Rapport Calcul Scientifique 2

2 - Exemple de la première itération du jeu

Dans un premier temps, on considère que le jeu se déroule dans une grille de taille 5x5. Initialiser un tableau numpy tab de taille 5x5 contenant les valeurs suivantes:

0|0|0|0|0

0|0|0|0|0

0|0|0|1|0

0|0|1|1|1

0|0|0|1|0

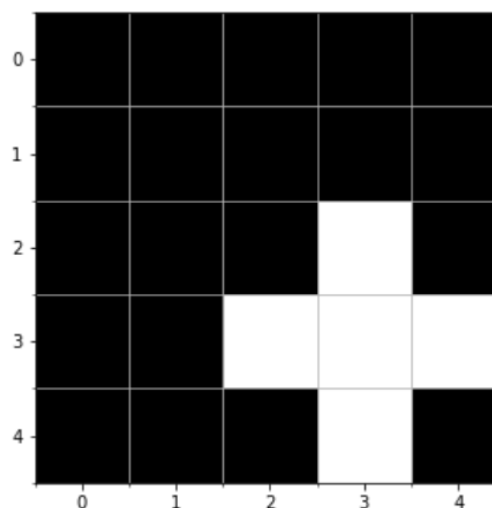
```
tab = np.zeros((5,5))
tab[2,3]=1
tab[3,2:5]=1
tab[4,3]=1
tab
```

Pour créer la matrice désirée, on créer d'abord une matrice de taille 5x5 nul grâce à np.zeros(), puis on remplace les valeurs par 1 aux coordonnées désirés :

tab[2,3]=1 stocke la valeur 1 à la deuxième ligne et troisième colonne de tab.

En utilisant le TP3, afficher sous forme d'image la matrice tab.

```
plt.figure(figsize=(15,15))
plt.imshow(tab,cmap='gray')
plt.title('Matrice tab')
# Affichage de la grille
ax = plt.gca()
ax.set_xticks(np.arange(0, 5))
ax.set_yticks(np.arange(0, 5))
ax.set_xticks(np.arange(-.5, 5, 1), minor=True);
ax.set_yticks(np.arange(-.5, 5, 1), minor=True);
ax.set_xticklabels(np.arange(5))
ax.set_yticklabels(np.arange(5))
plt.grid(which='minor')
plt.show()
```



-plt.figure(figsize=(5,5)) permet de choisir la taille de l'affichage

-plt.imshow(tab,cmap='gray') affiche le graphique, ici en noir et blanc grâce à cmap='gray'

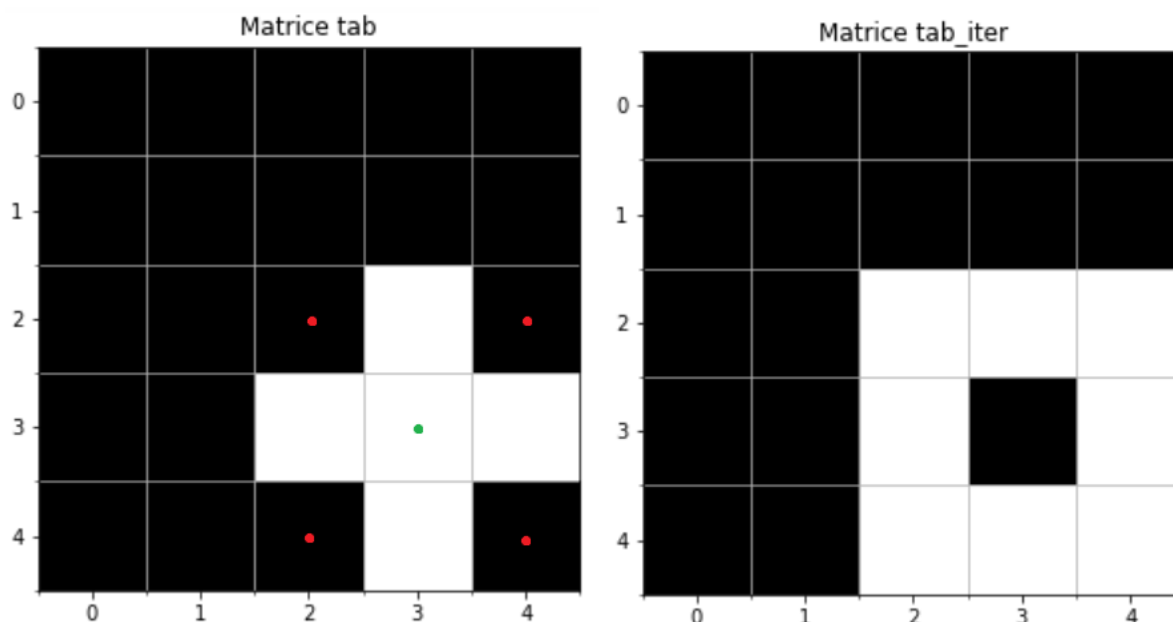
-ax = plt.gca() stocke les axes du graphique dans ax

