DM 4 Correction

2026 sujet 05 — Correction de l'exercice 1

Exercice sur les algorithmes de tri, les dictionnaires, la programmation orientée objet et les arbres binaires de recherche.

Question 1

Pour Alex :

- 100 m : `(20 - 13.0) * 10 = 70`
- saut en longueur : `5.2 * 20 = 104`
- lancer de poids : `9.0 * 10 = 90`
- 1500 m : `(500 - 310.0) * 1 = 190`

Score total : `70 + 104 + 90 + 190 = 454`.

Question 2

```
def nb_points(epreuve, valeur):
    points = 0
    if epreuve == '100m':
        points = (20 - valeur) * 10
    elif epreuve == 'longueur':
        points = valeur * 20
    elif epreuve == 'poids':
        points = valeur * 10
    elif epreuve == '1500m':
        points = (500 - valeur) * 1
    return points
```

Question 3

```
def score(athlete):
    total = 0
    performances = athlete['performances']
    for epreuve in performances:
        valeur = performances[epreuve]
        total += nb_points(epreuve, valeur)
    athlete['score'] = total
```

Question 4

```
def classer(l):
    n = len(l)
    for i in range(n):
        max_index = i
        for j in range(i + 1, n):
            if l[j]['score'] > l[max_index]['score']:
                max_index = j
        temp = l[i]
        l[i] = l[max_index]
        l[max_index] = temp
```

Question 5

La fonction `classer` effectue un **tri par sélection**.

Question 6

Pour une liste de taille `n`, le nombre de comparaisons est :

`(n - 1) + (n - 2) + ... + 1 = n * (n - 1) / 2`

Le coût est donc de l'ordre de n^2.

Question 7

```
class Athlete:
    def __init__(self, nom, m100, longueur, poids, m1500):
        self.nom = nom
        self.m100 = m100
        self.longueur = longueur
        self.poids = poids
        self.m1500 = m1500
        self.score = self.calculer_score()

    def calculer_score(self):
        total = 0
        total += nb_points('100m', self.m100)
        total += nb_points('longueur', self.longueur)
        total += nb_points('poids', self.poids)
        total += nb_points('1500m', self.m1500)
        return total

alex = Athlete('Alex', 13.0, 5.2, 9.0, 310.0)
```

Question 8

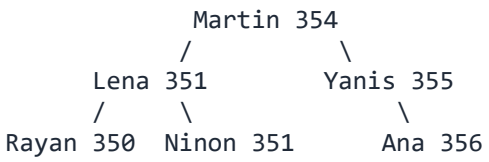
La méthode `calculer_score` est donnée dans la cellule précédente.

Question 9

Après insertion dans l'ordre :

1. Martin, score 354
2. Lena, score 351
3. Rayan, score 350
4. Yanis, score 355
5. Ninon, score 351
6. Ana, score 356

L'arbre obtenu est :



Question 10

```
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 == None:
                self.gauche = Noeud(athlete)
            else:
                self.gauche.inserer(athlete)
        else:
            if self.droite == None:
                self.droite = Noeud(athlete)
            else:
                self.droite.inserer(athlete)

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

Question 11

La méthode `classer` réalise un **parcours infixe inversé**.

Elle parcourt :

1. le sous-arbre droit ;
2. le nœud courant ;
3. le sous-arbre gauche.

Comme dans l'arbre binaire de recherche les scores les plus grands sont à droite, ce parcours donne les athlètes dans l'ordre décroissant des scores.

Question 12

Avec les dictionnaires et un tri, l'avantage est que la structure est simple et facile à comprendre. L'inconvénient est que le tri doit parcourir plusieurs fois la liste et coûte ici de l'ordre de n^2.

Avec un arbre binaire de recherche, l'avantage est que les athlètes peuvent être insérés progressivement et qu'un parcours donne directement le classement. L'inconvénient est que l'arbre peut devenir déséquilibré selon l'ordre d'insertion, ce qui peut dégrader les performances.

Script complet de test

```
def nb_points(epreuve, valeur):
    points = 0
    if epreuve == '100m':
        points = (20 - valeur) * 10
    elif epreuve == 'longueur':
        points = valeur * 20
    elif epreuve == 'poids':
        points = valeur * 10
    elif epreuve == '1500m':
        points = (500 - valeur) * 1
    return points

def score(athlete):
    total = 0
    performances = athlete['performances']
    for epreuve in performances:
        valeur = performances[epreuve]
        total += nb_points(epreuve, valeur)
    athlete['score'] = total

def classer(l):
    n = len(l)
    for i in range(n):
        max_index = i
        for j in range(i + 1, n):
            if l[j]['score'] > l[max_index]['score']:
                max_index = j
        temp = l[i]
        l[i] = l[max_index]
        l[max_index] = temp

class Athlete:
    def __init__(self, nom, m100, longueur, poids, m1500):
        self.nom = nom
        self.m100 = m100
        self.longueur = longueur
        self.poids = poids
        self.m1500 = m1500
        self.score = self.calculer_score()
```

```
    def calculer_score(self):
        total = 0
        total += nb_points('100m', self.m100)
        total += nb_points('longueur', self.longueur)
        total += nb_points('poids', self.poids)
        total += nb_points('1500m', self.m1500)
        return total

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 == None:
                self.gauche = Noeud(athlete)
            else:
                self.gauche.inserer(athlete)
        else:
            if self.droite == None:
                self.droite = Noeud(athlete)
            else:
                self.droite.inserer(athlete)

    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)
        if self.gauche is not None:
            self.gauche.classer(classement)
        return classement

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}
]

for athlete in l_athletes:
    score(athlete)

classer(l_athletes)
l_athletes
```