

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
```

```
__author__ = "Xavier Pessoles"
```

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import math
```

```
img = mpimg.imread("PhotoBD.png")
```

```
def affichage1(img1):
    """ Affichage d'une seule image"""
    plt.imshow(img1)
    plt.show()
```

```
def affichage2(img1, img2):
    """ Affichage de deux images"""
    f, ((ax1, ax2)) = plt.subplots(1, 2, sharex='col', sharey='row')
    ax1.imshow(img1)
    ax2.imshow(img2)
    plt.show()
```

```
def affichage3(img1, img2, img3):
    """ Affichage de trois images"""
    f, ((ax1, ax2, ax3)) = plt.subplots(1, 3, sharex='col', sharey='row')
    ax1.imshow(img1)
    ax2.imshow(img2)
    ax3.imshow(img3)

    plt.show()
```

```
## Question 1 : affichage de l'image
# plt.imshow(img)
# plt.show()
```

```
## Question 2 :
(h,l,p) = img.shape # Taille de l'image
print("Couleur du pixel 200x200 : ", img[300,300,:])
```

```
print("Niveau de rouge mini et maxi : ", img[:, :, 0].min(), "et", img[:, :, 0].max())
print("Niveau de vert mini et maxi : ", img[:, :, 1].min(), "et", img[:, :, 1].max())
print("Niveau de bleu mini et maxi : ", img[:, :, 2].min(), "et", img[:, :, 2].max())
```

```
## Question 3 : [0,1] -> [0,255] et [0,255] -> [0,1]
```

```
def conv_1_2_255(img):
    """
    Conversion des niveaux de couleurs de [0,1] -> [0,255]
    * Entrée :
    - img(numpy.ndarray) : image de shape (h,l,3)
    * Sortie :
    - img(numpy.ndarray) : image de shape (h,l,3)
    """
    a = 255*img
    a = a.astype(int)
    return a
```

```
def conv_255_2_1(img):
    """
    Conversion des niveaux de couleurs de [0,255] -> [0,1]
    * Entrée :
    - img(numpy.ndarray) : image de shape (h,l,3)
    * Sortie :
```

```

- img(numpy.ndarray) : image de shape (h,l,3)
"""
return (1/255)*img

## Question 4 : Histogramme des couleurs

def histogramme(img):
    """ Fonction permettant de tracer l'histogramme de chacune des couleurs
    * Entrée :
    - img(numpy.ndarray) : image de shape (h,l,3) (Codes couleurs comprises entre 0 et 1)
    """

    img = conv_1_2_255(img)
    histo = [[0 for j in range(256)] for j in range(3)]
    (l,h,p)=img.shape
    for i in range(l):
        for j in range(h):
            for k in range(3):
                histo[k][img[i,j,k]]+=1

    plt.plot(histo[0], "r")
    plt.plot(histo[1], "g")
    plt.plot(histo[2], "b")
    plt.show()
    return histo
# histogramme(img)

## Question 5 Trame des couleurs
imgr=np.zeros(img.shape)
imgr[:, :, 0]=img[:, :, 0]

imgv=np.zeros(img.shape)
imgv[:, :, 1]=img[:, :, 1]

imgb=np.zeros(img.shape)
imgb[:, :, 2]=img[:, :, 2]

#affichage3(imgr,imgv,imgb)

## Question 6 : Images saturées

img_sr = img.copy()
img_sr[:, :, 0]=1

img_sv = img.copy()
img_sv[:, :, 1]=1

img_sb = img.copy()
img_sb[:, :, 2]=1
#affichage3(img_sr,img_sv,img_sb)

## Question 7 : Images inversées

# img[:, :, :]=1-img[:, :, :]
img_inv= 1-img

#affichage2(img,img_inv)

## Question 8 : Transformation en noir et blanc
def convert_nb_moy(img,a,b,c):
    """
    Conversion d'une image en noir et blanc par pondération de chaque niveau de couleur
    """
    imgnb = np.zeros(img.shape)
    imgnb[:, :, 0]=(a/(a+b+c)*img[:, :, 0]+b/(a+b+c)*img[:, :, 1]+c/(a+b+c)*img[:, :, 2])* (1/(a+b+c))
    imgnb[:, :, 1]=imgnb[:, :, 0]
    imgnb[:, :, 2]=imgnb[:, :, 0]