-ax.set_xticks(np.arange(0, 5)) affiche les valeurs correspondantes (0 à 4) sur les axes du graphique

-La suite du code (de la ligne 8 à 13) permet l'affichage de la grille.

Par la suite, on considère les règles suivantes du jeu de la vie :

- Une case vide entourée d'exactement trois cellules donne naissance à une nouvelle cellule.
- Une cellule ayant strictement plus de trois voisines ou strictement moins de deux voisines sera morte à l'itération suivante.
- On considérera qu'une case a 8 cases voisines (haut, bas, droite, gauche et les 4 diagonales).



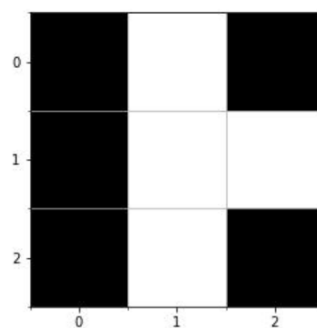
Ici les cases rouges ont exactement 3 voisins vivant, donc vivent à leur tour.

La case verte a exactement 4 voisins, donc elle meurt.

Maintenant, on va travailler sur une grille plus grande représentant le jeu de la vie. Déclarer un tableau de taille 50x50 contenant des zéros qui repésente la grille. Vous nommerez ce tableau `grille`.

```
grille=np.zeros((50,50))
```

Déclarez un tableau de taille 3x3 correspondant à l'image suivante:



Les cases blanches auront pour valeur 1 et les cases noires seront à 0. Vous nommerez ce tableau `debut`.

```
debut=np.zeros((3,3))
debut[0:3,1]=1
debut[1,2]=1
```

