## Python Files\kmoy.py

```python
from PIL import Image, ImageOps
import numpy as np
import random as rd
import math

# Lecture de l'image et changement du contraste
image = Image.open("./images/2.jpg",mode = 'r')
image = ImageOps.autocontrast(image, cutoff=10)

im = np.asarray(image)

nb_lignes, nb_colones, nb_subpixel = np.shape(im)                                   # nb_subpixel est le nombre d'élements d'un points (R, G, B) = 3
classe = [[-1 for i in range(nb_colones)] for j in range(nb_lignes)]                # Tableau des classes de chaque éléments
nb_classe = 8

# initialisation centres
centres = []

for i in range(nb_classe) :
    ir = rd.randint(0,nb_lignes-1)
    jr = rd.randint(0,nb_colones-1)
    while [im[ir,jr,s] for s in range(nb_subpixel)]  in centres :
        ir = rd.randint(0,nb_lignes-1)
        jr = rd.randint(0,nb_colones-1)
    centres.append([im[ir,jr,s] for s in range(nb_subpixel)])                       # On prend des pixels aléatoires dans l'image, et on vérifie qu'ils sont tous differents

# distance (voire pour prendre plus efficace ?)

def distance(e1: list, e2: list) :                                                  # distance entre deux couleurs
    d = len(e1)
    res = 0
    for i in range(d) :
        res = max(res,(e1[i]-e2[i])**2)
    return(res)

```

```
# ranger au plus proche des centres
def classer() :                                                    # permet de classer les elements dans chaque classe
    global classe, im, nb_lignes, nb_colones, centres
    b = False
    for i in range(nb_lignes) :                                    # On parcours toute l'image
        for j in range(nb_colones) :                               # --
            dmin = 0
            if classe[i][j] == -1 :                                # si il n'y a pas encore de classe
                dmin = math.inf                                    # la distance est la minimum
            else :
                dmin = distance(im[i,j], centres[classe[i][j]])
            for m in range(nb_classe) :
                dist = distance(list(im[i,j]), centres[m])
                if dist<dmin :
                    dmin = dist
                    classe[i][j] = m
                    b = True
    return(b)


# les centres bougent
def barycentre() :
    global classe, im, nb_lignes, nb_colones, centres
    dico = dict()
    for i in range(nb_classe) :
        dico[i] = []
    for i in range(nb_lignes) :
        for j in range(nb_colones) :
            dico[classe[i][j]].append(im[i,j])
    for i in range(nb_classe) :
        r, g, b = 0, 0, 0
        for e in dico[i] :
            r += e[0]
            g += e[1]
            b += e[2]
        m = len(dico[i])
        centres[i] = [r//m, g//m, b//m]
# kmoyennes
i=0
while classer() :
    i+=1
```

```
    print("\nstep"+str(i))
    print(centres)
    barycentre()
    im_classes = np.array([[centres[classe[i][j]] for j in range(nb_colones)] for i in range(nb_lignes)])
    res = Image.fromarray(im_classes, mode='RGB')


# affichage,sauvegarde

colors = [[0,0,0],[150,0,0],[0,150,0],[0,0,150],[128,128,0],[0,160,160],[192,0,192],[230,230,230]]  # liste des couleurs pour bien différencier les classes

# Enregistrer toutes les images
for lettre in range(nb_classe):
    im_classes = np.array([[im[i][j] if classe[i][j] == lettre else (0, 0, 0) for j in range(nb_colones)] for i in range(nb_lignes)],dtype=np.uint8)
    print(im_classes)
    res = Image.fromarray(im_classes, mode="RGB")
    res.show()
    res.save(f"./out/2_{lettre}.jpg")


b = np.array([[colors[classe[i][j]] for j in range(nb_colones)] for i in range(nb_lignes)],dtype=np.uint8)
res = Image.fromarray( b,mode="RGB")
res.show()
res.save("./out/out.jpg")
```