HMMA238_TP_elucsonjeanbaptiste

March 13, 2020

[149]:
```
# Numéro 1 )
# Chaîne de caractères filename correspondant à mon nom de mon fichier sous le␣
 ↪format suivant :filename=HMMA238_TP_prenomnom.ipynb
filename="HMMA238_TP_elucsonjeanbaptiste.ipynb"
print('Le nom de mon fichier est ' + filename)
```

```
Le nom de mon fichier est HMMA238_TP_elucsonjeanbaptiste.ipynb
```

[150]:
```
# Numéro 2)
# La création d'une variablede taille_str qui compte le nombre de caractères␣
 ↪dans la chaîne de caractères filename

# L'utilisation de la fonction len va nous donner la longueur de la chaîne
taille_str = len(filename)
print(taille_str)
```

```
36
```

[151]:
```
# Numéro 3
# 3)La créaction d'une variable ma_graine qui vaut le reste de la division␣
 ↪euclidienne de taille_str par 6
ma_graine =(taille_str,6)
r = taille_str%6 # Le reste de la division euclidienne
q = taille_str//6 # Le quotient de la division euclidienne
print(r,q)
```

```
0 6
```

[14]:
```
# Implementation sans numpy
# Question 4, a)
def calcul_nb_voisins(z):
    forme = len(z), len(z[0]) # On identifie le nombre de ligne et de colonne
    N = [[0,] * (forme[0]) for i in range(forme[1])]
    for x in range(1, forme[0] - 1):
        for y in range(1, forme[1] - 1):
```

```
            N[x][y]= z[x-1][y-1]+z[x][y-1]+z[x+1][y-1]\
            +z[x-1][y]+0                  +z[x+1][y]\
            +z[x-1][y+1]+z[x][y+1]+z[x+1][y+1]
    return N
Z = [[0,0,0,0,0,0],
     [0,0,0,1,0,0],
     [0,1,0,1,0,0],
     [0,0,1,1,0,0],
     [0,0,0,0,0,0],
     [0,0,0,0,0,0]]
calcul_nb_voisins(Z)

# La sortie obtenue en posant N=calcul_nb_voisins(Z)montre qu'on a autour de␣
 ↪l'univers du jeu des cellules mortes.
```

[14]:
```
[[0, 0, 0, 0, 0, 0],
 [0, 1, 3, 1, 2, 0],
 [0, 1, 5, 3, 3, 0],
 [0, 2, 3, 2, 2, 0],
 [0, 1, 2, 2, 1, 0],
 [0, 0, 0, 0, 0, 0]]
```

[1]:
```
# numéro 4, b)
def iteration_jeu(z):
    forme=len(z), len(z[0])
    """
        N=calcul_nb_voisin(z) # On met une docsting
    """
    for x in range(1, forme[0]-1):
        for y in range[1, forme[1]-1]:
            if z[x][y]==1 and (N[x][y]<2 or N[x][y]>3):
                z[x][y]= 0
            elif z[x][y]==0 and N[x][y] ==3:
                z[x][y] =1
    return Z
```

[ ]:
```
# 5
```

[1]:
```
# numéro 7) Implementation avec numpy
import numpy as np
# Définissons le vecteur vec comme suit :
vect = np.array([0, 1, 0, 0, 1, 1]) # Cet array est de taille (1,6)
nb_vect = np.zeros(vect.shape) # On initialise pour avoir la meme taille pour␣
 ↪vecteur nb_vect
nb_vect[1:-1] += (vect[:-2] + vect[2:])
nb_vect
# Le vecteur nb_vect correspond aux nombres de voisins que chaque élément du␣
 ↪vecteur vect contient.
```

[1]:
```
array([0., 0., 1., 1., 1., 0.])
```

[171]:
```
# numéro 9) La création d'une fonction iteration_jeu,similaire à iteration_jeu␣
 ↪mais qui prend comme entrée sortie des numpy array et non plus des listes.

def iteration_jeu_np(Z):

    """
        La création d'une fonction qui fait le calcul d'une itération du jeu de␣
 ↪la vie à partir d'un slicing

        En entrée on a une liste de listes qui est composée de 1 ou de 0

        Pour la sortie on a une ne itération du jeu
    """

    forme = len(Z), len(Z[0])
    Z = np.array(Z) # On effectue les calculs avec les arrays
    N = calcul_nb_voisins_np(Z)
    for x in range(1, forme[0]-1):
        for y in range(1, forme[1]-1):
            if Z[x][y] == 1 and (N[x][y] < 2 or N[x][y] > 3):
                Z[x][y] = 0
            elif Z[x][y] == 0 and N[x][y] == 3:
                Z[x][y] = 1
    return Z
```

[174]:
```
# numéro 10))Fonction qui donne n itérations du jeu

def jeu_np(Z_in, nb_iter):

    """
       La fonction qui présente la dernière itérationn du jeu de la vie

       En entrée on a une matrice Z_in qui est composée de 0 et et de 1
            On a un entier non nul positif d'itérations nb_iter

        Pour la sortie on a une idée sur le jeu après nb_iter itérations
    """

    for i in range(1, nb_iter+1):
        Z_in = iteration_jeu_np(Z_in)
    return(Z_in)

Z = [[0, 0, 0, 0, 0, 0],
     [0, 0, 0, 1, 0, 0],
     [0, 1, 0, 1, 0, 0],
     [0, 0, 1, 1, 0, 0],
```

```
     [0, 0, 0, 0, 0, 0],
     [0, 0, 0, 0, 0, 0]]

jeu_np(Z, 10)


# Ontrouve toujours la meme réponse que ceux des exercices 4 et 8.
```

[174]:
```
array([[0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0],
       [0, 0, 0, 1, 1, 0],
       [0, 0, 0, 1, 1, 0],
       [0, 0, 0, 0, 0, 0]])
```

[ ]:
```
# numéro 11)
```

[ ]:
```

```