Exemple de question et de sa réponse dans un mini projet

D'après la question 5 du n6 p176

Ecrire un programme en Python qui :

- Pose 5 questions correspondant aux 5 critères rouge, orange, jaune, rond, allongé d'un fruit à deviner.
- Enregistre les réponses de l'utilisateur qui peuvent être :
  - Non répondu
  - Non
  - Oui
- Tient compte du **jeu de données** de la p176 pour deviner *le fruit le plus proche* (parmi ceux du jeu de données) de celui décrit par l'utilisateur au travers de ses réponses.

		Critères				
		Rouge	Orange	Jaune	Rond	Allongé
Catégories	Tomate	1	0	0	1	0
	Banane	0	0	1	0	1
	Citron	0	0	1	1	0
	Pomme	1	0	0	1	0
	Pêche	0	1	0	1	0

- L'interface avec l'utilisateur se fera dans la console de l'environnement Python.

  Exemple

```
>>> programme()

1. Le fruit est-il rouge ?
  Pour "Non répondu" tapez null,
  pour "Non" tapez 0, pour "Oui" tapez 1 : 0

2. Le fruit est-il orange ?
  Pour "Non répondu" tapez null,
  pour "Non" tapez 0, pour "Oui" tapez 1 : 1

3. Le fruit est-il jaune ?
  Pour "Non répondu" tapez null,
  pour "Non" tapez 0, pour "Oui" tapez 1 : 0

4. Le fruit est-il rond ?
  Pour "Non répondu" tapez null,
  pour "Non" tapez 0, pour "Oui" tapez 1 : 1

5. Le fruit est-il allonge ?
  Pour "Non répondu" tapez null,
  pour "Non" tapez 0, pour "Oui" tapez 1 : 0

 Réponse :  peche
```

Consigne : Après avoir lu et compris les six fonctions et leur organisation, dans **Spyder**, ouvrez un nouveau fichier. Enregistrez-le dans votre dossier personnel avec le nom *NOM_Prenom_fruits_Python.py*.
Saisissez dedans toutes les lignes de code Python sans les commentaires. Vérifiez le fonctionnement.

Le programme réalise l'enchainement de fonctions :

```
#  Systeme de reconnaissance de fruit
#  Auteur : Laurent Beaussart



# 1. Ecriture de la fonction distance de Hamming

def d_hamming(nb1, nb2):

    """
    Calcule la distance de Hamming (Richard)
    entre nb1 et nb2

    Parametres
    ----------
        nb1, nb2 : de type chaine de caracteres de meme longueur.
                   Ce sont les chaines dont on veut calculer la distance.

    Renvoie :
    ---------
        distance : de type entier
                C'est la distance de Hamming entre nb1 et nb2.

    """

    distance = 0  #  Initialisation
    for i in range(len(nb1)):
        if nb1[i] != nb2[i]:
            distance += 1
    return distance



# 2.  Construction a la main de la liste des listes [nombre, categorie].

    """
      [["10010", "tomate"], ["00101", "banane"], ...]
      On a les cinq critères : rouge, orange, jaune, rond, allonge.
      La categorie "tomate" correspond a : Vrai, Faux, Faux,
      Vrai, Faux soit "10010".
        La categorie "banane" correspond a : Faux, Faux, Vrai,
      Faux, Vrai soit "00101".

    """

liste = [["10010", "tomate"], ["00101", "banane"], ["00110", "citron"],\
         ["10010", "pomme"], ["01010", "peche"]]
```

```
#  3. Fonction calcule_distance entre l'individu à trouver et les categories
#    du jeu de donnees.

def calcule_distance(nb_inconnu):
    """
    Renvoie liste_distances qui est la liste de listes [distance, categories].

    Parametres :
    ------------
        nb_inconnu : de type chaine de caracteres
                     C'est la chaine qui donne les criteres
                     de l'individu inconnu. Par exemple '11001' si l'individu
                     est Rouge, Orange et allonge.

       liste : de type liste de listes de deux chaines de caracteres.
               C'est la liste du jeu de donnees
               [["10010", "tomate"], ["00101", "banane"], ...

               Ce n'est pas un paramètre nommé, mais elle est utilisee par
               la fonction.

    Renvoie :
    --------
    liste_distances : de type liste de la forme [[2, "tomate"], [0, "banane"]...
                         C'est la liste des listes [distance, categorie]
                    (distance de Hamming entre l'individu inconnu et les
                    categories du jeu de donnees).

    """


    liste_distances = []  # Initialisation

    for i in range(len(liste)):
        distance = d_hamming(nb_inconnu, liste[i][0])
        liste_distances.append([distance, liste[i][1]])

    return liste_distances
```