```

```

return imgnb

imgnb1 = convert_nb_moy(img,1,1,1)
imgnb2 = convert_nb_moy(img,10,1,1)
imgnb3 = convert_nb_moy(img,1,2,1)
imgnb4 = convert_nb_moy(img,10,5,1)

"""
h1 = histogramme(imgnb1)
h2 = histogramme(imgnb2)
h3 = histogramme(imgnb3)
h4 = histogramme(imgnb4)

plt.plot(h1[0])
plt.plot(h2[0])
plt.plot(h3[0])
plt.plot(h4[0])
"""

#affichage3(img,imgnb1,imgnb2)
#affichage3(img,imgnb3,imgnb4)

## Question 9 : Transformation en noir et blanc

def convert_nb_2(img):
    imgnb = np.zeros(img.shape)
    (h,l,p) = img.shape
    for i in range(h):
        for j in range(l):
            for k in range(p):
                imgnb[i,j,k]=0.5*(max(img[i,j,:])+max(img[i,j,:]))
    return imgnb

# affichage2(img,convert_nb_2(img))

## Question 10 tracer des fonctions
les_x = np.linspace(0,1,1000)

def f2(x):
    return 0.5+0.5*math.sin(math.pi*(x-0.5))

vf2 = np.vectorize(f2)

"""
plt.plot(les_x,np.sqrt(les_x))
plt.plot(les_x,vf2(les_x))
plt.show()
"""

## Question 11 tracer des fonctions

def contraste(img,f):
    imgc = np.zeros(img.shape)
    (h,l,p) = img.shape
    for i in range(h):
        for j in range(l):
            for k in range(p):
                imgc[i,j,k]=f(img[i,j,k])

    return imgc

imgnb = convert_nb_moy(img,1/3,1/3,1/3)
img1 = contraste(img,math.sqrt)
img2 = contraste(img,f2)

#affichage3(img,img1,img2)

## Question 12

```

```
def convolution(img,M):
    imgc = np.zeros(img.shape)
    (h,l,p) = img.shape
    for i in range(1,h-1):
        for j in range(1,l-1):
            for k in range(p):
                imgc[i,j,k]=np.sum(M*img [i-1:i+2,j-1:j+2,k])
    return imgc

img_255 = conv_1_2_255(img)
M1 = [[1/8,1/8,1/8],[1/8,-1/8,1/8],[1/8,1/8,1/8]]
M2 = [[1,1,1],[1,-2,1],[1,1,1]]
img_cv1 = convolution(img_255,M1)
img_cv2 = convolution(img_255,M2)

img_11 = conv_255_2_1(img_cv1)
img_12 = conv_255_2_1(img_cv2)

#M=np.array(M)
affichage3(img,img_11,img_12)
```

```
# -*- coding:utf-8 -*-
```

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
```

```
# Q1
img = mpimg.imread("./PhotoBD.png")
# plt.imshow(img)
# plt.show()
```

```
# Q2
print(img.shape)
print(img.dtype)
print(img[300,300])
print(img.max())
print(img.min())
# (360, 640, 3)
# float32
# [0.38039216 0.41568628 0.34901962]
# 1.0
# 0.011764706
```

```
# Q3
def conv_1_2_255(img):
    """renvoie une copie de l'image img
    les éléments sont initialement des flottants dans [0,1]
    ils sont en sortie des entiers entre 0 et 255"""
    a = img*255
    a = a.astype(np.int)
    return a
```

```
b = conv_1_2_255(img)
print(b)
# [[[ 66  93 148]
#      [ 66  93 148]
#      [ 66  93 148]
#      ...
```

```
print(b.max())
print(b.min())
# 255
# 3
```

```
def conv_255_2_1(img):
    """renvoie une copie de l'image img
    les éléments sont initialement des entiers entre 0 et 255
    ils sont en sortie des flottants dans [0,1]"""
    a = img / 255
    a = a.astype(np.float)
    return a
```

