```python
# -*- coding: utf-8 -*-
#
# 2024-2025 - TP7 IA : algorithme des K moyennes codé à partir de Quentin Fortier
# site : https://cpge-itc.github.io/itc2/dl/apprentissage/cours/kmeans/kmeans.html
# Auteur : Quentin Fortier et recomposé par Yvan Crévits
# Date de création : 3 avril 2024
# Date de modification (adaptation) : 6 mars 2025

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

# Fermeture des fenêtre déjà ouvertes (pas demandé dans le TP)
plt.close('all')

# II. Préparation de données à manipuler
print('')
# Q.1
dim, nb = 2, 20 # Dimension dim de 2 et nb = 20 données
# Génération de paquets de points en agglomérant des données
X = np.vstack([np.array(p + np.random.randn(nb, dim))
               for p in [[3, 2], [0, -2], [-3, 3], [-3, -3]]]).tolist()
''' La fonction vstack() agglomère les sous-tableaux verticalement, c'est à dire
en les passant à la ligne les uns des autres.
'''

# Q.2
''' Les points sont générés au hasard autour des 4 points repérés
par leurs coordonnées : [3, 2], [0, -2], [-3, 3], [-3, -3] et convertis en
liste pour les traitements ultérieurs (ce n'est pas traité en tableaux NumPy)
'''
# Q.3
''' Il faut importer le module numpy avec np pour nom d'alias'''
# Cette ligne sera complétée plus tard
plt.scatter([x[0] for x in X], [x[1] for x in X]) # Expliquer
# Cette ligne sera complétée plus tard
# Cette ligne sera complétée plus tard
plt.title('Tirage de ' + str(nb) + ' points initiaux aléatoires')
plt.savefig('kmeans_init.pdf') # Pour garder le résultat (optionnel)
plt.show()

# Q.4
''' En ligne 7, le tracé place des points par leurs coordonnées, à la différence
```

```
de plot qui trace des segements de droites entre les points. Les coordonées de
ceux-ci sont construites dans les deux listes définies en comprhénsion.
L'adapatation du script nécessite d'importer matplotlib.pyplot.
Côté observation, les points aux position aléatoires sont bien placés autour
des points de coordonnées des lignes 3 et 4, traité en Q.2.'''

# III. Installation des centres, évaluation et caractérisation de leur position
print('III. Installation des centres, évaluation et caractérisation de leur position')
# Q.5
''' Il faut définir K : K = 4, d'où le script final suivant'''

# Génération aléatoire de points avec K = 4 centres initialement aléatoires
dim, nb, K = 2, 20, 4 # 20 pts, dans un plan (dim 2) pour 4 classes envisagées
#nb *= 10
# Génération de paquets de points en agglomérant des données
X = np.vstack([np.array(p + np.random.randn(nb, dim))
               for p in [[3, 2], [0, -2], [-3, 3], [-3, -3]]]).tolist()
# Construction de points aléatoires agglomérés en 4 paquets (4 classes)
centres = (np.random.rand(K, dim)*6 - 3).tolist() # Placement des centres aléatoires
plt.scatter([x[0] for x in X], [x[1] for x in X]) # Tracé des points
plt.scatter([x[0] for x in centres], [x[1] for x in centres], \
            marker='X', s=100, c='g') # Tracé des centres initiaux
#            marker='D', s=100, c='r') # Tracé des centres initiaux
plt.title('Tirage de '+str(nb)+' points initiaux aléatoires')
plt.savefig('kmeans_init.pdf')
plt.show()

# Q.6 — Manipulations

# Q.7 - Fonction centre()
# Codage de l'algorithme
def centre(X): # Calcule la moyenne des coordonnées des vecteurs de X
    c = [0.]*dim # Init. vect.du plan : limitation à dim=2 dimensions (variable globale)
    if len(X) == 0: # Vecteur nul ?
