Projet Séquence 1

I. Explications

Dans la plupart des supermarchés, une balance permet aux clients de peser et étiqueter les fruits et légumes qu'ils achètent. Une caméra effectue une reconnaissance du produit et propose, à l'aide d'un menu, le produit s'il est reconnu ou bien un choix de produits similaires.

Modèle KH 800 SV de BIZERBA (Image du constructeur).

Le programme qui simule le fonctionnement d'une telle balance existe déjà et vous est donné ()"seq_1_proj_1_fichier_eleve.py".)

La caméra sera remplacée par une affectation dans le programme du code hexadécimal de la couleur du produit.

A partir de cette référence donnant la couleur du produit, vous devrez afficher la liste des produits de la même couleur dominante.

II. Le programme initial

```
import math
```

```
def liste_fruits_legumes():
    """
    fonction qui crée une liste des fruits et légumes par couleur
    3 couleurs dominantes : rouge, vert, jaune
    chaque produit est caractérisé par sa couleur décomposée en RVB et sa
couleur dominante

    Entrée
    ------
    rien

    Retourne
    --------
    listefl : liste des fruits et légumes
    """

    listefl=[]

    #produits à couleur dominante rouge = 0
```

```
    Listefl =  [["tomate", 247, 75, 12, 0] ,
               ["poivron rouge", 225, 36, 63, 0],
               ["pomme Gala", 197, 11, 9, 0]]
    #produits à couleur dominante verte = 1
    listefl.append(    ["courgette", 9, 191, 18, 1],
                       ["poireau", 20, 132, 26, 1],
                       ["pomme Granny Smith", 83, 221, 90, 1])
    #produits à couleur dominante jaune = 2
    listefl.append(    ["banane", 233, 255, 23, 2],
                       ["citron", 254, 240, 4, 2],
                       ["raisin", 241, 253, 12, 2])

    return listefl
```

```
def extraire_rvb(nb_hexa):
    """
    Fonction qui découpe une code couleur en hexadécimal sur 24 bits en couleurs
RVB
    Utilisation d'un dictionnaire

    Entrée
    ------
    nb_hexa : chaine de caractères

    Retourne
    --------
    Rouge : entier   # compris entre 0 et 255
    Vert : entier   # compris entre 0 et 255
    Bleu : entier   # compris entre 0 et 255
    """
    code_hexa = {'0':0 , '1' : 1 , '2' : 2 , '3' : 3 , '4' : 4 , '5' : 5 , '6' :
6 , '7' : 7 , '8' :8 , '9' : 9 ,'A':10 , 'B':11 , 'C':12 ,'D':13 , 'E':14 ,
'F':15 }

    assert (len(nb_hexa) == 6) , 'doit être codé sur 6 caractères'

    Rouge = code_hexa[nb_hexa[0]]*16 + code_hexa[nb_hexa[1]]
    Vert = code_hexa[nb_hexa[2]]*16 + code_hexa[nb_hexa[3]]
    Bleu = code_hexa[nb_hexa[4]]*16 + code_hexa[nb_hexa[5]]

    return [Rouge, Vert, Bleu]
```

```
def distances_euclidiennes(rouge , vert ,bleu ,listefl):

    """
    Fonction qui calcule les distances (formule euclidienne de Pythagore)
    entre les coordonnées R, V et B du produits et ceux de la liste

    Entrée
    ------
    rouge, vert, bleu : composantes RVB du produit
    listefl :liste des fruits et légumes enregistrés

    Retourne
```

```
    --------
    dist : table : Liste de couples [distance, indice]
    """

    dist = []    #Initialisation

    for i in range(len(liste)):
       calcul_distance = int(math.sqrt((rouge - listefl[i][1])**2 + (vert -
listefl[i][2])**2 + (bleu - listefl[i][3])**2))
       dist.append([calcul_distance, i])

    return dist
```

```
def tri_distances(distances_a_trier):

    """
    Fonction qui tri les distances par ordre croissant
