Concours blanc Informatique 2023-24 - corrigé

Barème: 1 point pour chaque question.

Problème 1.

1.
```
SELECT COUNT(*) FROM Bouee WHERE nomSite = "Porquerolles";
```

2.
```
SELECT idTempete FROM Bouee B
JOIN Tempete T ON B.idBouee = T.idBouee
WHERE localisation = 'Mediterranee' AND finTempete < '2023-01-01';
```

3.
```
SELECT DISTINCT nomSite
FROM Bouee
WHERE idBouee IN (SELECT DISTINCT idBouee
                  FROM Tempete
                  WHERE debutTempete >= '2023-01-01');
```

4.
```
SELECT C.idCampagne, C.idBouee, COUNT(T.idTempete) AS Nombre_Tempetes
FROM Campagne C
JOIN Tempete T ON C.idBouee = T.idBouee
     AND T.debutTempete >= C.debutCampagne
     AND T.finTempete <= C.finCampagne
WHERE C.debutCampagne >= '2023-01-01' AND C.finCampagne <= '2023-12-31'
GROUP BY C.idCampagne;
```

5.
```
SELECT B.idBouee
FROM Bouee B
WHERE idBouee NOT IN (SELECT idBouee FROM Tempete
                      WHERE debutTempete >= '2023-01-01'
                      AND  finTempete <= '2023-12-31')
```

6.
```
import sqlite3

def get_Mesures(idBouee, date_debut, date_fin):
    # Connexion à la base de données
    conn = sqlite3.connect('OceanData.db')
    c = conn.cursor()

    # Exécution de la requête SQL pour récupérer les mesures
    c.execute("SELECT Date, Hauteur FROM Mesures
               WHERE idBouee = ? AND Date BETWEEN ? AND ?"
              , (idBouee, date_debut, date_fin))

    # Récupération des résultats
    mesures = c.fetchall()

    # Fermeture de la connexion
```

```
    conn.close()
    return mesures
```

7.
```
def interpolation_lagrange(X, Y, x):
    n = len(X)
    hauteur_interpolee = 0
    for i in range(n):
        hauteur_interpolee_i = Y[i]
        for j in range(n):
            if j != i:
                hauteur_interpolee_i *= (x - X[j]) / (X[i] - X[j])
        hauteur_interpolee += hauteur_interpolee_i
    return hauteur_interpolee
```

8.
```
def simulation_hauteur(tab_dates, tab_hauteurs, d_estim):
    estimation = interpolation_lagrange(tab_dates, tab_hauteurs, d_estim)
    return estimation
```

9.
```
date_debut_campagne = 1617097290
date_fin_campagne = 1617356490
idBouee = 455

tab_x = list(range(date_debut_campagne, date_fin_campagne + 1))
tab_mesures = get_Mesures(idBouee, date_debut_campagne, date_fin_campagne)
tab_dates = [mesure[0] for mesure in tab_mesures]
tab_hauteurs = [mesure[1] for mesure in tab_mesures]

tab_y = [simulation_hauteur(tab_dates, tab_hauteurs, date)
         for date in tab_x]
```

10.
```
import matplotlib.pyplot as plt

idBouee = 455
plt.plot(tab_x, tab_y)
plt.xlabel('Date')
plt.ylabel('Hauteur de la mer')
plt.title('Simulation de la variation de niveau de la mer')
plt.show()
```

Problème 2.

- Question 1:

```
def occurrences(texte):
    freqChar = {}
    for char in texte:
        if char in freqChar:
            freqChar[char] += 1
```

```
        else:
            freqChar[char] = 1
    return freqChar
```

- Questions 2, 3 et 4

```
class Arbre:
    def __init__(self, char='',frequence=0,noeudGauche=None,noeudDroit=None):
        self.char = char
        self.frequence = frequence
        self.noeudGauche = noeudGauche
        self.noeudDroit = noeudDroit

    def estFeuille(self):
        return self.noeudGauche is None and self.noeudDroit is None

    def __lt__(self, other):
        if self.frequence < other.frequence:
            return True
        elif self.frequence == other.frequence:
            return ord(self.char) < ord(other.char)
        else:
            return False
```

- Question 5:

```
def inserer(tab, A):
    for i, arbre in enumerate(tab):
        if A < arbre:
            tab.insert(i, A)
            return
    tab.append(A)
```

- Question 6:

```
def creerFeuilles(freqChar):
    feuilles = []
    # les feuilles triées par fréquence croissante,
    # puis par ordre ASCII croissant en cas d'égalité de fréquence.
    for char, freq in sorted(freqChar.items(),key=lambda x:(x[1],ord(x[0]))):
        feuilles.append(Arbre(char, freq))
    return feuilles
```

- Question 7:

```
def creer_arbre(tab):
    while len(tab) > 1:
        A = tab.pop(0)
        B = tab.pop(0)
        fusion = Arbre(frequence=A.frequence + B.frequence, noeudGauche=A,
                       noeudDroit=B)
        inserer(tab, fusion)
    return tab[0]
```

- Question 8:

```
def codeBinaire(arbre, dico={}, code=""):
    if arbre.estFeuille():
        dico[arbre.char] = code
    else:
        if arbre.noeudGauche:
            codeBinaire(arbre.noeudGauche, dico, code + '0')
        if arbre.noeudDroit:
            codeBinaire(arbre.noeudDroit, dico, code + '1')
    return dico
```

- Question 9:

```
def compresser(texte, dico):
    code_compressé = ""
    for char in texte:
        code_compressé += dico[char]

    # Calcul du taux de compression
    taille=len(texte) * 8 #taille en bits de la séquence initiale
    taille_compressée=len(code_compressé) #taille de la séquence compressée
    taux_compression = ((taille - taille_compressée) / taille) * 100

    print("Taux de compression:", "{:.2f}".format(taux_compression), "%")

    # Conversion de la chaîne binaire en format binaire Python
    code_compressé = "0b" + code_compressé

    return code_compressé
```

- Question 10:

```
def decompresser(code, dico):
    code_binaire = code[2:]  # enlever "0b"

    texte_original = ""
    code_temp = ""

    for bit in code_binaire:
        code_temp += bit
        for char, codage in dico.items():
            if codage == code_temp:
                texte_original += char
                code_temp = ""
                break

    return texte_original
```