

```
#####
#          TP0 - Révisions de lière année (traitement d'image et graphes)          #
#####

import matplotlib.pyplot as plt
import numpy as np

# Première partie
#####

# Question 1 :
#####
image_c = plt.imread("Chiffres_Couleur.png")

def convertir_image(img:np.ndarray) -> np.ndarray :
    img_g = np.zeros((len(img),len(img[0])))
    for i in range(len(img_g)):
        for j in range(len(img_g[0])):
            niveau_gris = 0.2126*img[i,j,0]+0.7152*img[i,j,1]+0.0722*img[i,j,2]
            img_g[i,j] = np.float32(niveau_gris)
    return img_g

image_g= convertir_image(image_c)

def affiche(img):
    # Fonction pour afficher l'image
    plt.close()                # On ferme les images ouvertes
    plt.imshow(img,cmap='gray',vmin=0.0,vmax=1.0)    # On trace l'image en niveau de gris
    plt.show()                 # On montre l'image

# Question 2 :
#####

def dim(img:np.ndarray) -> tuple :
    nl,nc=len(img),len(img[0])
    return nl,nc

# Autre solution

# def dim(img:np.ndarray) -> tuple :
#     nl,nc=np.shape(img)      # La fonction numpy "shape" renvoi les dimensions du tableau
#     return nl,nc

# Question 3 :
#####

def img_blanche(nl:int,nc:int) -> np.ndarray :
    liste = [[1.]*nc for i in range(nl)]    # Crée une liste de nl listes de nc 1. (flottant)
    return np.array(liste)                  # On renvoi le tableau numpy de cette liste

# Autre solution

# def img_blanche(nl:int,nc:int) -> np.ndarray :
#     return np.ones((nl,nc))               # Fonction numpy pour renvoyer un tableau de 1
```

Question 4 :

#####

```
def convertir_N_B(img:np.ndarray,seuil:float) -> np.array :
    nl,nc=dim(img)          # On mémorise les dimensions de l'image originale
    img_nb=img_blanche(nl,nc) # On crée une image blanche de même dimensions (tableau numpy)
    for i in range(nl):      # On parcourt toutes les lignes de l'image
        for j in range(nc):  # On parcourt tous les pixels de la ligne
            if img[i,j] < seuil: # Si le niveau de gris du pixel est inférieure au seuil
                img_nb[i,j]= 0.0 # alors le pixel de l'image noir ou blanc sera noir
            else:              # Sinon
                img_nb[i,j]= 1.0 # le pixel de l'image noir ou blanc sera Blanc
    return img_nb             # On retourne l'image noir ou blanc
```

Autre solution

```
# def convertir_N_B(img:np.ndarray,seuil) -> np.array :
#     nl,nc=dim(img)          # On mémorise les dimensions de l'image originale
#     img_nb=[]               # On initialise l'image noir ou blanc avec une liste vide
#     for i in range(nl):     # On parcourt toutes les lignes de l'image originale
#         ligne=[]            # On initialise la liste des pixels de la ligne avec une liste vide
#         for j in range(nc): # On parcourt tous les pixels de la ligne de l'image originale
#             if img[i,j] < seuil: # Si le niveau de gris du pixel de l'originale est inférieure au
seuil
#                 ligne.append(0.0) # alors on ajoute un pixel Noir à la liste ligne
#             else:              # Sinon
#                 ligne.append(1.0) # on ajoute un pixel Blanc à la liste ligne
#         img_nb.append(ligne)   # On ajoute la ligne à l'image noir ou blanc
#     return np.array(img_nb)   # On retourne le tableau numpy de l'image noir ou blanc
```

Question 5 :

#####

```
def sym_horiz(img:np.ndarray) -> np.array :
    nl,nc=dim(img)          # On mémorise les dimensions de l'image originale
    img_sym=img_blanche(nl,nc) # On crée une image blanche de même dimensions (tableau numpy)
    for i in range(nl):      # On parcourt toutes les lignes de l'image blanche
        img_sym[i]=img[nl-1-i] # On lui donne les couleurs de la ligne symétrique de
l'original
    return img_sym           # On retourne l'image symétrique
```

Autre solution

```
# def sym_horiz(img:np.ndarray) -> np.array :
#     nl,nc=dim(img)          # On mémorise les dimensions de l'image originale
#     img_sym=[img[nl-1-i] for i in range(nl)] # On crée une liste de nl tableau (1*nc) qui sont
#                                             # les lignes symétriques de l'image originale
#     return np.array(img_sym) # On retourne le tableau numpy de l'image symétrique
```

Question 6 :