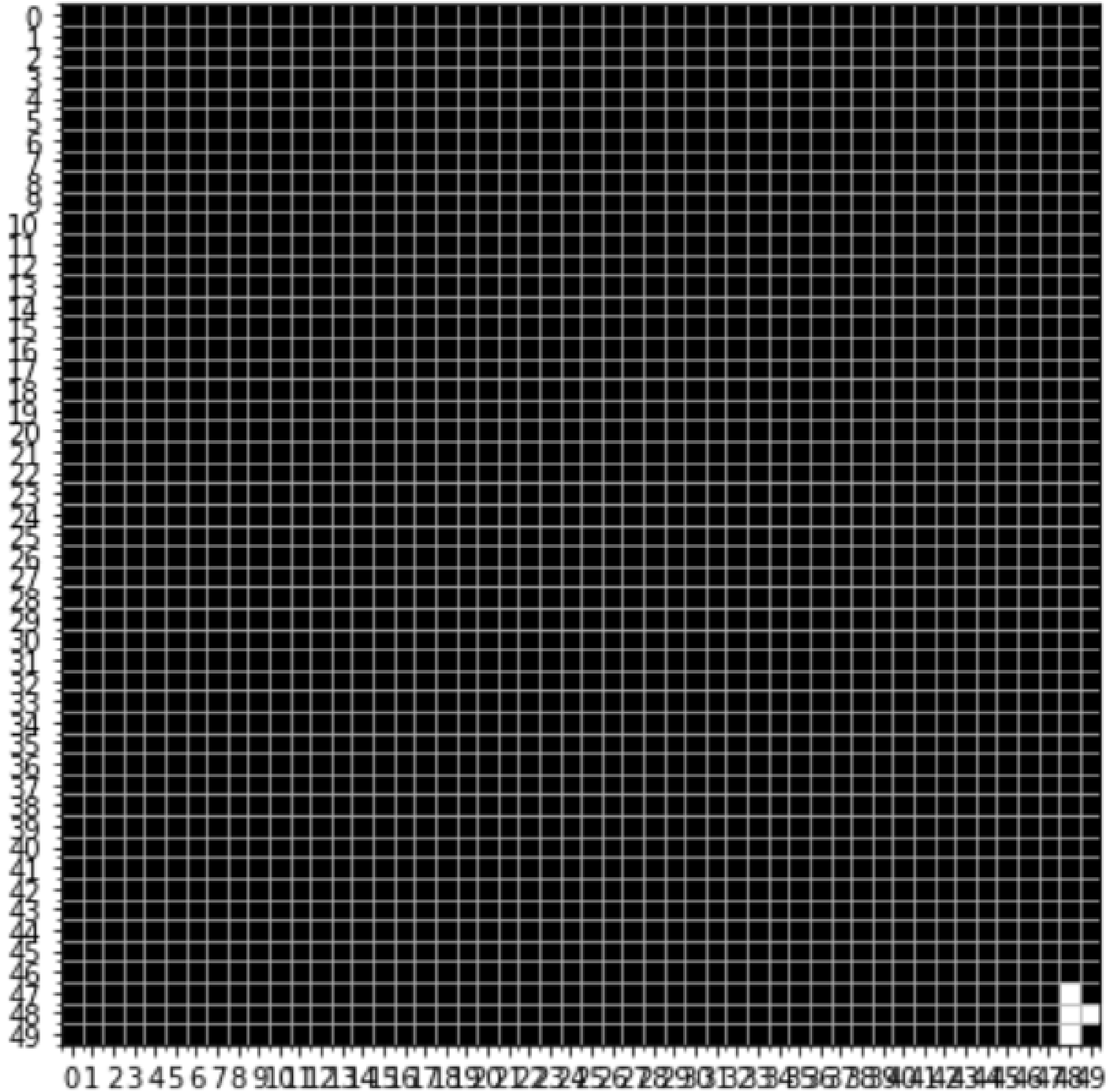
On utilise la même commande que précédemment. *grille* représentera la grille de jeu.

Recopiez le contenu du tableau `debut` en bas à droite dans le tableau `grille`. Vous devez faire celà en une seule instruction.

```
grille[47:50,47:50]=debut
```

Le tableau début est de dimension (3,3), la grille est de dimension (50,50), il faut donc placer le tableau debut aux positions 50-3=47, donc de la case 47 à 50, soit [47 :50,47 :50]

On obtient :



3.1 - Calcul du nombre de voisines en utilisant les coordonnées des cases voisines

Déclarez un tableau `e` contenant toutes les entiers de 0 à 50 non compris.

```
e=np.arange(0,50,1)
```

Np.arange permet tirer des valeurs dans un intervalle suivant un certains pas, ici on tire les valeurs dans l'intervalle [0,50[avec un pas de 1

Recopiez ce tableau 50 fois pour avoir une matrice `N` contenant `e` sur chaque ligne.

```
N = np.tile(e,(50,1))
N.shape
```

```
(50, 50)
```

Np.tile recopie un tableau passer en argument (ici e), 50 fois, sur les lignes (axe 1).

