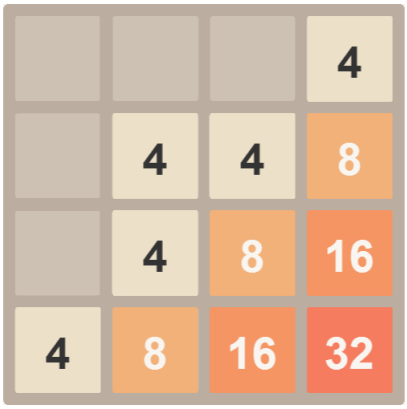
Le 2048 *est un jeu vidéo développé par Gabriele Cirulli en* 2014. *Le principe inspiré de celui du taquin consiste à faire glisser des tuiles numériques sur une grille carré de* 4×4 *cases jusqu'à obtenir le nombre* 2048. *Par exemple si on déplace les tuiles vers la droite, elles glissent toutes horizontalement si des cases vides le permettent et deux cases adjacentes sur la même ligne et portant le même nombre se combinent en une seule case portant le double du nombre. Si on déplace les tuiles vers la gauche ou verticalement vers le bas ou vers le haut, on peut de même combiner des cases sur une même ligne ou une même colonne. Avant chaque mouvement un* 2 *ou un* 4 *apparaît aléatoirement sur l'une des cases vides.*



L'objectif de ce DM est de développer un jeu de 2048 qui s'exécute dans la console `Python`. On donne ci-dessous une capture d'écran d'un début de partie. La grille est affichée avant chaque mouvement avec le 2 ou le 4 apparu aléatoirement dans l'une des cases vides après le mouvement précédent.

```
In [*]: 1 jeu_2048()
|  0  |  0  |  0  |  0  |
|  0  |  0  |  0  |  0  |
|  0  |  0  |  0  |  4  |
|  0  |  0  |  0  |  0  |
--------------------
Déplacement 4 (gauche) 6 (droite)  8 (haut) 9 (bas) ?  4
|  0  |  0  |  0  |  0  |
|  0  |  4  |  0  |  0  |
|  4  |  0  |  0  |  0  |
|  0  |  0  |  0  |  0  |
--------------------
Déplacement 4 (gauche) 6 (droite)  8 (haut) 9 (bas) ?  9
|  0  |  0  |  0  |  0  |
|  0  |  0  |  0  |  0  |
|  0  |  2  |  0  |  0  |
|  4  |  4  |  0  |  0  |
--------------------
Déplacement 4 (gauche) 6 (droite)  8 (haut) 9 (bas) ?  4
|  0  |  0  |  0  |  0  |
|  0  |  0  |  2  |  0  |
|  2  |  0  |  0  |  0  |
|  8  |  0  |  0  |  0  |
--------------------

Déplacement 4 (gauche) 6 (droite)  8 (haut) 9 (bas) ?
```

1. Créer un fichier `2048-Eleve1-Eleve2.py` dans un éditeur `Python`. Les codes `Python` des questions suivantes seront ajoutés successivement dans ce fichier.

2. Importer les fonctions `random` et `randint` du module `random` avec les instructions suivantes :

```
from random import randint, random
```

3. Écrire une fonction `grille_remplie(val)` qui retourne une liste de listes représentant une grille carrée de 4 lignes et 4 colonnes dont toutes les cases contiennent la valeur `val` passée en argument. On pourra compléter le modèle ci-dessous. Dans le reste du devoir on désigne par `grille` une liste de listes de 4 lignes et 4 colonnes.

```
def grille_remplie(val):
    grille = []
    for lig in range(4):
        ligne = []
        for col in range(4):
            ................
        ................
    return grille
```

Voici un exemple d'exécution souhaitée :

```
In [2]: grille_remplie(734)
Out[2]:
[[734, 734, 734, 734],
 [734, 734, 734, 734],
 [734, 734, 734, 734],
 [734, 734, 734, 734]]
```

