```python
import numpy as np
import matplotlib.pyplot as plt
import random as rd


# on va adapter les fonctions de kmeans pour que l'on récupère
# à la fin la liste des coordonées des pixels par clusters


def distance(x1,x2):
    '''distance euclidienne entre deux pixels'''
    #pb car les pixels sont encodés en uint8, donc il faut tout convertir en float
    d=0.0
    for i in range(len(x1)):
        d+=(float(x1[i])-float(x2[i]))**2
    return np.sqrt(d)

def centre(liste):
    '''calcule l'isobarycentre de la liste de pixels'''
    p=len(liste) #le nombre de points
    #p ne pourra pas être nul car il y aura toujours au moins le centre
    n=len(liste[0]) #la dimension de nos pixels

    c=[] #pour stocker les coordonnées du centre
    for i in range(n):#calcul de la ième coordonnée du centre
        s=0
        for j in range(p):
            s+=float(liste[j][i]) #converion en flottant car uint8 au départ
        c.append(s/p)
    return c

def debut(tableau,k):
    '''choisit au hasard k pixels distincts dans le tableau'''
    n,p,q=np.shape(tableau)

    liste_coord=[[i,j] for i in range(n) for j in range(p)]


    choix_coord=[]
    while len(choix_coord)<k:
        x=rd.choice(liste_coord)

        if x not in choix_coord:
            choix_coord.append(x)

    return [tableau[coord[0],coord[1]] for coord in choix_coord]

def clusters(Lc,tableau):
    '''Lc est la liste des centres, tableau le tableau des pixels,
    retourne la liste des clusters et la liste de leurs coordonnées'''
    k=len(Lc) #le nombre de clusters
    liste_clusters=[[] for i in range(k)]
    liste_coord=[[] for i in range(k)]
    n,p,q=np.shape(tableau)

    #on va affecter chaque pixel à un cluster
    for i in range(n):
        for j in range(p):
            ind_min=0
            d_min=+np.inf
            for ind in range(k):
                d=distance(tableau[i,j],Lc[ind])
                if d<d_min:
                    ind_min=ind
                    d_min=d
            liste_clusters[ind_min].append(tableau[i,j])
            liste_coord[ind_min].append([i,j])

    return liste_clusters,liste_coord

def arret(Lc_old,Lc_new,epsilon):
    k=len(Lc_old)
```

```
    test=True
    for i in range(k):
        if distance(Lc_old[i],Lc_new[i])>epsilon:
            test=False
    #le test sera encore True si toutes les distances ont été plus petites que epsilon
    return test



points_ex=[[1,2],[3,4],[6,7],[11,2],[23,4],[6,70]]


def kmeans(tableau,k):
    Lc=debut(tableau,k) #on initialise aléatoirement les centres
    test=False
    compteur=0 #pour savoir combien d'itérations seront nécessaires
    while test==False and compteur<10:
        compteur+=1
        liste_clusters=clusters(Lc,tableau) #la liste des clusters et de leurs
        coordonnées

        #on va recalculer tous les centres
        Lc_new=[]
        for liste in liste_clusters[0]:
            Lc_new.append(centre(liste))

        test=arret(Lc,Lc_new,0.5)

        Lc=Lc_new #on actualise les centres

    return Lc,liste_clusters,compteur

def compression(tableau,k):
    n,p,q=np.shape(tableau)
    newtab=np.zeros((n,p,q),dtype=int)
    triplet=kmeans(tableau,k)
    Lc,liste_clusters=triplet[0],triplet[1]
    print(np.shape(Lc))
    for i in range(k):
        liste_coord=liste_clusters[1][i]
        for coord in liste_coord:
            newtab[coord[0],coord[1]]=np.array([Lc[i][0],Lc[i][1],Lc[i][2]])
    return newtab


tab=plt.imread('tigre.bmp') #convertit l'image en un tableau numpy
# print(tab)
# print('Taille du tableau:',np.shape(tab))
n,p,q=np.shape(tab)

plt.imshow(tab) #affichage du tableau dans Python, sous forme d'image
plt.savefig('tigre.png')
plt.show()

print(n,p)

tigre_compressé=compression(tab,8)
plt.imshow(tigre_compressé)
plt.savefig('tigre_compressé.png')
plt.show()
```