Concaténez la transposé de `N` avec `N`. Vous mettrez le résultat dans la variable `points`.

```
points = np.stack((N.T,N), axis=2)
```

Un des outils de concaténation est np.stack, on concatène N avec sa transposée (N.T), selon l'axe 2

Ajoutez une dimension à `points` pour que ce tableau soit de dimension (50,50,1,2).

```
points=points.reshape(50,50,1,2)
```

Pour redimensionner une matrice on utilise .reshape(*dimension désirée*)

Déclarez un tableau `t` contenant les valeurs -1,0,1

```
t=[-1,0,1]
```

Recopiez ce tableau 3 fois pour avoir une matrice `tmp` contenant `t` sur chaque ligne.

```
tmp=np.tile(t,(3,1))
tmp
```

```
array([[-1,  0,  1],
       [-1,  0,  1],
       [-1,  0,  1]])
```

On utilise à nouveau np.tile qui répète 3 fois t sur l'axe 1

Concaténez la transposé de `tmp` avec `tmp`. Vous mettrez le résultat dans la variable `v`.

```
v=np.stack((tmp.T,tmp), axis=2)
```

Redimensionez `v` pour qu'il soit de dimension (9,2).

```
v=v.reshape(9,2)
v
```

```
array([[-1, -1],
       [-1,  0],
       [-1,  1],
       [ 0, -1],
       [ 0,  0],
       [ 0,  1],
       [ 1, -1],
       [ 1,  0],
       [ 1,  1]])
```

On réitère le principe de concaténation et de redimensionnement.

Gardez dans v les valeurs tel que la première colonne est différente de la seconde ou dont la valeur dans la première colonne est différente de 0

```
v=v[(v[:,0]!=v[:,1])+(v[:,0]!=0)]
v
```

```
array([[-1, -1],
       [-1,  0],
       [-1,  1],
       [ 0, -1],
       [ 0,  1],
       [ 1, -1],
       [ 1,  0],
       [ 1,  1]])
```

v conserve sa valeur si la première colonne est différente de la seconde donc si :

v[:,0] !=v[:,1]

v[:,0] représente les valeurs de la première colonne

v[:,1] représente les valeurs de la seconde colonne

v conserve sa valeur si la valeur dans la première colonne est différente de 0 :

v[:,0] != 0

On obtient v[(v[:,0]!=v[:,1])+(v[:,0]!=0)], le « + » représente un « ou »

Ajoutez des dimensions à `v` pour que ce tableau soit de dimension (1,1,8,2).

```
v=v.reshape(1,1,8,2)
```

Sommez `points` et `v`. Vous mettrez le résultat dans le tableau `coord_voisins`.

```
coord_voisins=points+v
coord_voisins.shape
```

```
(50, 50, 8, 2)
```

On redimensionne la matrice **v** grâce a .reshape puis on la somme a **points**

Remplacez dans `coord_voisins` toutes les valeurs à 50 par 0. À votre avis, pourquoi cette opération est-elle nécessaire ?

```
coord_voisins[(coord_voisins[:,:]==50)]=0
```

coord_voisins stocke les coordonnées des voisins d'une cellule, hors la cellule 50 est hors champ(la grille contient 50 valeurs [0,…,49]) et doit donc être remise à 0

En utilisant grille et coord_voisins, comptez le nombre de cellule voisines vivantes pour chaque case de la grille. Pour rappel 0 indique l'absence de cellule dans la case et 1 une présence. Vous stockerez le résultat dans nbr_voisines.

nbr_voisines=

grille[coord_voisins[:,:,0,0],coord_voisins[:,:,0,1]]+grille[coord_voisins[:,:,1,0],coord_voisins[:,:,1,1]]+grille[coord_voisins[:,:,2,0],coord_voisins[:,:,2,1]]+grille[coord_voisins[:,:,3,0],coord_voisins[:,:,3,1]]+grille[coord_voisins[:,:,4,0],coord_voisins[:,:,4,1]]+grille[coord_voisins[:,:,5,0],coord_voisins[:,:,5,1]]+grille[coord_voisins[:,:,6,0],coord_voisins[:,:,6,1]]+grille[coord_voisins[:,:,7,0],coord_voisins[:,:,7,1]]