On donne le code d'une fonction `afficher_grille(grille)` qui permet d'afficher une grille dans la console.

```
def afficher_grille(grille):
    for lig in range(4):
        for col in range(4):
            print('|{:^5d}'.format(grille[lig][col]),end='')
        print('|')
```

Voici un exemple d'exécution :

```
In [5]: afficher_grille(grille_remplie(2048))
| 2048 | 2048 | 2048 | 2048 |
| 2048 | 2048 | 2048 | 2048 |
| 2048 | 2048 | 2048 | 2048 |
| 2048 | 2048 | 2048 | 2048 |
```

4. Pour décaler tous les nombres d'une même ligne vers la gauche on propose la fonction suivante qui modifie la ligne `nlig` de la `grille` passée en argument. L'algorithme comprend une boucle externe `Tant que` contenant deux boucles internes `Tant que` successives.

```
Fonction deplacer_gauche(grille, nlig)
    fin_zero ← 0
    debut_zero ← 0
    Tant que fin_zero < 4
        # Point d'arrêt 1
        Tant que debut_zero < 4 et grille[nlig][debut_zero] != 0
            debut_zero ← debut_zero + 1
        Si debut_zero == 4 alors
            fin_zero ← 4
        Sinon
            fin_zero ← debut_zero + 1
        # Point d'arrêt 2
        Tant que fin_zero < 4 et grille[nlig][fin_zero] == 0
            fin_zero ← fin_zero + 1
        Si fin_zero < 4 alors
            tmp ← grille[nlig][fin_zero]
            grille[nlig][fin_zero] ← grille[nlig][debut_zero]
            grille[nlig][debut_zero] ←tmp
        # Point d'arrêt 3
```

a. Quelle est la condition d'arrêt de la boucle `Tant que` externe ?

b. On se place dans le corps de la boucle externe :

- Que représente le contenu de la variable `debut_zero` à la fin de la première boucle `Tant que` interne ?
- Que représente le contenu de la variable `fin_zero` à la fin de la seconde boucle `Tant que` interne ?

c. Exécuter la fonction `deplacer_gauche` sur la ligne `[0,4,0,2,0]` en affichant dans un tableau les valeurs des variables `debut_zero` et `fin_zero` au niveau des trois points d'arrêt placés dans le corps de boucle externe.

5. Implémenter en `Python` la fonction `deplacer_gauche(grille, nlig)`.

6. Écrire en `Python` une fonction `deplacer_droite(grille, nlig)` qui modifie la `grille` passée en argument en décalant tous les nombres de la ligne `nlig` vers la droite.

7. Recopier et compléter la fonction `transpose(grille)` ci-dessous pour qu'elle retourne la grille transposée de la `grille` passée en argument : les colonnes deviennent des lignes et vice-versa.

```
def transpose(grille):
    grille_trans = grille_remplie(0)
    for lig in range(4):
        for col in range(4):
            ....................
    return grille_trans
```

Voici un exemple d'exécution souhaitée :

```
In [12]: grille = [[2,0,4,4], [4,4,2,4], [8,16,32,8], [2,4,2,4]]

In [13]: transpose(grille)
Out[13]: [[2, 4, 8, 2], [0, 4, 16, 4], [4, 2, 32, 2], [4, 4, 8, 4]]
```

8. Écrire une fonction `decaler_haut(grille, ncol)` qui retourne une **nouvelle** grille, distincte de la `grille` passée en argument, obtenue en décalant tous les nombres non nuls de `grille` vers le haut. On utilisera la fonction `transpose`.

Voici un exemple d'exécution souhaitée :

```
In [17]: grille1 = [[2,0,4,4], [0,2,0,0], [4,2,0,4], [0,4,2,8]]

In [18]: grille2 = decaler_haut(grille1, 0)

In [19]: grille2
Out[19]: [[2, 0, 4, 4], [4, 2, 0, 0], [0, 2, 0, 4], [0, 4, 2, 8]]
```

9. Écrire une fonction `decaler_bas(grille, ncol)` qui retourne une **nouvelle** grille, distincte de la `grille` passée en argument, obtenue en décalant tous les nombres non nuls de `grille` vers le bas.