#####

```
def sym_verti(img:np.ndarray) -> np.array :
    nl,nc=dim(img)          # On mémorise les dimensions de l'image originale
    img_sym=img_blanche(nl,nc) # On crée une image blanche de même dimensions (tableau numpy)
    for i in range(nl):      # On parcourt toutes les lignes de l'image blanche
        for j in range(nc):  # On parcourt tous les pixels de la ligne
            img_sym[i,j]=img[i,nc-1-j] # On lui donne la couleur du pixel symétrique de l'original
    return img_sym           # On retourne l'image symétrique
```

Autre solution

```
# def sym_verti(img:np.ndarray) -> np.array :
#     nl,nc=dim(img)                # On mémorise les dimensions de l'image originale
#     img_sym=[]                    # On crée une liste vide
#     for i in range(nl):           # On parcourt toutes les lignes de l'image originale
#         img_sym.append([img[nc-1-i] for j in range(nc)]) # On ajoute à la liste une ligne symétrique
#                                     # de la ligne de l'image originale
#     return np.array(img_sym)       # On retourne le tableau numpy de l'image symétrique
```

Question 7 :

#####

```
def rot_gauche(img:np.ndarray) -> np.array :
    nl,nc=dim(img)                # On mémorise les dimensions de l'image originale
    img_tour=img_blanche(nc,nl)    # On crée une image blanche de dimensions inversées (tableau
numpy)
    for i in range(nc):           # On parcourt toutes les lignes de l'image tournée
        for j in range(nl):       # On parcourt tous les pixels de la ligne d el'iamge tournée
            img_tour[i,j]=img[j,nc-1-i] # On lui donne la couleur du pixel de l'image originale
    return img_tour               # On retourne l'image symétrique
```

Autre solution

```
# def rot_gauche(img:np.ndarray) -> np.array :
#     nl,nc=dim(img)                # On mémorise les dimensions de l'image originale
#     img_tour=[]                    # On crée une liste vide
#     for i in range(nc):           # On parcourt toutes les colonnes de l'image originale
#         img_tour.append([img[j,nc-1-i] for j in range(nl)]) # On ajoute à la liste une ligne
#                                     # correspondant à la colonne de
#                                     # l'image originale
#     return np.array(img_tour)     # On retourne le tableau numpy de l'image tournée
```

Question 8 :

#####

```
def rot_180(img:np.ndarray) -> np.array :
    return rot_gauche(rot_gauche(img)) # Deux rotations de +90° <=> Une rotation de 180°
```

Autre solution

```
# def rot_180(img:np.ndarray) -> np.array :
#     return sym_verti(sym_horiz(img)) # 2 symétries axiales <=> Une rotation d'angle 2*alpha
#                                     # où alpha est l'angle entre les deux axes de symétrie
```

Question 9 :

#####

```
def rot_droite(img:np.ndarray) -> np.array :
    for k in range(3):
        img=rot_gauche(img)        # Trois rotations de +90° <=> Une rotation de -90°
    return img
```

Autre solution

```
# def rot_180°(img:np.ndarray) -> np.array :
#     return rot_gauche(rot_180(img))
```

```

# Deuxième partie
#####

exemple_c=plt.imread("Exemple.png")
exemple_g=convertir_image(exemple_c)
exemple=convertir_N_B(exemple_g,0.7)

# Question 10 :
#####

def graphe_vide(nl:int,nc:int)->np.ndarray :
    graph = [[None]*nc for i in range(nl)] # Crée 1 liste de nl listes de nc None
    return np.array(graph)                # Renvoie le tableau Numpy de cette liste

# Question 11 :
#####

def liste_voisins(imgnb:np.array,l:int,c:int) -> list :
    nl,nc=dim(imgnb)                       # On mémorise les dimension de l'image
    liste=[]                               # On initialise la liste des voisins
    for i in range(max(0,l-1),min(nl,l+2)): # On parcourt toutes les lignes voisines
        for j in range(max(0,c-1),min(nc,c+2)): # On parcourt tous les pixels voisins de la ligne
            if imgnb[i,j]==0 and (i,j)!=(l,c): # Si le pixel voisin est noir et pas (l,c)
                liste.append((i,j)) # On ajoute le pixel à la liste des voisins
    return liste                           # On retourne la liste des voisins

# Question 12 :
#####

def creer_graphe(img:np.ndarray,seuil:float)->np.ndarray :
    nl,nc = dim(img)                       # On mémorise les dimension de l'image
    imgnb = convertir_N_B(img,seuil) # On passe l'image du niveau de gris au Noir ou Blanc
    graph = graphe_vide(nl,nc)             # On initialise un graphe vide aux dimensions de l'image
    for i in range(nl):                    # On parcourt toutes les lignes de l'image
        for j in range(nc):                # On parcourt tous les pixels de la ligne
            if imgnb[i,j] == 0 :           # Si le pixel est un pixel noir
                list_v=liste_voisins(imgnb,i,j)
                graph[i,j] = (None,list_v) # On ajoute le couple 'None' et
                                           # la liste des voisins noirs du pixel
    return graph                           # On retourne le graphe

# Question 13 :
#####

def indicage_pixels(graph:np.ndarray,l:int,c:int,ind:int)->None:
    indice,List_v = graph[l,c]             # initialisation des variables indice et List_v
    if indice == None :                     # Si le pixel n'est pas indicé (indice = None)
        graph[l,c] = (ind,List_v)          # On indice le pixel en modifiant le couple du graphe
    for pixel in List_v :                   # On parcourt tous les voisins (noirs)
        indicage_pixels(graph,pixel[0],pixel[1],ind) # On change aussi l'indice
                                                    # du voisin