Nous cherches à stocker dans nbr_voisines le nombre de voisins vivant autour d'une cellule, exemple :

nbr_voisines[48,48]=3 voisins vivant autour de cette cellule

- La matrice coord_voisins contient toutes les coordonnées les valeurs autours d'une cellule.
- La grille contient 1 si une cellule est vivante et 0 si elle est morte.

Il faut donc regarder à chaque coordonnée des cellules voisines (coord_voisins), la valeur contenue dans grille, puis sommer la valeur de l'état de cette cellule (0 ou 1).

Pour illustrer, si l'on souhaite connaitre le nombre de voisins vivant de la cellule (1,1), cela reviens à la somme des états des cellules voisines de (1,1)

coord_voisins[1,1]	Sois :
Grille[[[0, 0],]+	grille[coord_voisins[:,:,0,0],coord_voisins[:,:,0,1]]+
Grille[[0, 1],]+	grille[coord_voisins[:,:,1,0],coord_voisins[:,:,1,1]]+
Grille[[0, 2],]+	grille[coord_voisins[:,:,2,0],coord_voisins[:,:,2,1]]+
Grille[[1, 0],]+	grille[coord_voisins[:,:,3,0],coord_voisins[:,:,3,1]]+
Grille[[1, 2],]+	grille[coord_voisins[:,:,4,0],coord_voisins[:,:,4,1]]+
Grille[[2, 0],]+	grille[coord_voisins[:,:,5,0],coord_voisins[:,:,5,1]]+
Grille[[2, 1],]+	grille[coord_voisins[:,:,6,0],coord_voisins[:,:,6,1]]+
Grille[[2, 2]])]	grille[coord_voisins[:,:,7,0],coord_voisins[:,:,7,1]]

Affichez les bloques 5x5 inférieur/droit du tableau `nbr_voisines`. Stochez le dans une variable nbr_proches1.

```
nbr_proches1=nbr_voisines[45:,45:]
print(nbr_proches1)
```

```
[[0. 0. 0. 0. 0.]
 [0. 0. 1. 1. 1.]
 [0. 0. 2. 2. 3.]
 [0. 0. 3. 3. 3.]
 [0. 0. 2. 2. 3.]]
```

On affiche toutes les valeurs à partir de la ligne/colonne : 50-5=45

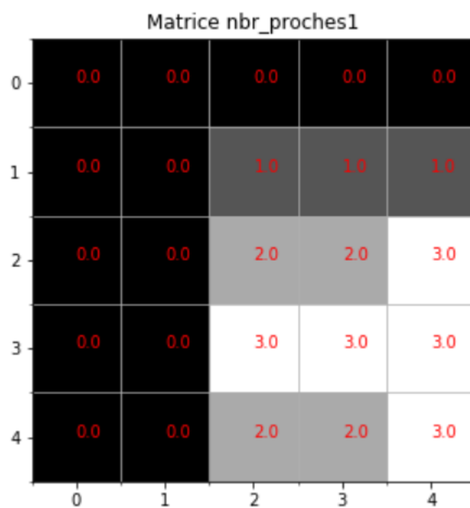
Affichez nbr_proches1 en image. Question Bonus: ajouter les valeurs de chaque case sur la figure.

```
for i in range(5):
    for j in range(5):
        ax.text(j, i, nbr_proches1[i,j] ,color='r')

plt.show()
```

On affiche **nbr_proches** avec la méthode précédente.

Pour afficher les valeurs sur chaque case on utilise une boucle qui parcourt les coordonnées de toute cellule, puis affiche la valeur de **nbr_proches1** a ces coordonnées.



