Exercice 3 (8 points)

Cet exercice porte sur le langage SQL, sur la programmation en Python et la recherche textuelle.

Le sujet d'une étude porte sur les papillons, la corrélation entre leur présence et celle de certaines plantes ainsi que sur la classification de nouvelles espèces.

Partie A. Corrélation avec la présence des plantes

Dans cet exercice, on pourra utiliser les clauses du langage SQL pour :

- construire des requêtes d'interrogation à l'aide de `SELECT`, `FROM`, `WHERE` (avec les opérateurs logiques `AND`, `OR`), `JOIN ... ON` ;
- construire des requêtes d'insertion et de mise à jour à l'aide de `UPDATE`, `INSERT`.

Dans le cadre de cette étude, une base de données `faune_flore.db` a été créée pour étudier la corrélation entre la présence d'espèces de papillons et celle de certaines plantes. Cette base de données regroupe les tables `papillon`, `plante` et `zone_geographique`.

La table `papillon` comporte les informations suivantes :

- l'identification du papillon (`num`) ;
- le nom commun du papillon (`nomCo`) ;
- le nom scientifique du papillon (`nomSc`) ;
- la taille moyenne du papillon en millimètres (`taille`) ;
- le principal habitat du papillon (`habitat`) ;
- la zone géographique où le papillon est le plus présent (`zone`). Cet attribut fait référence à l'attribut `num` de la table `zone_geographique`.

Un extrait de cette table est donné ci-après.

papillon					
num	nomCo	nomSc	taille	habitat	zone
458	Monarque	Danaus plexippu	100	Prairies	3
459	Citron de Provence	Gonepteryx cleopatra	30	Prairies	1
460	Paon-du-jour	Aglais io	6	Jardins	6
461	Machaon	Papilio machaon	85	Forêts	2
462	Petite Tortue	Aglais urticae	30	Prairies	5
463	Robert-le-Diable	Polygonia c-album	25	Forêts	4

La table `plante` comporte les informations suivantes :

- l'identification de la plante (`num`) ;
- le nom commun de la plante (`nomCo`) ;
- le nom scientifique de la plante (`nomSc`) ;
- le principal habitat de la plante (`habitat`) ;
- la zone géographique où elle est la plus présente (`zone`). Cet attribut fait référence à l'attribut `num` de la table `zone_geographique`.

Un extrait de la table `plante` est donné ci-dessous.

plante				
num	nomCo	nomSc	habitat	zone
128	Orchidée Phalaenopsis	Phalaenopsis	Forêts	5
129	Bambou	Bambusoideae	Forêts	3
130	Rose	Rosa	Haies	2
131	Lilas	Syringa	Haies	6
132	Coquelicot	Papaver rhoeas	Jardins	4
133	Lavande	Lavandula	Collines	1

La table `zone_geographique` contient les informations suivantes :

- l'identification de la zone géographique (`num`) ;
- le nom de la zone (`zone`).

Un extrait de la table `zone_geographique` est donné ci-après.

zone_geographique	
num	zone
1	Afrique du Nord
2	Amérique du Nord
3	Amérique du Sud
4	Asie
5	Asie du Sud
6	Europe

1. Donner la définition d'une clé primaire.

2. Expliquer pourquoi l'attribut `habitat` de la table `papillon` ne peut pas être une clé primaire.

3. Donner le résultat obtenu suite à l'exécution de la requête suivante si on l'applique sur l'extrait de table donné :

```
SELECT taille
FROM papillon
WHERE nomCo='Machaon'
```

Après avoir mesuré l'envergure de plusieurs papillons Petite Tortue, un des scientifiques de l'étude a calculé la nouvelle moyenne des tailles pour ce papillon, qui est maintenant de 50 mm.

4. Écrire une requête qui met à jour la table `papillon`, suite au calcul de cette nouvelle moyenne.

5. Écrire une requête qui affiche le nom commun de tous les papillons présents dans les prairies et dont la taille est strictement inférieure à 55 mm.

6. Écrire le résultat obtenu suite à l'exécution de la requête suivante si on l'applique sur les extraits des tables donnés.

```
SELECT nomSc
FROM plante
JOIN zone_geographique
ON plante.zone = zone_geographique.num
WHERE zone_geographique.zone = 'Asie'
```

7. Écrire une requête qui affiche le nom commun des papillons et celui des plantes qui se trouvent dans le même habitat et dont la taille des papillons est strictement inférieure à 55 mm.

8. Écrire le résultat obtenu suite à l'exécution de la requête suivante si on l'applique sur les extraits des tables donnés.

```
SELECT papillon.nomCo, plante.nomCo
FROM papillon
JOIN zone_geographique
ON papillon.zone = zone_geographique.num
JOIN plante
ON plante.zone = zone_geographique.num
WHERE zone_geographique.zone = 'Europe'
```

9. Écrire une requête qui affiche le nom commun des papillons qui se trouvent dans la même zone géographique que les coquelicots.

