix` qui prend en paramètre un entier `n` impair strictement positif et renvoie une liste de liste de caractères représentant une grille carrée de taille `n` contenant des `'*'` sauf une croix centrale contenant des `'+'`. Par exemple `grille_croix(5)` vaut `[['*', '*', '+', '*', '*'], ['*', '*', '+', '*', '*'], ['+', '+', '+', '+', '+'], ['*', '*', '+', '*', '*'], ['*', '*', '+', '*', '*']]`

On suppose maintenant qu'on travaille avec une grille à `nbl` lignes et `nbc` colonnes, toujours représentée par la liste de ses lignes.

15. Écrire une fonction `colonne` qui prend en paramètre une liste de liste représentant une grille et un indice de colonne `ind` valide dans cette grille et renvoie la colonne d'indice `ind`.
16. Écrire une fonction `transposee` qui prend en paramètre une liste de liste représentant une grille non vide et renvoie une nouvelle grille qui est la transposée du paramètre.

Pour cet exercice on ne travaille pas sur le contenu des cases d'une grille mais simplement sur les indices de ses cases. Il est nécessaire d'introduire des fonctions intermédiaires pour rendre les calculs plus lisibles.

17. Écrire une fonction `cases_premiere_diagonale` qui prend en paramètre les dimensions d'une grille contenant `nbl` lignes et `nbc` colonnes et les indices `(i0, j0)` d'une case dans cette grille, et qui renvoie la liste des positions de la première diagonale passant par `(i0, j0)` (sens \), en partant du haut à gauche

Par exemple :

- `cases_premiere_diagonale(2, 0, 5, 7)` vaut `[(2,0), (3,1), (4,2)]`
- `cases_premiere_diagonale(4, 4, 5, 7)` vaut `[(0,0),(1,1), (2,2), (3,3) ,(4,4)]`

18. Écrire une fonction `cases_deuxieme_diagonale` qui prend en paramètre les dimensions d'une grille contenant `nbl` lignes et `nbc` colonnes et les indices `(i0, j0)` d'une case dans cette grille, et qui renvoie la liste des positions de la deuxième diagonale passant par `(i0, j0)` (sens /), en partant du haut à droite.

Par exemple :

- `cases_deuxieme_diagonale(2, 0, 5, 7)` vaut `[(0,2), (1,1), (2, 0)]`
- `cases_deuxieme_diagonale(3, 3, 5, 7)` vaut `[(0, 6), (1, 5), (2, 4), (3, 3), (4, 2)]`