3.2 - Calcul du nombre de voisine en utilisant un produit de convolution

Définissez un tableau `filtre` de taille 3x3 contenant que des 1 sauf en son centre. Le centre sera, quand à lui, à 0.

```
filtre=np.zeros((3,3))
filtre[:,:]=1
filtre[1,1]=0
```

En utilisant un produit de convolution entre `grille` et `filtre`, comptez le nombre de cellule voisines vivantes pour chaque case de la grille. Pour rappel 0 indique l'absence de cellule dans la case et 1 une présence. Vous stockerez le résultat dans `nbr_voisines2`.

```
nbr_voisines2=sc.signal.convolve2d(grille,filtre,'same')
nbr_voisines2
```

Sc.convolve2d(grille,filtre,'same') permet de faire une convolution entre deux matrices, ici grille et filtre.

Affichez les bloques 5x5 inférieur/droit du tableau `nbr_voisines2` et nommer le 'nbr_proches2'.Que remarquez vous?

```
nbr_proches2=nbr_voisines2[nbr_voisines2.shape[0]-5:nbr_voisines2.shape[0],nbr_voisines2.shape[1]-5:nbr_voisines2.shape[1]]
nbr_proches2
```

```
array([[0., 0., 0., 0., 0.],
       [0., 0., 1., 1., 1.],
       [0., 0., 2., 2., 3.],
       [0., 0., 3., 3., 3.],
       [0., 0., 2., 2., 3.]])
```

On veut afficher les 5 dernière ligne et colonnes du tableau :

Soit [nb lignes-5 : nb lignes , nb colonnes-5 : nb colonnes]

3.3 - Etapes du jeu de la vie

Définissez deux tableaux `vec2` et `vec3` qui sont de la même dimension que le tableau `nbr_voisines2`.

- Le tableau `vec2` contient `True` lorsque le tableau `nbr_voisines2` contient 2 et `False` sinon.
- De même `vec3` contient `True` lorsque `nbr_voisines2` contient 3 et `False` sinon.

```
vec2=(nbr_voisines2==2)
vec3=(nbr_voisines2==3)
```

vec2 VRAI si nbr_voisines2 == 2

vec3 VRAI si nbr_voisines2 == 3

Faites une multiplication terme à terme des tableaux (1-grille) et vec3. A votre avis, à quelle règle du jeu de la vie cette opération correspond ?

```
tt=(1-grille)*vec3
```

tt correspond à la règle « Une case vide entourée d'exactement trois cellules donne naissance à une nouvelle cellule »

Calculez l’expression suivante : `grille*(v2+v3)`. Expliquez ce que fait ce calcul et à quoi il peut bien servir

```
produit=grille*(vec2+vec3)
```

Ce calcul permet de mettre une cellule à 0 si ne cellule possédant strictement deux ou trois voisines vivantes, si elle respecte la condition elle reste vivante sinon elle meurt.

En utilisant les questions précédentes, calculer la première itération du jeu de la vie dont la position initiale est le tableau grille et afficher l'image correspondante.

```
egale3=(nbr_voisines==3)
sup3=(nbr_voisines>3)
inf2=(nbr_voisines<2)

grille[(egale3[:,:]==True)]=1
grille[(sup3[:,:]==True)]=0
grille[(inf2[:,:]==True)]=0
```

- Une case vide entourée d'exactement trois cellules donne naissance à une nouvelle cellule :

egale3 est VRAI si le nombre de voisins d'une cellule est 3

Donc pour toute les cellules de la **grille**, si **egale3** est vrai (si la cellule à 3 voisins vivants) alors elle est vivante (égale à 1) :

grille[(egale3[:,:]==True)]=1

- Une cellule ayant strictement plus de trois voisines ou strictement moins de deux voisines sera morte à l'itération suivante

