```
import matplotlib.image as img
import numpy as np


## Questions 1 et 2
ada = img.imread("portrait.png")
print(ada)                # un morceau du tableau ada
print(type(ada))          # il s'agit d'un tableau numpy (numpy.ndarray) à 3
    dimensions (ce qui correspond à
                          # un tableau en deux dimensions, lignes et colonnes, de
                              pixels
                          # i.e. de tableaux à une dimension contenant les trois
                              composantes de chaque couleur).
print(len(ada))           # nb h de sous-tableaux de ada i.e. nombre de lignes sur l'
    image
print(len(ada[0]))        # nb l de sous-sous-tableaux de ada (ou le nombre de sous-
    tableaux dans une ligne)
                          #    i.e. nombre de colonnes
print(len(ada[0, 0]))     # nb de couleurs additives utilisées, ici 3 : R, V et B
print(np.shape(ada))      # tuple contenant les éléments précédents (h, l, 3)
print(ada.shape)

''' Notons que si l'on ne place pas le fichier dans le répertoire de travail,
    on peut l'ouvrir en indiquant le chemin d'accès, par exemple:
    ada = img.imread("C:/Users/sebas/Documents/PCSI info 2022-2023/TP-09/portrait.png
        ")
'''

## Questions 3 et 4

import matplotlib.pyplot as plt
plt.figure(1)
plt.imshow(ada)
plt.show()

''' Les graduations sont les numéros de pixels, de 0 à len(ada)-1 en ordonnée
        et de 0 à len(ada[0])-1 en abscisse.

    Il s'agit de l'informaticienne Ada Lovelace qui vécut au XIXe siècle
        et fut l'autrice du tout premier programme informatique.
'''


## Question 5

def niveaux_gris(image):
    n = len(image)      # nb de lignes
    p = len(image[0])  # nb de colonnes, on peut aussi exploiter image.shape
    gris = np.zeros((n,p,3))
    for i in range(n):
        for j in range(p):
            a = image[i,j] # les trois composantes du pixel en i, j
            valeur = 0.299*a[0] + 0.587 * a[1] + 0.114*a[2]
            gris[i,j,0] = valeur
            gris[i,j,1] = valeur
            gris[i,j,2] = valeur
    return(gris)

ada_gris = niveaux_gris(ada)
plt.figure(2)
plt.imshow(ada_gris)
plt.show()
```

```
## Questions 6 et 7

ada_gris_B = ada_gris[:,:,0]

# pour vérifier les dimensions :
print(np.shape(ada_gris))
print(np.shape(ada_gris_B))

plt.figure(3)
plt.imshow(ada_gris_B)
plt.show()

''' L'image n'apparaît pas en niveaux de gris mais avec des couleurs arbitraires :
        une couleur est associée à chaque niveau de gris.
    Il faut donc préciser en argument de plt.imshow que l'on souhaite
        une visualisation en niveau de gris.
'''

plt.figure(4)
plt.imshow(ada_gris_B,'gray')
plt.show()



## Questions 8, 9 et 10

def seuil(imageB,s):
    a, b = imageB.shape
    nouveau = np.zeros((a, b))
    for i in range(a):
        for j in range(b):
            if imageB[i, j] >= s:
                nouveau[i, j]= 1
    return nouveau


photo = img.imread("Particules.png")
# plt.figure(99)
# plt.imshow(photo)
photoB = photo[:,:,0] # passage au format B
photo_seuil = seuil(photoB, 0.65)
plt.figure(5)
plt.imshow(photo_seuil,'gray')
plt.show()

''' Après plusieurs tests, on retiendra s = 0.65
    Cependant, malgré un seuillage plutôt efficace, certains pixels du fond
       apparaissent en noir.
        Pour éviter le problème, on pourrait procéder à un floutage de la photo de
           départ
        qui enlève de facto les pixels trop noirs ou trop blancs.
'''

photo2 = img.imread("Particules_2.png")
# plt.figure(89)
# plt.imshow(photo2)
photoB = photo2[:,:,0] # passage au format B
photo2_seuil = seuil(photoB, 0.65)
plt.figure(6)
plt.imshow(photo2_seuil,'gray')
plt.show()
```

```
## Question 11

def voisins(tableau, k, l):
    """ entrée: tableau de nombres et deux indices entiers k et l
        sortie : deux nombres (les valeurs contenues à gauche et en haut de l'élément
             ligne k, colonne l)
            avec valeur -1 si l'on est au bord.