10. Écrire une fonction `fusion_ligne_gauche(grille, nlig)` qui combine les cases adjacentes d'une même ligne en remplaçant deux cases adjacentes portant le même nombre par une case avec le double à gauche et une case vide à droite.

 La fonction doit s'appliquer à une ligne traitée d'abord par `decaler_gauche(grille, ncol)` et modifier la `grille` passée en argument.

Voici un exemple d'exécution souhaitée :

```
In [59]: grille2 = [[8,8,4,4], [0,2,0,0], [4,2,0,4], [0,4,2,8]]

In [60]: fusion_ligne_gauche(grille2, 0)

In [61]: grille2
Out[61]: [[16, 8, 0, 0], [0, 2, 0, 0], [4, 2, 0, 4], [0, 4, 2, 8]]
```

11. Écrire une fonction `fusion_ligne_droite(grille, nlig)` qui effectue le même traitement que la fonction `fusion_ligne_gauche` mais en fusionnant les cases vers la droite. La fonction doit s'appliquer à une ligne traitée d'abord par `decaler_droite(grille, ncol)` et modifier la `grille` passée en argument.

12. Écrire des fonctions `fusion_col_haut(grille, ncol)` et `fusion_col_bas(grille, ncol)` qui fusionnent les cases de la `grille` respectivement vers le haut ou vers le bas. Les fonctions doivent s'appliquer à une colonne traitée d'abord par `decaler_haut(grille, ncol)` ou `decaler_bas(grille, ncol)` mais en retournant une nouvelle grille distincte de la `grille` passée en argument.

Voici un exemple d'exécution souhaitée :

```
In [116]: grille4 = [[8,8,4,0], [0,2,0,4], [4,2,0,4], [0,4,2,4]]

In [117]: fusion_col_bas(grille4, 3)
Out[117]: [[8, 8, 4, 0], [0, 2, 0, 0], [4, 2, 0, 8], [0, 4, 2, 4]]

In [118]: grille3 = [[8,8,4,4], [0,2,0,4], [4,2,0,4], [0,4,2,0]]

In [119]: fusion_col_haut(grille3, 3)
Out[119]: [[8, 8, 4, 4], [0, 2, 0, 8], [4, 2, 0, 0], [0, 4, 2, 0]]
```

13. Recopier et compléter la fonction `liste_case_vide(grille)` qui retourne la liste des couples d'index `(lig, col)` des cases vides (avec 0) dans la `grille` passée en argument.

```
def liste_case_vide(grille):
    L = []
    for lig in range(4):
        for col in range(4):
            if grille[lig][col] == 0:
                ....................
    return L
```

14. Écrire une fonction `maximum(grille)` qui retourne le nombre maximal contenu dans la `grille` passée en argument.

15. Recopier et compléter la fonction `jeu2048()` pour que son exécution dans la console permette de simuler une partie de 2048 qui s'arrête lorsqu'il n'y a plus de cases vides dans la grille pour y placer aléatoirement un 2 ou un 4 ou lorsque le maximum de la grille est 2048.

```
def jeu_2048():
    #initialisation
    urne = [2,4]
    grille = grille_remplie(0)
    case_vide = liste_case_vide(grille)
    (lig, col) = case_vide[randint(0, len(case_vide) - 1)]
    grille[lig][col] = urne[randint(0, 1)]
    afficher_grille(grille)
    while ................:
        #boucle de jeu
        print('-'*20)
        deplacement = int(input("Déplacement 4 (gauche) 6 (droite) 8 (
            haut) 9 (bas) ? "))
        ....................
        ....................
```