```
c = conv_255_2_1(b)
print(c)
# [[ [0.25882353 0.36470588 0.58039216]
#      [0.25882353 0.36470588 0.58039216]
#      [0.25882353 0.36470588 0.58039216]
#      ...
```

```
print(c.max())
print(c.min())
# 1.0
# 0.011764705882352941
```

```
# Q4
def histo(img):
    """renvoie l'histogramme des couleurs de img
    qui est formée d'entiers entre 1 et 255
    sous la forme d'un tableau 256x3
    chaque colonne correspond à une couche R, V ou B
    chaque ligne au nombre d'occurrence du numéro de la ligne dans la couche"""
```

```
histogramme = np.zeros((256,3),np.int)
(n,p,_) = img.shape
for k in range(3):
    for i in range(n):
        for j in range(p):
            histogramme[img[i,j,k],k] += 1
plt.plot(histogramme[:,0], 'r')
plt.plot(histogramme[:,1], 'g')
plt.plot(histogramme[:,2], 'b')
# plt.show()
plt.savefig("./histogramme.png")
```

histo(b)

```
# Q5
plt.figure()
rouge=np.zeros(img.shape,dtype=np.float)
rouge[:, :, 0] = c[:, :, 0]
plt.title("trame rouge")
plt.imshow(rouge)
plt.savefig("./trame_rouge.png")

plt.figure()
vert=np.zeros(img.shape,dtype=np.float)
vert[:, :, 1] = c[:, :, 1]
plt.title("trame verte")
plt.imshow(vert)
plt.savefig("./trame_vert.png")

plt.figure()
bleu=np.zeros(img.shape,dtype=np.float)
bleu[:, :, 2] = c[:, :, 2]
plt.title("trame bleue")
plt.imshow(bleu)
plt.savefig("./trame_bleue.png")

# plt.show()

# Q6
plt.figure()
saturee_rouge=np.ones(img.shape,dtype=np.float)
saturee_rouge[:, :, 1:] = c[:, :, 1:]
plt.title("image saturée en rouge")
plt.imshow(saturee_rouge)
plt.savefig("./saturee_rouge.png")

plt.figure()
saturee_vert=np.ones(img.shape,dtype=np.float)
saturee_vert[:, :, 0] = c[:, :, 0]
saturee_vert[:, :, 2] = c[:, :, 2]
plt.title("image saturée en vert")
plt.imshow(saturee_vert)
plt.savefig("./saturee_vert.png")

plt.figure()
saturee_bleu=np.ones(img.shape,dtype=np.float)
saturee_bleu[:, :, :2] = c[:, :, :2]
plt.title("image saturée en bleu")
plt.imshow(saturee_bleu)
plt.savefig("./saturee_bleu.png")

# plt.show()

# Q7
plt.figure()
plt.title("inversion de couleur n -> 1-n")
plt.imshow(1-c)
plt.savefig("./inversion_couleur.png")

# Q8
```

```
def convert_nb_moy(img, r=1., v=1., b=1.) :
    """renvoie une nouvelle image, moyenne pondérée par
    r v b des couleurs de img"""
    # on normalise les coefficients
    r, v, b = r/(r+v+b), v/(r+v+b), b/(r+v+b)
    moyenne = r*img[:, :, 0] + v*img[:, :, 1] + b*img[:, :, 2]
    noiretblanc = np.zeros(img.shape, img.dtype)
    noiretblanc[:, :, 0] = moyenne
    noiretblanc[:, :, 1] = moyenne
    noiretblanc[:, :, 2] = moyenne
    return noiretblanc
```

```
plt.figure()
plt.title("en noir et blanc (moyenne pondérée)")
plt.imshow(convert_nb_moy(img))
plt.savefig("./en_noir_et_blanc_moyenne_ponderee.png")
```

Q9