    Algorithme du tri par insertion

    Entrée
    ------
      distances_a_triees : liste non triées (liste de listes [distance, index])

    Retourne
    --------
      distances_a_triees : liste triées (liste de listes [distance, index])
    """

    for i in range(1, len(distances_a_trier)):
        if distances_a_trier[i][0] < distances_a_trier[i-1][0]:
            memo = distances_a_trier[i]
            j = i

            while j > 0 and memo[0] < distances_a_trier[j-1][0]:
                distances_a_trier[j] = distances_a_trier[j-1]
                j = j - 1

            distances_a_trier[j] = memo

    return distances_a_trier
```

```
def recherche_k_voisins(dist, listefl, kvoisins):
    """
    Fonction qui recherche les k plus proches voisins

    Entrée
    ------
    dist : liste des distances non triées entre les composantes du produit
          saisies et les produits de la liste
    listefl : liste des fruits et légumes enregistrés
    kvoisins : entier : nombre de plus proches voisins voulus

    Retourne
    --------
```

```
    nombre_de_produit_par_couleur : liste des nb de produits par couleur
    """

    # Tri des distances par ordre croissant
    distances_triees = tri_distances(dist)

    #recherche et affiche les produits de la liste qui sont les K plus proches
voisins
    nombre_de_produit_par_couleur = [0, 0, 0]

    # Affiche les k premières valeurs de la liste des voisins déjà triée.
    for i in range(kvoisins):
        print("Voisin n°"+str(i + 1) + " :", listefl[distances_triees[i][1]])
        nombre_de_produit_par_couleur[listefl[distances_triees[i][1]][4]] += 1

    return nombre_de_produit_par_couleur
```

```
def classe_produit(nombre_de_produit_par_couleur,listefl):
    """
    Fonction qui recherche la classe du produit cad de sa couleur dominante

    Entrée
    ------
    nombre_de_produit_par_couleur : liste des nb de produits par couleur
    listefl :liste des fruits et légumes enregistrés

    Retourne
    --------
    coul_dom : couleur dominante du produit
    """

    Couleurs = ("rouge", "vert", "jaune")

    #recherche du max dans nombre_de_produit_par_couleur
    max=0
    for i in range(len(nombre_de_produit_par_couleur)):
        if nombre_de_produit_par_couleur[i] > max:
            max = nombre_de_produit_par_couleur[i]
            coul_dom = i

    print("Le produit est de couleur dominante "+couleurs[coul_dom]+"\n")

    return coul_dom
```

```
#------------ programme principal ----------------------------------------

# Création de la liste des fruits et légumes enregistrés
liste = liste_fruits_legumes()

# Simulation de la couleur du produit en composantes RVB de 0 à 255
RVB = extraire_rvb('FF8038')
rouge = RVB[0]
vert = RVB[1]
```

```
bleu = RVB[2]

# Calcul des distances (formule euclidienne de Pythagore)
distances = distances_euclidiennes(rouge, vert, bleu, liste)    #contient la
liste des distances entre les composantes du produit saisies et les produits de
la liste


# Détermination des K plus proches couleurs
k = 5

nb_de_produit_par_couleur = recherche_k_voisins(distances, liste, k)

#détermination de la classe du produit cad de sa couleur dominante
couleur_dominante = classe_produit(nb_de_produit_par_couleur, liste)

#affiche les produits de la couleur dominante
print("Produits de la même couleur dominante :")
for i in range(len(liste)):
    if liste[i][4] == couleur_dominante:
        print(liste[i][0], end = "--")
```

