```python
#Le programme présenté ici est en dimension n.
#Si on prend n=2, on se retrouve dans R^2, comme l illustration du cours


import numpy as np
import matplotlib.pyplot as plt
import random as rd
import time #pour faire des pauses dans l'affichage

##############################################################################
##############################################################################
# Première partie "facile": affichage dun nuage de points et de centres choisis au hasard


#génération d'un nuage de points
def points_random(n,p):
    '''retourne une liste de p points de R^n, de coordnonnées aléatoires entre 0 et 100'''
    liste=[]
    for i in range(p):
        point=[rd.randint(0,100) for j in range(n)] #génère n coordonnées du point
        liste.append(point)
    return liste

#tracé de ce nuage
n=2   #on se place en dimension 2
p=300  #on va prendre 300 points
liste_points=points_random(n,p)
listeX=[liste_points[i][0] for i in range(p)]  #on récupère les abscisses
listeY=[liste_points[i][1] for i in range(p)]  #on récupère les ordonnées
plt.plot(listeX,listeY,'ob')
plt.show()

#choix de k centres au hasard
def debut(liste,k):
    '''choisit au hasard k points distincts parmi la liste'''
    #attention rd.choices(liste,k=k) ne fonctionne pas car il peut y avoir des répétions
    choix=[]
    while len(choix)<k:
        x=rd.choice(liste)
        if x not in choix:
            choix.append(x)
    return choix

#tracé des centres en plus du nuage
n=2   #on se place en dimension 2
p=300  #on va prendre 300 points
liste_points=points_random(n,p)
listeX=[liste_points[i][0] for i in range(p)]  #on récupère les abscisses
listeY=[liste_points[i][1] for i in range(p)]  #on récupère les ordonnées
plt.plot(listeX,listeY,'ob')

liste_centres=debut(liste_points,3) #on va faire 3 classes
listecX=[liste_centres[i][0] for i in range(3)]  #on récupère les abscisses
listecY=[liste_centres[i][1] for i in range(3)]  #on récupère les ordonnées
plt.plot(listecX,listecY,'*r')
plt.show()

##############################################################################
##############################################################################
# Programmation de l algorithme des k moyennes

def distance(A,B):
    '''distance euclidienne entre deux points
    A et B sont deux couples si on est dans R^2(les coordonnées des deux points)'''
    d=0
    for i in range(len(A)):
        d+=(A[i]-B[i])**2
    return np.sqrt(d)

```

```
70  def centre(liste):
71      '''calcule l'isobarycentre d'une liste de points'''
72      p=len(liste) #le nombre de points
73      #p ne pourra pas être nul car il y aura toujours au moins le centre
74      n=len(liste[0]) #la dimension de nos points
75
76      c=[] #pour stocker les coordonnées du centre
77      for i in range(n):#calcul de la ième coordonnée du centre
78          s=0
79          for point in liste:
80              s+=point[i]
81          c.append(s/p)
82      return c
83
84
85
86  def clusters(Lc,Lp):
87      '''Lc est la liste des centres, Lp la liste des points,retourne la liste des
        clusters'''
88      k=len(Lc) #le nombre de clusters
89      liste_clusters=[[] for i in range(k)] #toutes les listes sont vides au début
90      for point in Lp: #on va affecter chaque point à un cluster
91          i_min=0
92          d_min=+np.inf
93          for i in range(k):
94              d=distance(point,Lc[i])
95              if d<d_min:
96                  i_min=i
97                  d_min=d
98          liste_clusters[i_min].append(point)
99      return liste_clusters
100
101 def arret(Lc_old,Lc_new,epsilon):
102     '''retourne False si certains centres de classe ont bougé de plus de epsilon
        retourne True sinon, dans ce cas on arretera le programme'''
103     k=len(Lc_old)
104     test=True
105     for i in range(k):
106         if distance(Lc_old[i],Lc_new[i])>epsilon:
