TP image - Photomaton

1. Traitement d'image en Python

1.1. Travail préliminaire

```python
import numpy as np
import matplotlib.pyplot as plt

picasso = plt.imread('picasso.png')
matisse = plt.imread('matisse.png')

#pour afficher les dimensions du tableau :
dimension = picasso.shape    #renvoie un tuplet (nb_ligne, nb_col, nb_pixels)

#pour créer un tableau de mêmes dimensions mais rempli de 0
image =np.zeros_like(picasso)

plt.imshow(picasso)
plt.show()
```

1.2. Transformation par symétrie

5. On doit parcourir chaque pixel du tableau pour en modifier le contenu (3 informations de couleurs). La fonction aura donc 2 boucles `for` imbriquées (complexité quadratique). Il faut vérifier que l'on va bien parcourir **tout** le tableau et aussi éviter de "sortir" du tableau (erreur de type `out of range`). Pour cela, il est indispensable de tester "à la main" les indices extrêmes des boucle `for`.

```python
def symetrie(img):
    """ prend en argument une matrice-image img
    renvoie la matrice-image résultat de la symétrie d'axe vertical """
    new_img =np.zeros_like(img)
    N_lignes =img.shape[0]
    N_colonnes =img.shape[1]
    for x in range(N_lignes):    # x prend les valeurs entières entre 0 et N_lignes-1 (inclues)
        for y in range(N_colonnes): # y prend les valeurs entières entre 0 et N_colonnes-1 (inclues)
            new_img[x,N_colonnes-1-y] =img[x,y]      # Vérification pour x=0, y=0
                                                     # et pour x=N_lignes-1, y=N_colonnes-1
    return new_img

picasso_sym =symetrie(picasso)
```

Python
```
plt.imshow(picasso_sym)
plt.show()
```

7. On s'attend à avoir une fonction d'ordre 2.

Python
```
picasso_retour = symetrie(symetrie(picasso))
plt.imshow(picasso_retour)
plt.show()
```

1.3. Rotation d'un quart de tour

8.

Python
```
def rotationQT(img):
    """prend en argument une matrice-image
    renvoie la matrice-image résultat de la rotation d'un quart de tour"""
    N_lignes = img.shape[0]
    N_colonnes = img.shape[1]
    N_pixels = img.shape[2]
    new_img = np.zeros((N_colonnes, N_lignes, N_pixles))
    for x in range(N_lignes):
        for y in range(N_colonnes):
            new_img[y,N_lignes-1-x] = img[x,y]
    return new_img

picasso_QT = rotationQT(picasso)
plt.imshow(picasso_QT)
plt.show()
```

2. Ordre d'une transformation

9. Pour cette fonction, on en connaît pas le nombre d'itérations. Une boucle while s'impose.

Python
```
def ordre(fct,img):
    """prend en argument une fonction bijective fct et une matrice-image img
    renvoie l'ordre de transformation de la fonction fct (integer)"""
    ordre =1
    new_img =fct(img)
    while (new_img==img).all() !=True:
        new_img =fct(new_img)
        ordre +=1
    return ordre
```

3. Transformation du photomaton

13.

Python
```
def coord_photo(x,N):
    """prend en arguments une coordonnée d'un pixel (abscisse ou ordonnée)
    ainsi que le nombre de lignes ou de colonnes.
    renvoie la nouvelle position de ce pixel après la transformation du photomaton"""
    if x%2 ==0:
        return x/2
    else:
        return N/2 +x//2


def photomaton(img):
    """prend en argument une matrice-image img
    renvoie une nouvelle matrice_image obtenue par la transformation du photomaton."""
    new_img =np.zeros_like(img)
    N_lignes =img.shape[0]
    N_colonnes =img.shape[1]
    for x in range(N_lignes):
        for y in range(N_colonnes):
                new_img[coord_photo(x,N_lignes),coord_photo(y,N_colonnes)] =img[x,y]
    return new_img
```

15.

Python
```
def periode_photomaton(img):
    """prend en argument une matrice-image
    renvoie la période de la transformation du photomaton"""
    p,q =img.shape[0],img.shape[1]
    n =1
    while (2**n-1)%(p-1) !=0 or (2**n-1)%(q-1) !=0:
        n +=1
    return n
```

16.

Python
```
def photomaton2(img,n):
    """prenden arguments une image img et un entier n
    retourne la nième itérée de l'image"""
    new_img1 =np.copy(img)
    new_img2 =np.zeros_like(img)
    N_lignes =img.shape[0]
    N_colonnes =img.shape[1]
    for k in range(n):
        for x in range(N_lignes):
            for y in range(N_colonnes):
                    new_img2[coord_photo(x,N_lignes),coord_photo(y,N_colonnes)] =new_img1[x,y]
        new_img1 =np.copy(new_img2)
    return new_img2
```

4. Transformation du boulanger

18.

Python
```
def coord_boulanger(x,y,N_lignes,N_colonnes):
    """prend en arguments les coordonnées d'un pixel (x,y)
    renvoie les coordonnées de ce pixel après la transformation du boulanger"""
    if x%2 ==0:
        x1 =x/2
        y1 =2*y
    else:
        x1 =x//2
        y1 =2*y +1
    if y1 <N_colonnes:
        x2 =x1
        y2 =y1
    else:
        x2 =N_lignes - 1 - x1
        y2 =N_colonnes - 1 -y1
    return x2,y2
```

Python

```
def boulanger(img):
    """prend en argument une image.
    retourn la transformée du boulanger de celle-ci."""
    new_img =np.zeros_like(img)
    N_lignes =img.shape[0]
    N_colonnes =img.shape[1]
    for x in range(N_lignes):
        for y in range(N_colonnes):
                new_img[coord_boulanger(x,y,N_lignes,N_colonnes)] =img[x,y]
    return new_img
```