Proposition de correction

Exercice 1

Partie A

Q1

une clé primaire doit identifier un enregistrement de façon unique. Or un artiste peut créer différents albums.

Q2

Nightwish

The Rasmus

Q3

1986

2001

1986

Q4

```
UPDATE CD
SET Annee = 2000
WHERE id_album = 4
```

Q5

```
SELECT CD.Titre_album
FROM CD
JOIN Rangement
ON CD.id_album =  Rangement.id_album
WHERE Rangement.Numero_etagere = 1
ORDER BY CD.Titre_album
```

Q6

il doit d'abord supprimer les clés étrangères pour éviter une erreur référentielle

1. dans la table Rangement :

   ```
   DELETE FROM Rangement
   WHERE id_album = 5
   ```

2. puis dans CD :

   ```
   DELETE FROM CD
   WHERE id_album = 5
   ```

3. et enfin dans Artiste :

   DELETE FROM Artiste

   WHERE Nom_artiste = 'The Rasmus'

Partie B

Q7

Méthode de cryptographie qui utilise la même clé pour le chiffrement et le déchiffrement.

Q8

Méthode de cryptographie qui utilise une clé publique (pour chiffrer) distribué à tout le monde et une clé privée (pour déchiffrer) gardée secrète par le propriétaire.

Q9

1. Le serveur chiffre la clé C avec la clé publique de Bob
2. Le serveur envoie la clé C chiffrée à Bob
3. Bob déchiffre la clé C avec sa clé privée

Exercice 2

Partie A

Q1

```
class Marchandise:
    def __init__(self, p: int, v: int) -> 'Marchandise':
        assert v > 0, "erreur de volume"
        self.prix   = p
        self.volume = v
```

Q2

```
m1 = Marchandise(20, 7)
```

Q3

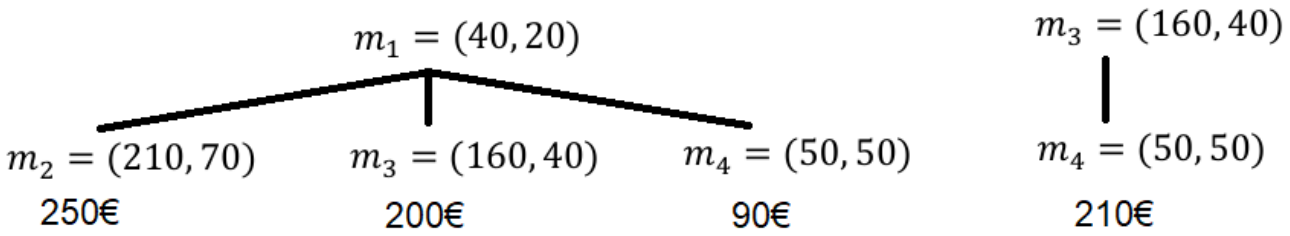
```
def ratio(self) -> float:
    return self.prix / self.volume
```

Q4

```
def prixListe(tab: list) -> int:
    return sum([m.prix for m in tab])
```

Partie B

Q5



- (m1, m2), (m1, m3), (m1, m4), (m3, m4)
- 250€ (m1, m2)

Q6

Algorithme glouton

Q7

```
def tri(tab: list) -> None:
    n = len(tab)
    for i in range(1, n):
        marchandise = tab[i]
        j = i-1
        while j >= 0 and marchandise.ratio() > tab[j].ratio():
            tab[j+1] = tab[j]
            j = j-1
        tab[j+1] = marchandise
```

Q8

- tri par insertion
- complexité quadratique $O(n^2)$

Q9

```
def charge(tab: list, volume: int) -> list:
    tri(tab)
    chargement = []
    n = len(tab)
    for i in range(n):
        if tab[i].volume <= volume:
            chargement.append(tab[i])
            volume -= tab[i].volume
    return chargement
```

Partie C

Q10

```
def chargeOptimale(tab: list, v_restant: int, i: int) -> list:
    if i >= len(tab):
        return []
    else:
        if tab[i].volume > v_restant:
            return chargeOptimale(tab, v_restant, i+1)
        else:
            option1 = chargeOptimale(tab, v_restant, i+1)
            option2 = [tab[i]] + chargeOptimale(tab, v_restant - tab[i].volume, i+1)
            if prixListe(option1) > prixListe(option2):
                return option1
            else:
                return option2
```

Exercice 3

Partie A

Q1

- nom : str
- denivele : int
- longueur : float
- couleur : str
- ouverte : bool

Q2

```
def set_couleur(self):
        if self.denivele >= 100:
            self.couleur = 'noire'
        elif self.denivele >= 70:
            self.couleur = 'rouge'
        elif self.denivele >= 40:
            self.couleur = 'bleue'
        else:
            self.couleur = 'verte'
```

Q3

Proposition D

Q4

```
def fermeture_pistes(self):
    for piste in self.pistes:
        if piste.couleur == 'verte':
            piste.ouverture = False
```

Q5

```
def pistes_de_couleur(pistes : list, couleur : str) -> list:
    return [p.nom for p in pistes if p.get_couleur() == couleur]
```

Q6

```
def semi_marathon(L : list) -> bool:
    distance = 0.
    liste_pistes = lievre_blanc.get_pistes()
    for nom in L:
        for piste in liste_pistes:
            if piste.get_nom() == nom:
                distance = distance + piste.get_longueur()
    return distance > 21.1
```

Partie B

Q7

domaine['E']['F']

Q8

```
def voisins(G : dict, s : str) -> list:
    return [s for s in G[s]]
```

Q9

```
def longueur_chemin(G : dict, chemin : list) -> float:
    precedent = chemin[0]
    longueur  = 0
    for i in range(1, len(chemin)):
        longueur  = longueur + domaine[precedent][chemin[i]]
        precedent = chemin[i]
    return longueur
```

Q10

La fonction s'appelle elle-même

Q11

```
def parcours_dep_arr(G : dict, depart : str, arrivee : str) -> list:
    liste = parcours(G, depart)
    return [chemin for chemin in liste if chemin[-1] == arrivee]
```

Q12

```
def plus_court(G : dict, depart : str, arrivee : str) -> list:
    liste_chemins = parcours_dep_arr(G, depart, arrivee)
    chemin_plus_court = liste_chemins[0]
    minimum = longueur_chemin(G, chemin_plus_court)
    for chemin in liste_chemins:
        longueur = longueur_chemin(G, chemin)
        if longueur < minimum:
            minimum = longueur
            chemin_plus_court = chemin
    return chemin_plus_court
```

Q13

Le chemin le plus court n'est pas le plus rapide car l'algorithme ne tient pas compte du dénivelé.