Remarques :

La liste retournée par la fonction distances_euclidiennes est : [[69, 0], [97, 1], [138, 2], [256, 3], [236, 4], [198, 5], [133, 6], [123, 7], [133, 8]]

La liste retournée après le tri est : [[69, 0], [97, 1], [123, 7], [133, 6], [133, 8], [138, 2], [198, 5], [236, 4], [256, 3]]

Pour les k plus proches voisins, on ne prend que les k premiers couples les plus proche du produit testé. A noter que cette liste est classée du plus proche au moins proche.

La structure de données retenue est une table :

```
[["tomate", 247, 75, 12, 0], ["poivron rouge", 225, 36, 63, 0], ["pomme Gala",
197, 11, 9, 0], ["courgette", 9, 191, 18, 1], ["poireau", 20, 132, 26, 1],
[« Pomme Granny Smith", 83, 221, 90, 1], ["banane", 233, 255, 23, 2], ["citron",
254, 240, 4, 2], ["raisin", 241, 253, 12, 2] ]
```

III. Le projet

Nous voulons modifier ce programme pour utiliser deux dictionnaires dont les structures sont les suivantes :

1) Un dictionnaire qui contient les couleurs dominantes (à partir d'une majorité des couleurs de la roue chromatique ci-contre) et trois ajouts : marron, blanc et noir.

```
dico_roue_chromatique = { "noir" : [0, 0, 0], "blanc" : [255, 255, 255]
, "bleu" : [0, 0, 255], "rouge" : [255, 0, 0] , "jaune" : [255, 255,
0], "orange" : [255, 165, 0] , "vert" : [0, 128, 0] , "violet" : [238,
130, 238] , "marron" : [88, 41, 0] }
```

2) Un dictionnaire contenant des références de fruits et de légumes avec leurs codes couleurs RVB et leur dominante

```
Dictionnaire_fl = {"tomate": {"rouge": 247, "vert": 75, "bleu": 12, "dominante":
"rouge"}, "poivron rouge": {"rouge": 225, "vert": 36, "bleu": 63, "dominante":
```

"rouge"}, "pomme gala": {"rouge": 197, "vert": 11, "bleu": 9, "dominante": "rouge"}, "orange": {"rouge": 225, "vert": 165, "bleu": 0, "dominante": "orange"}, "mandarine": {"rouge": 225, "vert": 165, "bleu": 0, "dominante": "orange"}, "potimarron": {"rouge": 225, "vert": 165, "bleu": 0, "dominante": "orange"}, "courgette": {"rouge": 9, "vert": 191, "bleu": 18, "dominante": "vert"}, "poireau": {"rouge": 20, "vert": 132, "bleu": 26, "dominante": "vert"}, "pomme Granny Smith": {"rouge": 83, "vert": 221, "bleu": 90, "dominante": "vert"}, "banane": {"rouge": 233, "vert": 255, "bleu": 23, "dominante": "jaune"}, "citron": {"rouge": 254, "vert": 240, "bleu": 4, "dominante": "jaune"}, "raisin": {"rouge": 241, "vert": 253, "bleu": 12, "dominante": "jaune"}, "céleri rave": {"rouge": 248, "vert": 251, "bleu": 217, "dominante": "blanc"}, "aubergine": {"rouge": 44, "vert": 16, "bleu": 5, "dominante": "noir"}, "champignon de Paris": {"rouge": 247, "vert": 235, "bleu": 224, "dominante": "blanc"}, "pomme de terre bio": {"rouge": 81, "vert": 60, "bleu": 0, "dominante": "marron"},"raisin_noir" : {"rouge": 36 , "vert": 35 , "bleu": 39 , "dominante": "noir" }, "brocoli" : {"rouge": 72 , "vert": 118 , "bleu": 59 , "dominante": "vert" }}

Dans un programme KNN, on répertorie d'abord tous les éléments les plus proches et ensuite on regarde la dominante majoritaire dans ces éléments retenus.

Dans ce projet, au lieu, de rechercher d'abord la liste de fruits et légumes proches, le programme recherche les couleurs dominantes les plus proches par la méthode des plus proches voisins (dans celles de la roue chromatique puis ensuite affiche les fruits et légumes potentiels).

On obtient une liste de tuples (clé : la dominante, distance : un entier)

Cette liste sera triée en utilisant la méthode de tri par sélection (en remplacement du tri par insertion du programme)

Reste ensuite à retrouver dans le dictionnaire_fl les fruits et légumes ayant ces dominantes

Par exemple, si l'on veut les deux couleurs dominantes voisines de la couleur dont le code est 'FF8038'

On trouve :

distance : 67 ---> dominante : orange ['orange', 'mandarine', 'potimarron']

distance : 138 ---> dominante : jaune ['banane', 'citron', 'raisin']

IV. Evaluation

Au niveau de l'évaluation, on tiendra compte des critères suivants :

1) Des traces écrites que vous aurez utilisées
2) Des explications sur les modifications faites
3) Des commentaires dans les programmes
4) De la « propreté » de la programmation
5) Des choix des tests