```

Question 14 :

#####

```
def indicage_graph(graph:np.ndarray)->int:
    indice = 0                                # On initialise l'indice à 0
    nl,nc=dim(graph)                          # On mémorise les dimensions du graph
    for i in range(nl):                       # On parcourt toutes les lignes du graph
        for j in range(nc):                  # On parcourt tous les pixels de la ligne
            if graph[i,j]!=None and graph[i,j][0]==None:  # Si le pixel du graph est noir
                                                                # et qu'il n'est pas indicé
                indicage_pixels(graph,i,j,indice)          # Le pixel est indicé ainsi que
                                                                # ceux de la composante connexe
            indice += 1                        # On incrémente l'indice pour la composante connexe suivante
    return indice                            # On retourne "indice" qui est le nombre d'indices du graphe
```

Question 15 :

#####

```
def trace_chiffre(graph:np.ndarray,ind:int)->np.ndarray:
    nl,nc=dim(graph)                          # On mémorise les dimensions de l'image (ou du graphe)
    image_chiffre=img_blanche(nl,nc)          # On initialise une image blanche
    for i in range(nl):                       # On parcourt toutes les lignes de l'image
        for j in range(nc):                  # On parcourt tous les pixels de la ligne
            if graph[i,j]!= None and graph[i,j][0]==ind:  # Si le pixel du graphe est noir
                                                                # et a le même indice que "ind"
                image_chiffre[i,j]=0.0        # On noircit le pixel de l'image blanche
    return image_chiffre                     # On retourne l'image
```

Question 16 :

#####

```
def animation_chiffres(img:np.ndarray,seuil:float)->None:
    nl,nc=dim(img)                            # On mémorise les dimensions de l'image
    graph=creer_graph(img,seuil)              # On crée le graphe de l'image
    Nbr_indices=indicage_graph(graph)         # On l'indice et on mémorise le nombre d'indices
    for i in range(Nbr_indices):              # On parcourt tous les indices (tous les chiffres)
        diapo=trace_chiffre(graph,i)          # On crée l'image du chiffre
        affiche(diapo)                        # On affiche cette image
        plt.pause(0.3)                        # pendant 0,3 secondes
```

Question 17 :

#####

```
def animation_zoom(img:np.ndarray,seuil:float)->None:
    nl,nc=dim(img)                            # On mémorise les dimensions de l'image
    graph=creer_graph(img,seuil)              # On crée le graphe de l'image
    Nbr_indices=indicage_graph(graph)         # On l'indice et on mémorise le nombre d'indices
    for i in range(Nbr_indices):              # On parcourt tous les indices (tous les chiffres)
        diapo=trace_chiffre(graph,i)          # On crée l'image du chiffre
        l_min,l_max=nl-1,0                    # On initialise les limites du chiffres lignes
        c_min,c_max=nc-1,0                    # et colonnes (mini=maxi et maxi=mini)
```

```
for k in range(nl):          # On parcourt toutes les lignes de l'image
    for l in range(nc):      # On parcourt tous les pixels de l'image
        if diapo[k,l]==0.0:  # Si les pixels sont noirs
            if l_min>k : l_min=k  # On ajuste la limite supérieure
            if l_max<k : l_max=k  # On ajuste la limite inférieure
            if c_min>l : c_min=l  # On ajuste la limite gauche
            if c_max<l : c_max=l  # On ajuste la limite droite
diapo_zoom=img_blanche(l_max+1-l_min,c_max+1-c_min) # On crée une image blanche aux
                                                    # dimensions du chiffre (petite)
for k in range(l_max+1-l_min):          # On parcourt tous les pixels de la
    for l in range(c_max+1-c_min):      # grande image dans les limites
        diapo_zoom[k,l]=diapo[k+l_min,l+c_min] # On noirci les pixels correspondant
                                                    # dans la petite image
affiche(diapo_zoom)                    # On affiche cette image
plt.pause(0.3)                          # pendant 0,3 secondes
```