Partie B. Classification d'une nouvelle espèce

Les espèces de papillons sont regroupées dans une liste de dictionnaires. Pour simplifier, seuls les attributs `num` (l'identifiant du papillon), `nomCo` (son nom commun), `nomSc` (son nom scientifique) et `taille` (sa taille) seront considérés dans cette partie. Une partie de la liste `papillon` est donnée ci-dessous :

```
papillon = [
    {'num': 458, 'nomCo': 'Monarque',
     'nomSc': 'Danaus plexippu', 'taille': 100},
    {'num': 459, 'nomCo': 'Citron de Provence',
     'nomSc': 'Gonepteryx cleopatra', 'taille': 30},
    {'num': 460, 'nomCo': 'Paon-du-jour',
     'nomSc': 'Aglais io', 'taille': 6},
    {'num': 461, 'nomCo': 'Machaon',
     'nomSc': 'Papilio machaon', 'taille': 85},
    {'num': 462, 'nomCo': 'Petite Tortue',
     'nomSc': 'Aglais urticae', 'taille': 50},
    {'num': 463, 'nomCo': 'Robert-le-Diable',
     'nomSc': 'Polygonia c-album', 'taille': 25}
]
```

Le but de cette partie est de trier la liste des papillons par ordre croissant de taille et de classifier une nouvelle espèce photographiée.

La fonction `tri_collec` renvoie la liste de dictionnaires des papillons triée par ordre croissant de taille.

```
 1 def tri_collec(collec):
 2     """Renvoie la collection des papillons triées
 3     par ordre croissant de leur taille.
 4     Paramètre:
 5         collec : liste de dictionnaires des papillons
 6     Renvoie:
 7         liste triée par ordre croissant des tailles
 8         des papillons.
 9     """
10     for i in range(1, len(collec)):
11         pap = collec[i]
12         j = ...
```

```
13          while j > 0 and collec[j - 1]['taille'] > ...:
14              collec[j] = collec[j - 1]
15              j = ...
16          collec[j] = pap
17      return ...
```

10. Recopier et compléter les lignes 12, 13, 15 et 17 de la fonction `tri_collec`.

11. Nommer le tri utilisé.

12. Indiquer, en justifiant, parmi les propositions suivantes quel est le coût en temps de ce tri, dans le pire cas, pour un tableau de taille *n* : *linéaire*, *quadratique*, *logarithmique* ou *exponentiel*.

L'algorithme des *k* plus proches voisins est utilisé pour classifier la nouvelle espèce photographiée.

13. Expliquer brièvement le principe de cet algorithme.

Cette nouvelle espèce montre beaucoup de ressemblance avec l'espèce 'Aglais io' mais diffère par la taille et la couleur des motifs des ailes.

Pour vérifier l'hypothèse que la nouvelle espèce est l'espèce 'Aglais io' comportant une mutation génétique, une recherche naïve d'une séquence caractéristique des papillons 'Aglais io' est réalisée sur la chaîne d'ADN extraite de la nouvelle espèce. Une chaîne d'ADN est représentée en Python par une chaine de caractères. Cette recherche utilise la fonction `recherche_seq(seq, chaine)` qui renvoie l'indice du premier caractère de `seq` si la séquence `seq` est présente dans la chaîne d'ADN `chaine` et -1 sinon.

14. Recopier et compléter les lignes 15 et 17 de la fonction `recherche_seq`.

```
def recherche_seq(seq, chaine):
    """Renvoie l'indice du premier caractère de
       chaine où commence `seq` si la séquence `seq`
       se trouve dans la chaine de caractères chaine,
       -1 sinon
    Paramètres:
        seq : séquence à rechercher
        chaine : chaine d'ADN
    Renvoie:
        indice du premier caractère de seq dans
        la chaine, -1 sinon.
    """
    for i in range(len(chaine)-len(seq) + 1):
        j = 0
        while j < len(seq) and ...:
            j += 1
        if ...:
            return i
    return -1
```

La fonction `recherche_BMH(seq, chaine)`, donnée ci-dessous, implémente l'algorithme de Boyer-Moore Horspool.

```
def dico_lettres(seq):
    d = {}
    for i in range(len(seq)-1):
        d[seq[i]] = i
    return d

def recherche_BMH(seq, chaine):
    decalage = dico_lettres(seq)
    i = 0
    n = len(seq)
    while i <= len(chaine) - n:
        j = n-1
        while j >= 0 and chaine[i + j] == seq[j]:
            j -= 1
        if j == -1:
            return i
        else:
            if chaine[i + n - 1] in decalage:
                i += n - decalage[chaine[i + n-1]] - 1
            else:
                i += n
    return -1
```

15. Expliquer le principe de cet algorithme et son avantage par rapport à la fonction naïve `recherche_seq`.