```python
# -*- 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 initalement 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 initalement 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_verte.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)
```