107             test=False
108     #le test sera encore True si toutes les distances ont été plus petites que epsilon
109     return test
110
111
112
113 def kmeans(Lp,k,epsilon):
114     '''à partir d'une liste de points Lp, va retourner une liste de k centres
        une liste de k clusters, et un compteur indiquant le nombre d'iterations'''
115     Lc=debut(Lp,k) #on initialise aléatoirement les centres
116     test=False
117     compteur=0 #pour savoir combien d'itérations seront nécessaires
118     while test==False:
119         compteur+=1
120         liste_clusters=clusters(Lc,Lp) #la liste des clusters
121
122         #on va recalculer tous les centres
123         Lc_new=[]
124         for liste in liste_clusters:
125             Lc_new.append(centre(liste))
126
127         test=arret(Lc,Lc_new,epsilon)
128
129         Lc=Lc_new #on actualise les centres
130
131     return Lc,liste_clusters,compteur
132
133
134 def kmeans_tracé(Lp,k,epsilon):
135     numéro=0 #pour numéroter les figures sauvegardées
136
137     #un choix de k couleurs au hasard
138     liste_couleurs = [[rd.random(),rd.random(),rd.random()] for i in range(k)]
139
140
```

```
141
142         Lc=debut(Lp,k) #on initialise aléatoirement les centres
143
144         #tracé de tous les individus et des premiers centres
145         liste_x=[coord[0] for coord in Lp]
146         liste_y=[coord[1] for coord in Lp]
147         plt.plot(liste_x,liste_y,'ob')
148         listec_x=[coord[0] for coord in Lc]
149         listec_y=[coord[1] for coord in Lc]
150         plt.scatter(listec_x,listec_y,s=300,marker='*',color='red')
151         plt.savefig('étape'+str(numéro)+'.png')
152         plt.show()
153         time.sleep(3) #pause de 2 secondes
154         numéro+=1
155
156         test=False
157         compteur=0 #pour savoir combien d'itérations seront nécessaires
158         while test==False and compteur<3:
159
160             compteur+=1
161             liste_clusters=clusters(Lc,Lp) #la liste des clusters
162
163             #le tracé
164             i=0
165             for liste in liste_clusters:
166                 couleur=liste_couleurs[i]
167                 liste_x=[coord[0] for coord in liste]
168                 liste_y=[coord[1] for coord in liste]
169                 plt.plot(liste_x,liste_y,'o',color=couleur)
170                 i+=1
171             listec_x=[coord[0] for coord in Lc]
172             listec_y=[coord[1] for coord in Lc]
173             plt.scatter(listec_x,listec_y,s=300,marker='*',color='red')
174             plt.savefig('étape'+str(numéro)+'.png')
175             plt.show()
176             time.sleep(2) #pause de 2 secondes
177             numéro+=1
178
179             #on va recalculer tous les centres
180             Lc_new=[]
181             for liste in liste_clusters:
182                 Lc_new.append(centre(liste))
183
184
185             test=arret(Lc,Lc_new,epsilon)
186
187             Lc=Lc_new #on actualise les centres
188
189             #le tracé
190             i=0
191             for liste in liste_clusters:
192                 couleur=liste_couleurs[i]
193                 liste_x=[coord[0] for coord in liste]
194                 liste_y=[coord[1] for coord in liste]
195                 plt.plot(liste_x,liste_y,'o',color=couleur)
196                 i+=1
197             listec_x=[coord[0] for coord in Lc]
198             listec_y=[coord[1] for coord in Lc]
199             plt.scatter(listec_x,listec_y,s=300,marker='*',color='red')
200             plt.savefig('étape'+str(numéro)+'.png')
201             plt.show()
202             time.sleep(2) #pause de 2 secondes
203             numéro+=1
204
205         print(compteur)
206
207     #éxécution de notre programme
208     n=2   #on se place en dimension 2
209     p=300  #on va prendre 300 points
210     liste_points=points_random(n,p)
211
212     kmeans_tracé(liste_points,3,0.0001)
```