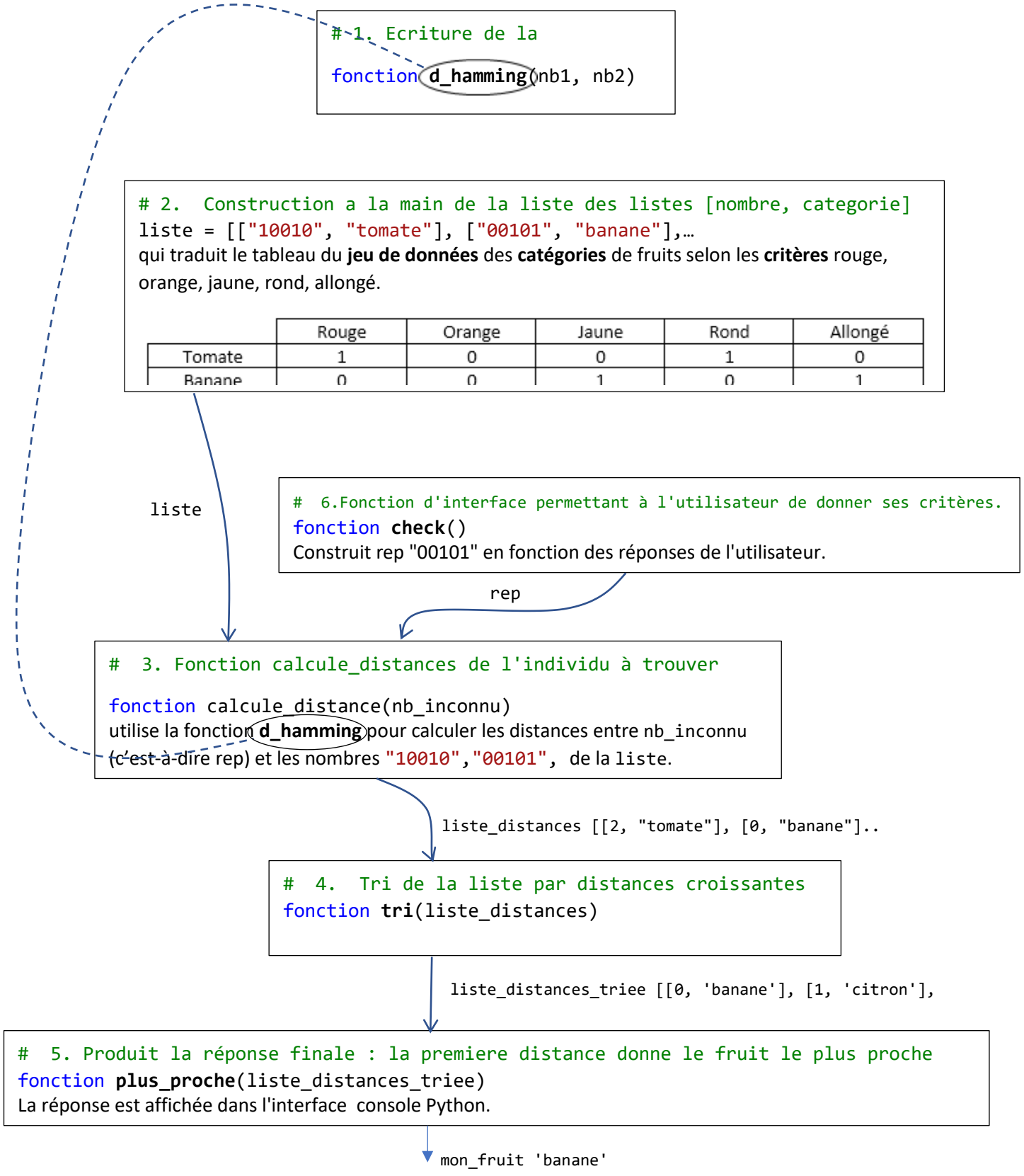
```
#  4. Fonction tri qui trie la liste des distances renvoyees par la fonction
#  calcule_distance par ordre croissant de distances.


import operator
#  La bibliotheque operator contient la fonction itemgetter utile pour
#  indiquer le numero de colonne a considerer pour trier un tableau.
#  Exemple liste_triee = sorted(liste, key=operator.itemgetter(0,1)) trie
#  le tableau par ordre croissant dans la colonne d'index 0 et, en cas
#  d'égalité, par ordre croissant dans la colonne d'index 1.

def tri(liste_distances):
    """
    Crée un nouveau tableau avec les memes elements que le
    tableau liste_distances mais en le triant.

    Parametres :
    ------------
        liste_distances : de type liste de listes à deux éléments.
        C'est le tableau renvoye par la fonction calcule_distance.
        Exemple : [[2, 'tomate'], [0, 'banane'], [1, 'citron'], ...]

    Renvoie :
    ---------
        liste_distances_triee : de type liste de listes à deux éléments.
        Exemple : [[0, 'banane'], [1, 'citron'], [2, 'tomate'], ...]

    """

    liste_distances_triee = sorted(liste_distances,key=operator.itemgetter(0,1))
    # Une copie de liste est creee puis la copie est triee.

    return liste_distances_triee
```

```python
#  5. Fonction qui renvoie le resultat sous la forme d'une chaine de caracteres.

def plus_proche(liste_distances_triee):
    """

    Parametres :
    ------------

    liste_distances_triees : de type liste de listes de [entier, chaine de c.].
    C'est la liste des listes [distance, "Fruit"] classees par ordre croissant
    des distances.
    Exemple : [[0, 'banane'], [1, 'citron'], [2, 'tomate'], ...


    Renvoie :
    ---------
        mon_fruit : de type chaine de caracteres.
    C'est le fruit le plus proche selon les critères entrés par le joueur.

    """


    le_fruit_le_plus_proche = liste_distances_triee[0][1]

    return le_fruit_le_plus_proche
```

```
#  6. Construire la réponse à partir des indications de l'utilisateur.
def check():
    """
    Pose 5 questions qui sont les 5 critères puis construit la reponse
    donnée par le joueur sous forme d'une chaine de caracteres avec null
    pour Non répondu, 0 pour Non et 1 pour Oui.
    Parametres :
    ------------
        Aucun. Les données sont fournies à la fonction directement par les
        instructions qu'elle contient.
        Exemple : 3. Le fruit est-il jaune ?
        Le joueur tape "null" pour Non répondu, 0 pour Non ou 1 pour Oui.
        Les questions sont mises dans la liste q.
        Dans la boucle for, les éléments de la liste q sont énumérés.
        Les questions apparaissent tour à tour au joueur grace a l'instruction
        input.
        input récupère la valeur r de chaque reponse donnee par l'utilisateur.