```
def convert_nb_minmax(img) :
    """renvoie une nouvelle image, moyenne des
    min et max des trois niveaux pour chq pixel"""
    # on normalise les coefficients
    moyenne = .5*(img.min(axis=2)+img.max(axis=2))
    noiretblanc = np.zeros(img.shape, img.dtype)
    noiretblanc[:, :, 0] = moyenne
    noiretblanc[:, :, 1] = moyenne
    noiretblanc[:, :, 2] = moyenne
    return noiretblanc
```

```
plt.figure()
plt.title("en noir et blanc (moyenne des max/min)")
plt.imshow(convert_nb_minmax(img))
plt.savefig("./en_noir_et_blanc_minmax.png")
```

#Q10

```
def f(x) :
    return np.sqrt(x)

def g(x) :
    return .5*(1+np.sin(np.pi*(x-.5)))
```

```
vg = np.vectorize(g)
```

```
X = np.linspace(0, 1, 100)
Yf = f(X)
Yg = vg(X)
```

```
plt.figure()
plt.plot(X, Yf, label='f')
plt.plot(X, Yg, label='g')
plt.grid()
plt.legend(loc="best")
plt.savefig("./f_et_g.png")
```

#Q11

```
imgnb = convert_nb_minmax(img)
imgnb_contraste_f = f(imgnb)
imgnb_contraste_g = g(imgnb)
```

```
plt.figure()
plt.title("contrastée par f")
plt.imshow(imgnb_contraste_f)
plt.savefig("./contrastee_par_f.png")
```

```
plt.figure()
plt.title("contrastée par g")
plt.imshow(imgnb_contraste_g)
plt.savefig("./contrastee_par_g.png")
```

par f, les petites valeurs sont augmentées,

```
# tandis que par g, les petites valeurs sont diminuées
# Avec g, on accentue les différences donc le contraste.
```

```
# Q12
```

```
def convolution(image, matrice):
    """renvoie une image obtenue en appliquant la convolution
    de matrice "matrice" à "image" """
    (n,p,q) = image.shape
    imgconv = np.zeros((n,p,q))
    for i in range(1,n-1): # on n'applique pas la convolution sur les bords
        for j in range(1,p-1):
            # la formule proposée est trace(transposée(A)*B) cf produit scalaire
            for k in range(3):
                imgconv[i,j,k] = np.trace(image[i-1:i+2,j-1:j+2,k].dot(matrice))
    return imgconv
```

```
matrice1 = np.array([[1/8, 1/8, 1/8],
                    [1/8, -1/8, 1/8],
                    [1/8, 1/8, 1/8]])
```

```
matrice2 = np.array([[1, 1, 1],
                    [1, -2, 1],
                    [1, 1, 1]])
```

```
imgconv1 = convolution(img, matrice1)
imgconv2 = convolution(img, matrice2)
```

```
plt.figure()
plt.title("convolée par matrice 1")
plt.imshow(imgconv1)
plt.savefig("./convolee_par_matrice_1.png")
```

```
plt.figure()
plt.title("convolée par matrice 2")
plt.imshow(imgconv2)
plt.savefig("./convolee_par_matrice_2.png")
# Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
# [0..255] for integers).
# Je pense qu'on dépasse les valeurs 0 1
print(imgconv2.min(),imgconv2.max())
# 0.0 6.0352941155433655
# On normalise
imgconv2 = imgconv2 / imgconv2.max()
plt.figure()
plt.title("convolée par matrice 2 puis normalisé")
plt.imshow(imgconv2)
plt.savefig("./convolee_par_matrice_2_puis_normalise.png")
```

```
# Q14
# c'est vite compliqué !
# Si on suppose que l'on veut diviser la taille (hauteur et largeur) par deux :
# on peut prendre un pixel sur 2
# ou faire des moyennes des pixels supprimés (comme pour la convolution)
```

```
def reduit(image):
    return image[::2,::2,:]
```

```
imgreduite = reduit(img)
print(imgreduite.shape)
# (180, 320, 3)
plt.figure()
plt.title("réduite en supprimant un pixel sur deux")
plt.imshow(imgreduite)
plt.savefig("./reduite_en_supprimant_un_pixel_sur_deux.png")

plt.imsave("./reduite.png",imgreduite)
```