sup3 est VRAI si le nombre de voisins d'une cellule est supérieur à 3

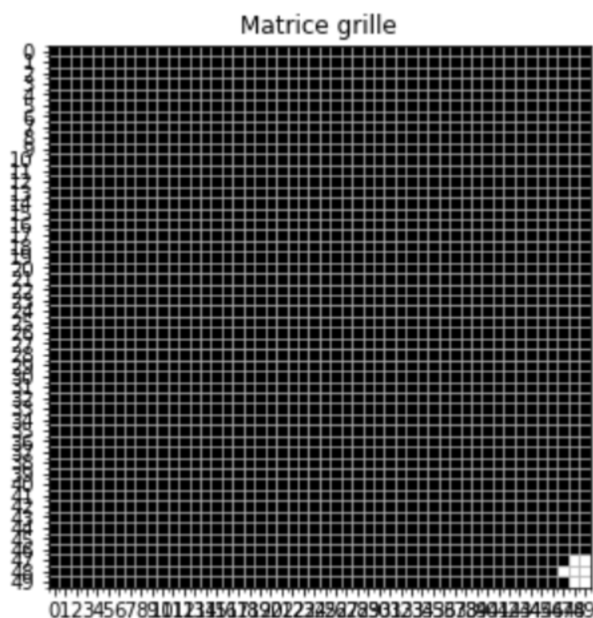
inf2 est VRAI si le nombre de voisins d'une cellule est infèrieur à 2

Donc pour toute les cellules de la **grille**, si **sup3** ou **in2** est vrai (si la cellule àplus de trois voisines ou moins de deux voisines) alors elle meurt (égale à 0) :

grille[(sup3[:,:]==True)]=0

grille[(inf2[:,:]==True)]=0

<u>On obtient :</u>



Déduire une fonction jeu_de_la_vie(grille,n) calculant et retournant les n premières itérations du jeu de la vie dont la position initiale est le tableau grille. Vous avez le droit d'utiliser une boucle for pour cette question

```python
def jeu_de_la_vie(grille,n):

    #Blocs optionnel
    grille=np.zeros((50,50))
    debut=np.zeros((3,3))
    debut[0:3,1]=1
    debut[1,2]=1
    grille[47:50,47:50]=debut
    #---------------
    filtre=np.zeros((3,3))
    filtre[:,:]=1
    filtre[1,1]=0

    for i in range(n):

        nbr_voisin=sc.signal.convolve2d(grille,filtre,'same')

        egale3=(nbr_voisin==3)

        sup3=(nbr_voisin>3)
        inf2=(nbr_voisin<2)

        grille[(egale3[:,:]==True)]=1
        grille[(sup3[:,:]==True)]=0
        grille[(inf2[:,:]==True)]=0

    return grille
```

Le Bloc optionnel permet de remettre la grille au cas 1. Si l'on souhaite utiliser la fonction avec une grille passée en argument, il suffit de mettre en commentaire ce Bloc.

