DM 4 - Terminale NSI

Exercice

Cet exercice porte sur les algorithmes de tri et d'un parcours d'arbre binaire de recherche et sur la programation orientée objet

Contexte

Un club d'athlétisme organise une compétition inter-scolaire regroupant quatre disciplines :

- le 100 mètres ;
- le saut en longueur ;
- le lancer du poids ;
- le 1500 mètres.

Chaque athlète réalise une performance dans chacune de ces épreuves. Pour établir le classement final, un score global est calculé à partir de ses résultats selon des règles de conversion simples. L'objectif du programme est de modéliser ces athlètes et d'implémenter différentes méthodes de tri pour les classer du premier au dernier.

Le score d'un athlète est la somme des points obtenus dans chaque épreuve. Ces points s'obtiennent selon les règles suivantes :

- points du **100 m** : $(20 - t) * 10$ avec t le temps en secondes ;
- points du **saut en longueur** : $d * 20$ avec d la distance en mètres ;
- points du **lancer de poids** : $d * 10$ avec d la distance en mètres ;
- points du **1500 m** : $(500 - t) * 1$ avec t le temps en secondes.

Partie A : Dictionnaire et tri

1. L'athlète **Alex** a réalisé les performances suivantes :
   - **100 m** : 13.0 s ;
   - **saut en longueur** : 5.2 m ;
   - **lancer de poids** : 9.0 m ;
   - **1500 m** : 310.0 s.

   Calculer le score total de l'athlète.

Pour faire le calcul des points obtenus à une épreuve, on dispose de la fonction `nb_points` ci-dessous :

```
1 def nb_points(epreuve, valeur):
2     points = 0
3     if epreuve == '100m':
4         points = (20 - valeur) * 10
```

```
 5      elif epreuve == 'longueur':
 6          ...
 7      elif epreuve == 'poids':
 8          ...
 9      elif epreuve == '1500m':
10          ...
11      return ...
```

2. Recopier et compléter les lignes 6, 8, 10 et 11 de la fonction `nb_points`.

Les performances d'un athlète sont regroupées dans un dictionnaire dont les clés sont les différentes épreuves. Chaque athlète est représenté par un dictionnaire qui stocke son nom, ses performances et son score. Tous les athlètes sont regroupés dans une liste.

On donne ci-dessous un exemple d'une liste d'athlètes avant le calcul des scores.

```
l_athletes = [
    {'nom': 'Alex',
     'performances': {'100m': 13.0, 'longueur': 5.2, 'poids':
9.0, '1500m': 310.0},
     'score': 0
    },
    {'nom': 'Anna',
     'performances': {'100m': 12.2, 'longueur': 5.8, 'poids':
8.5, '1500m': 230.0},
     'score': 0
    },
    {'nom': 'Rayan',
     'performances': {'100m': 13.5, 'longueur': 5.0, 'poids':
9.5, '1500m': 205.0},
     'score': 0
    }
]
```

Pour faire le calcul du score obtenu par un athlète, on dispose de la fonction `score` ci-dessous :

```
1 def score(athlete):
2     total = 0
3     performances = athlete[...]
4     for epreuve in performances:
5         valeur = performances[...]
6         total += ...
7     athlete['score'] = total
```

3. Recopier et compléter les lignes 3, 5 et 6 de la fonction `score`.

Pour obtenir le classement des athlètes du plus grand score au plus petit score, on réorganise la liste `l` des athlètes avec la fonction `classer(l)` qui prend en paramètre la liste `l` d'athlètes :

```
 1 def classer(l):
 2     n = len(l)
 3     for i in range(n):
 4         max_index = i
 5         for j in range(i + 1, n):
 6             if ...:
 7                 max_index = j
 8         temp = l[i]
 9         l[i] = l[max_index]
10         l[max_index] = temp
```

4. Recopier et compléter la ligne 6 de la fonction `classer`.

5. Parmi les propositions suivantes, choisir celle qui correspond au type de tri effectué par la fonction `classer`.

   - tri par insertion
   - tri par sélection
   - tri à bulles
   - tri fusion

6. Donner le coût de la fonction `classer`, c'est à dire le nombre de comparaisons d'éléments de la liste, en fonction de la taille de la liste.

Partie B : Arbre binaire de recherche

Une nouvelle approche est adoptée, fondée sur la programmation orientée objet et la mise en œuvre d'un arbre binaire de recherche, afin de modéliser les athlètes et de produire leur classement automatique en fonction de leur score.

```
 1 class Athlete:
 2     def __init__(self, nom, m100, longueur, poids, m1500):
 3         self.nom = nom
 4         self.m100 = m100
 5         self.longueur = longueur
 6         self.poids = poids
 7         self.m1500 = m1500
 8         self.score = self.calculer_score()
 9
10     def calculer_score(self):
11         ...
```

7. Créer l'objet `alex` qui modélise l'athlète **Alex** avec ses performances données à la question 3

8. Écrire le code de la méthode `calculer_score()`.

Une fois créés, les athlètes sont insérés dans un arbre binaire de recherche. Chaque athlète y constitue un nœud, positionné selon son score de manière à assurer la propriété suivante : pour chaque athlète, tous les athlètes de son sous-arbre gauche

ont un score strictement plus petit et tous les athlètes de son sous-arbre droit un score plus grand ou égal.

9. Dessiner l'arbre obtenu après l'insertion, dans l'ordre, des six athlètes suivants.

Ordre	Nom	Score
1	Martin	354
2	Lena	351
3	Rayan	350
4	Yanis	355
5	Ninon	351
6	Ana	356

La classe `Noeud` qui assure l'organisation structurée des athlètes dans un arbre binaire de recherche.

```
class Noeud:
    def __init__(self, athlete):
        self.valeur = athlete
        self.gauche = None
        self.droite = None

    def inserer(self, athlete):
        if athlete.score < self.valeur.score:
            if self.gauche == ...:
                self.gauche = Noeud(athlete)
            else:
                ...
        else:
            if self.droite == ...:
                self.droite = Noeud(athlete)
            else:
                ...
```

10. Recopier et compléter le code de la méthode `inserer`.

On ajoute la méthode ci-dessous à la classe `Noeud`, qui permet de classer les athlètes de manière décroissante.

```
    def classer(self, classement=None):
        if classement is None:
            classement = []
        if self.droite is not None:
            self.droite.classer(classement)
        classement.append(self.valeur.nom)
```

```
7           if self.gauche is not None:
8               self.gauche.classer(classement)
9           return classement
```

11. Expliquer quel type de parcours est mis en œuvre dans la méthode `classer`, et préciser dans quel ordre sont explorés les nœuds et comment cela permet de classer les athlètes en fonction de leur score.

12. Donner un avantage et un inconvénient des deux approches utilisées pour classer les athlètes :

    - Utiliser des dictionnaires et un tri
    - Utiliser un arbre binaire de recherche approche objet avec insertion dans un arbre binaire