Correction 26-NSIJ1G11

Exercice 1

Partie A

1. Le score est de $(20 - 13,0) \times 10 + 5,2 \times 20 + 9,0 \times 10 + (500 - 310,0) \times 1 = 454$ points.

2.
```python
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
```

3.
```python
def score(athlete):
    total = 0
    performances = athlete['performances']
    for epreuve in performances:
        valeur = performances[epreuve]
        total += nb_points(epreuve, valeur)
    athlete['score'] = total
```

4.
```python
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
```

5. C'est un tri par sélection. On parcourt la liste des athlètes, on trouve celui qui a le meilleur score et on le place en premier, puis on recommence pour les autres.

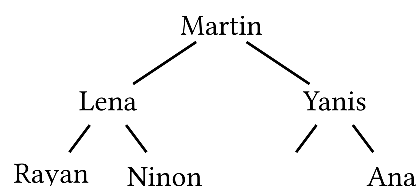
6. En notant n la taille de la liste, le nombre de comparaisons est de $n - 1 + n - 2 + \ldots + 1 = \frac{n(n-1)}{2}$, soit un ordre de grandeur en $O(n^2)$ (c'est une complexité quadratique).

Partie B

7.
```python
alex = Athlete('Alex', 13.0, 5.2, 9.0, 310.0)
```

8.
```python
  def calculer_score(self):
      points = (20 - self.m100) * 10
      points += self.longueur * 20
      points += self.poids * 10
      points += (500 - self.m1500) * 1
      return points
```

9.



Sous licence CC BY-NC-SA 4.0

10.
```python
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)
```

11. La méthode `classer` utilise un parcours en profondeur infixe pour classer les athlètes. Elle parcourt d'abord le sous-arbre gauche (appel résursif de la ligne 5), puis le noeud courant, puis le sous-arbre droit (appel résursif de la ligne 8). Cela permet de classer les athlètes dans l'ordre croissant de leur score, car les athlètes avec un score plus faible sont placés à gauche et ceux avec un score plus élevé sont placés à droite.

12. Pour l'approche dictionnaires et tri:
    - Avantage : Facile à implémenter et à comprendre (?)
    - Inconvénient : La complexité du tri par sélection est de $O(n^2)$, ce qui peut être inefficace pour de grandes listes d'athlètes.

    Pour l'approche arbre binaire de recherche:
    - Avantage : Permet une insertion rapide de nouveaux athlètes, car il suffit de les insérer à la bonne position dans l'arbre (complexité moyenne en $O(\log(n))$).
    - Inconvénient : Peut devenir déséquilibré si les athlètes sont insérés dans un ordre particulier, ce qui peut affecter les performances de l'insertion et du classement.

Exercice 2

1. Le message déchiffré est `CODE`.
2. Avec la clé `SECURITY1024`, la grille de chiffrement est :

	1	**2**	**3**	**4**	**5**	**6**
1	S	E	C	U	R	I
2	T	Y	1	0	2	4
3	A	B	D	F	G	H
4	J	K	L	M	N	O
5	P	Q	V	W	X	Z
6	3	5	6	7	8	9

   Ainsi, `BAC` va se chiffrer en `(3,2)(3,1)(1,3)`.

3. Le chiffrement de Polybe peut être qualifié de symétrique, car la même clé est utilisée pour le chiffrement et le déchiffrement. En effet, pour déchiffrer un message chiffré avec la grille de chiffrement, il suffit d'utiliser la même grille pour retrouver les lettres correspondantes aux coordonnées chiffrées.
4. Le résultat sera `AXU7BCDEFGHIJKLMNOPQRSTVWYZ012345689`.
5.
```python
def grille_vide(n):
    return [['' for _ in range(n)] for _ in range(n)]
```

Sous licence CC BY-NC-SA 4.0

6.
```python
def generer_grille(cle):
    ordre_insertion = generer_ordre(cle)
    grille = grille_vide(6)
    indice = 0
    for i in range(6):
        for j in range(6):
            grille[i][j] = ordre_insertion[indice]
            indice = indice + 1
    return grille
```

7.
```python
def dechiffrer(cle, message):
    resultat = ''
    grille = generer_grille(cle)
    for t in message:
        resultat = resultat + grille[t[0]-1][t[1]-1]
    return resultat
```

8.
```python
def generer_dico(cle):
    dico = {}
    grille = generer_grille(cle)
    for i in range(6):
        for j in range(6):
            dico[grille[i][j]] = (i+1, j+1)
    return dico
```

9.
```python
def chiffrer(cle, message):
    resultat = []
    dico = generer_dico(cle)
    for t in message:
        resultat.append(dico[t])
    return resultat
```

10. Un algorithme de chiffrement symétrique utilise la même clé pour le chiffrement et le déchiffrement, tandis qu'un algorithme de chiffrement asymétrique utilise une paire de clés : une clé publique pour le chiffrement et une clé privée pour le déchiffrement. Dans un algorithme de chiffrement asymétrique, la clé publique peut être partagée librement, tandis que la clé privée doit être gardée secrète.