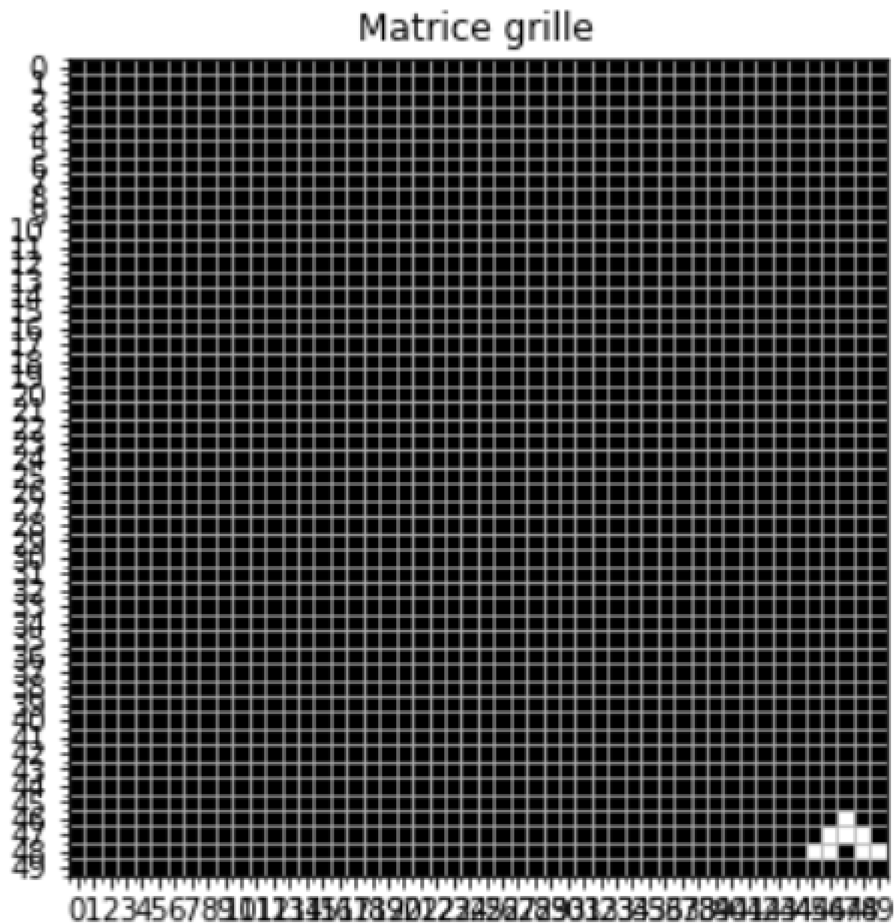
On commence par initialisé le filtre hors de la boucle afin qu'il ne soit pas modifier.

Dans la boucle répétée n fois, on recalcule **nbr_voisins** pour avoir le nombre de voisins après chaque modification de la grille.

Puis on fait les tests vus précédemment pour modifier l'état des cellules.

On retourne la **grille**

Tester la fonction précédente pour n=10. Et Affichez l'image correspondante de la grille.



Question Bonus:Créez une animation vidéo montrant 100 itérations du jeu de la vie en partant de `grille`.

```
import matplotlib.animation as animation

grille=np.zeros((50,50))
debut=np.zeros((3,3))
debut[0:3,1]=1
debut[1,2]=1
grille[47:50,47:50]=debut

filtre=np.zeros((3,3))
filtre[:,:]=1
filtre[1,1]=0

fig = plt.figure()
def updatefig(i):

    nbr_voisin=sc.signal.convolve2d(grille,filtre,'same')

    egale3=(nbr_voisin==3)

    sup3=(nbr_voisin>3)
    inf2=(nbr_voisin<2)

    grille[(egale3[:,:]==True)]=1
    grille[(sup3[:,:]==True)]=0
    grille[(inf2[:,:]==True)]=0

    # Affichage de la grille
    plt.figure(figsize=(15,15))
    plt.imshow(grille,cmap='gray')
    plt.title('Matrice grille')

    ax = plt.gca()
    ax.set_xticks(np.arange(0, 50))
    ax.set_yticks(np.arange(0, 50))
    ax.set_xticks(np.arange(-.5, 50, 1), minor=True);
    ax.set_yticks(np.arange(-.5, 50, 1), minor=True);
    ax.set_xticklabels(np.arange(50))
    ax.set_yticklabels(np.arange(50))
    plt.grid(which='minor')
    plt.show()

anim = animation.FuncAnimation(fig, updatefig, 20)
anim.save("grille.mp4", fps=5)
```

Pour créer une vidéo, on fait exactement comme la fonction **jeu_de_la_vie.**

Tout d'abord on définit les variables fixes hors de la fonction. **fig** correspondant à la figure créée.

On définit la fonction updatefig(i) avec i nombre de répétition, on y place le contenu de la boucle for de la fonction **jeu_de_la_vie.**

Pour créer la vidéo, il suffit d'utiliser **animation.FuncAnimation** à laquelle on passe en argument **fig**, la fonction **updatefig** et le nombre de répétitions **i**. On la sauvegarde grave à .save.