        return c # Renvoyé sans évaluation
    for x in X: # Pour chaque vecteur
        for i in range(len(x)): # Pour chaque composante
            c[i] += x[i] # Sommation
    for i in range(len(c)):
        c[i] /= len(X) # Division de chaque composante par le nombre n de vecteurs de données
    return c # Renvoi du vecteur centre (isobarycentre)

# Q.8 - Essai de centre()
x1 = (0, 0) ; x2 = (10, 0) ; x3 = (10, 20) ; x4 = (0, 20)
```

```
XX = [x1, x2, x3, x4]
print(centre(XX))

# Q.9 - Fonction d()
def d(x, y): # x et y sont deux vecteurs (listes ou tableaux)
    s = 0. # initialisation de la somme
    for i in range(len(x)): # pour chaque composante
        s += (x[i] - y[i])**2  # Sommation cumulée
    return s**.5  # Renvoi du résultat

# Essai de d()
x1 = (0, 0) ; x2 = (10, 0) ; x3 = (10, 20) ; x4 = (0, 20)
XX = [x1, x2, x3, x4]
print([d(x, centre(XX)) for x in XX])
print([d((0, 0), x) for x in XX])

# Q.10 - Fonction V()
def V(X): # X est un groupe de vecteurs (listes de listes ou tableaux)
    c = centre(X) # Évaluation du centre
    sd = 0. # initialisation de la somme des distances
    for x in X: # Pour chaque vecteur du groupe X
        sd += d(x, c)**2  # Sommation cumulée
    return sd  # Renvoi du résultat

# Essai de V()
x1 = (0, 0) ; x2 = (10, 0) ; x3 = (10, 20) ; x4 = (0, 20) # Rappel
XX = [x1, x2, x3, x4]
print(V(XX))

# IV. Algorithme des K-moyennes
print('IV. Algorithme des K-moyennes')
# Q.11 - Fonction calculer_centres()
def calculer_centres(classes): # Détermine le centre des K classes
    centres = [] # Initialisation liste des classes
    for i in range(len(classes)): # Scrutation des classes
        centres.append(centre(classes[i])) # Calcul et ajout des centres
    return centres # Renvoi des centres calculés

# Q.12 - Fonction plus_proche()
def plus_proche(x, centres): # Déterminer la classe de centre le + proche proche de x
    imin = 0 # Intialisation
    for i in range(len(centres)): # Recherche « classique » de minimum pour chaque classe
        if d(x, centres[i]) < d(x, centres[imin]):
            imin = i # Mise à jour de la classe d'affectation du vecteur x
```

```
    return imin # Renvoi l'indice de la classe appropriée (celle du centre le plus proche)

# Q.13 - Fonction calculer_classes()
def calculer_classes(X, centres): # Créer les classes de vecteurs les plus proches de chaque centre
    classes = [[] for i in range(len(centres))] # Initialisation des classes (vides)
    for x in X: # Pour chaque vecteur des observations
        classes[plus_proche(x, centres)].append(x) # Affectation du vecteur x à la classe de centre la plus proche
    return classes # Renvoi des classes remplies

# Q.14 - Fonction kmeans()
def kMeans(X, centres): # Algorithme des K-moyennes des vecteurs observation pour tous les centres
    centres2 = None # Initialisation des centres antérieurs
    while centres != centres2: # Répétition des déplacements & réaffectations des centres & classes
        centres2 = centres # Mémorisation centre actuel
        classes = calculer_classes(X, centres2) # Détermination des classes pour les nouveaux centres
        centres = calculer_centres(classes) # Calcul des nouveaux centres
    return classes # Renvoi des nouvelles classes après mise à jour)

# V. Évaluation de l'efficacité de l'algorithme
print('# V. Évaluation de l'efficacité de l'algorithme')
# Q.15 - Fonction inertie()
def inertie(classes, centres): # Calcul de l'inertie globale de toutes les classes
    s = 0. # Initialisation
    for i in range(len(centres)): # Pour chaque centre de classe
        for x in classes[i]: # Et pour chaque vecteur de la classe
            s += d(x, centres[i])**2 # Variance par sommation des carrés des distances
    return s # Renvoi de l'inertie

# Q.16 - Nouveau tracé du nuage de points après classification
classes = kMeans(X, centres) # # Appel de kmeans
cmap = ListedColormap(['r', 'g', 'b', 'purple'])
centres = calculer_centres(classes)
plt.figure()
plt.scatter([x[0] for x in X], [x[1] for x in X], c=[plus_proche(x, centres) for x in X], cmap=cmap)
plt.scatter([x[0] for x in centres], [x[1] for x in centres], marker='x', s=100, c=range(K), cmap=cmap)
plt.title("Inertie = " + str(inertie(classes, centres)))
plt.savefig('kmeans_base_final.pdf')
plt.show()

# VI. Recherche du nombre de classes K optimal
print('VI. Recherche du nombre de classes K optimal')
# Q.17 - Script de tracé de la courbe du coude
Kmax = 7 # Pour ce tracé, se limiter à K = 7 (inutile au-delà)
inerties = [] # Initialisation de la liste de stockage des inerties
```

```
for k in range(1, Kmax+1):
    centres = (np.random.rand(k, dim)*6 - 3).tolist()
    classes = kMeans(X, centres)
    centres = calculer_centres(classes)
    inerties.append(inertie(classes, centres))
plt.figure()
plt.plot(range(1, Kmax+1), inerties)
plt.xlabel("Valeur de K")
plt.ylabel("Inertie")
plt.grid()
plt.text(4, 190, '$K_{max}$ retenu')
plt.savefig('elbowCurve.pdf')
plt.show()
```