        Cette valeur r est concatenee a fin de la reponse rep.

    Renvoie :
    ---------
        rep : de type chaine de caracteres.
            C'est la concatenation des 5 chaines de caracteres ajoutees
            pendant la boucle for.
            Si le joueur repond "Non repondu" alors null est ajoute.
            Si le joueur repond "Non" alors 0 est ajoute.
            Si le joueur repond "Oui" alors 1 est ajoute.
        Exemple : Si le joueur répond Non, Non, Oui, Non, Oui alors rep
                  vaut "00101". La longueur de rep vaut 5.
                  Si au moins une question n'est pas repondue, alors "null"
                  remplace un "0" ou un "1" et la longueur de rep est
                  superieure a 5.
       Remarque : Cette fonction renvoie rep que si toutes les questions
                  ont ete repondues. Sinon elle repropose le questionnaire.
    """

    rep = ""

    while len(rep) != 5:
        rep = ""
        q = ["1. Le fruit est-il rouge ?",\
        "2. Le fruit est-il orange ?",\
        "3. Le fruit est-il jaune ?",\
        "4. Le fruit est-il rond ?",\
        "5. Le fruit est-il allonge ?"]

        #  Boucle for au cours de laquelle les 5 réponses sont recueillies.
        for question in q:
            r = input(question + '\n  Pour "Non répondu" tapez null, \n  pour "Non" tapez
0, pour "Oui" tapez 1 : ')
            rep = rep + r

    return rep
```

```
#  7.  Programme

def programme():
    """
    programme est la fonction "chef d'orchestre" qui appelle tour à tour les
    fonctions précédentes.

    Parametres :
    ------------
        Aucun : le programme est lancé depuis la console en saisissant
        programme()


    Renvoie :
    ---------
        Rien : Le fruit le plus proche ou le message d'alerte est affiché
        dans la console au moyen de la fonction Python print().


    """

    reponse_utilisateur = check()  #  reponse_utilisateur est une chaine de
                                   # caracteres, par exemple "00101"

    liste_distances = calcule_distance(reponse_utilisateur)#reponse_utilisateur
                                           # est le nombre inconnu a classer.

    liste_distances_triees = tri(liste_distances)


    mon_fruit = plus_proche(liste_distances_triees)

    print("\n Réponse : ", mon_fruit)
```

Schéma de l'enchainement des fonctions. Les numéros en vert indiquent dans quel ordre elles ont été *écrites*.




```
# 7. Le programme que lance l'utilisateur
fonction programme() exécute successivement :
   1. reponse_utilisateur = check()
   2. liste_distances = calcule_distance(reponse_utilisateur)
   3. liste_distances_triees = tri(liste_distances)
   4. mon_fruit = plus_proche(liste_distances_triees)
```