Exercice 3

Partie A

1 et 2.

```python
class Demineur :
    def __init__(self, hauteur, largeur, pourcentage_mines):
        '''
        Pour un pourcentage de 15,6 on donnera au paramètre
        pourcentage_mines la valeur de 0.156.
        '''
        assert 0.1 <= pourcentage_mines <= 0.3, 'Le pourcentage de mines doit être compris
entre 10% et 30%.'
        self.hauteur = hauteur
        self.largeur = largeur
        self.pourcentage_mines = pourcentage_mines
```

3.
```python
demineur_intermediaire = Demineur(16, 16, 0.156)
```

Partie B

4.
```python
    def grille_demineur_vide(self):
        return [[0 for _ in range(self.largeur)] for _ in range(self.hauteur)]
```

5.
```
def placer_mines(self):
    '''
    Cette fonction doit placer de façon aléatoire les mines dans la grille.
    Une mine sera représentée par le nombre -1.
    La méthode permet de mettre à jour l'attribut grille_demineur
    '''
    compteur_mines = 0 # Nombre de mines placées dans la grille.
    # nombre_bombes contient le nombre de mines à placer dans la grille.
    nombre_bombes = self.largeur * self.hauteur * self.pourcentage_mines
    while compteur_mines < nombre_bombes:
        ligne = randint(0, self.hauteur - 1)
        colonne = randint(0, self.largeur - 1)
        if self.grille_demineur[ligne][colonne] == 0:
            self.grille_demineur[ligne][colonne] = -1
            compteur_mines += 1
```

6.
```
def nombre_voisines_avec_mines(self, coordonnees_case):
    '''
    La méthode nombre_voisines_avec_mines renvoie le nombre de cases voisines
contenant une mine à la case dont les coordonnées sont passés en
paramètre(coordonnees_case) à la fonction.
    '''
    liste_voisines = self.voisines(coordonnees_case)
    nombre_mines = 0
    for case in liste_voisines:
        if self.grille_demineur[case[0]][case[1]] == -1:
            nombre_mines += 1
    return nombre_mines
```

7.
```
def generer_demineur(self):
    # Il faut penser à appeler la méthode placer_mines !
    self.placer_mines()
    for i in range(self.hauteur):
        for j in range(self.largeur):
            if self.grille_demineur[i][j] != -1:
                self.grille_demineur[i][j] = self.nombre_voisines_avec_mines((i, j))
```

Partie C

6.
```
def visibilite(self, coordonnees_case):
    if self.grille_demineur[coordonnees_case[0]][coordonnees_case[1]] == -1:
        for i in range(self.hauteur):
            for j in range(self.largeur):
                self.grille_visibilite[i][j] = True
    elif self.grille_demineur[coordonnees_case[0]][coordonnees_case[1]] == 0:
        self.grille_visibilite[coordonnees_case[0]][coordonnees_case[1]] = True
        for case in self.voisines(coordonnees_case):
            if not self.grille_visibilite[case[0]][case[1]]:
                self.visibilite(case)
    else:
        self.grille_visibilite[coordonnees_case[0]][coordonnees_case[1]] = True
```

Partie D

7. Les clés étrangères sont `joueur`, faisant référence à `id_joueur` de la table `Joueur`, et `niveau`, faisant référence à `niveau` de la table `Demineur`.

8. Une solution possible est :

```
SELECT niveau, score FROM Meilleur_score
WHERE joueur = "2";
```

9.
```
UPDATE Joueur
SET mot_de_passe= "cGhhxDE4"
WHERE pseudo = "Kirna";
```

10. Le résultat de la requête est :

    `"Raptor"`

    .

11. Une solution est de créer un nouveau niveau `débutant` dans `Demineur`, mettre à jour `Meilleur_score` et enfin supprimerle niveau `facile` de la table `Demineur`.

```
INSERT INTO Demineur
VALUES ("débutant", "8x8", 15.6);

UPDATE Meilleur_score
SET niveau = "débutant"
WHERE niveau = "facile";

DELETE FROM Demineur
WHERE niveau = "facile";
```