    """
    if k == 0:
        if l == 0:
            return (-1,-1)
        else:
            return(tableau[k,l-1],-1)
    else:
        if l == 0:
            return(-1, tableau[k-1,l])
        else:
            return (tableau[k,l-1],tableau[k-1,l])


## Question 12

def connexes(imageB_seuillée):
    """ fonction qui prend en argument une image seuillée
            en noir et blanc de particules noires (intensité 0) sur fond blanc (
                intensité 1)
            et qui "cherche" les composantes connexes.
        Elle renvoie une image (tableau de pixels) où,
            à l'intérieur de chaque composante connexe (i.e. particule),
            chaque pixel porte une étiquette identique (un nombre entier à partir de
                1),
            mais différente de celle des autres composantes connexes.
            Les pixels du fond portent l'étiquette -1
    """

    resul = imageB_seuillée * -1 # met des -1 sur le fond, les pixels des particules
        restent à 0
    n, p = np.shape(resul) # nb de lignes, nb de colonnes, n*p est le nb total de
        pixels
    E = np.arange(n*p+1) # tableau d'équivalence des étiquettes, il en faut au max
        autant que de pixels.
    # Initialement, en chaque position i, il y a l'entier i

    label = 1 #initialisation de la première étiquette

    # scan 1

    for i in range(n):
        for j in range(p):
            if resul[i,j] != -1: # on est sur une particule !
                if voisins(resul, i, j) == (-1, -1): # a priori, le pixel (i,j) est
                    sur une nouvelle particule
                    resul[i, j] = label # on l'étiquette
                    label += 1 # on incrémente l'étiquette pour plus tard (prochaine
                        particule)
                elif min(voisins(resul, i, j)) == -1 and max(voisins(resul, i, j)) !=
                     -1: # l'un des pixels voisins n'est pas le fond et a déjà donc
                    été étiqueté (label >=1 > -1)
                    resul[i, j] = max(voisins(resul,i,j)) # le pixel (i,j) touchant
                        ce voisin, il doit porter la même étiquette
                elif min(voisins(resul,i,j)) != -1  and max(voisins(resul,i,j)) !=
                    -1: # les deux pixels voisins n'appartiennent pas au fond
                    resul[i, j] = min(voisins(resul,i,j)) # le pixel (i,j) doit
                        porter la plus petite étiquette des deux
```

```
                    if min(voisins(resul,i,j)) != max(voisins(resul,i,j)) : # si les
                        deux voisins ne portent pas la même étiquette, ils se
                        touchent pourtant via le pixel (i,j)
                        E[int(max(voisins(resul,i,j)))] = min(voisins(resul,i,j)) #
                            il faut mettre à jour la liste des équivalences

    E_final = E[:label]

    #print("dernière valeur de label : ", label,"\n E tronqué à la fin de la boucle",
        E[0:label])


    # mise à jour globale de la liste des équivalences :
    # à chaque position i (correspondant à une étiquette i attribuée à un pixel) doit
         se trouver la valeur de la plus petite étiquette possible en contact avec
        les pixels d'étiquette i

    for i in range (len(E_final)):
        if E_final[i] != i:
            indice = E_final[i]
            E_final[i] = E_final[indice]

    final = np.copy(resul)

    # scan 2

    for i in range(n):
        for j in range(p):
            if final[i,j] != -1:
                final[i,j] = E[int(resul[i,j])] # on remplace toutes les étiquettes
                    grâce à la liste d'équivalence de façon à ce que les pixels de
                    chaque particule portent la même étiquette

    return(resul,final,E_final) # retourne le résultat de chacun des deux scans et la
         liste d'équivalence


## Question 13

# avec un détail de la photo :
# photo_test = seuil(img.imread("Particules_2_detail.png")[:,:,0],0.65)

#avec la photo complète :
photo_test = seuil(img.imread("Particules_2.png")[:,:,0], 0.65)

resul_test = connexes(photo_test)[0]
final_test = connexes(photo_test)[1]
E_test = connexes(photo_test)[2]

plt.figure(11)
plt.imshow(photo_test, 'gray')

plt.figure(12)
plt.imshow(resul_test)

plt.figure(13)
plt.imshow(final_test)

plt.show()
```

```python
## Question 14

def nb_particules(E):
    compteur = 0
    for i in range(1,len(E)): # 0 n'est pas une étiquette
        if E[i] == i : # Chaque étiquette i apparaît pour la première fois en
            position i
                    # (dans ce cas : E[i] == i), il suffit de les compter
            compteur += 1
    return(compteur)

print(nb_particules(E_test))
```