Proposition de correction

Exercice 1

Q1.a

Clef étrangère : pointe sur une clef primaire d'une table externe

Q1.b

permet d'établir des relations entre les tables film et acteur

Q2.a

```
SELECT titre, annee
FROM Film
ORDER BY annee, titre
```

Q2.b

```
INSERT INTO Acteur
VALUES(1250, 'YOUSSOUF', 'Fathia')
```

Q3

La table Joue_role possède une valeur de sa clé étrangère Idacteur qui pointe sur l'enregistrement à supprimer

Q4

```
SELECT Realisateur.realisateur_nom, Realisateur.realisateur_prenom
FROM Realisateur, Film
WHERE Film.titre = 'Léon'
AND Film.Idrealisateur = Realisateur.Idrealisateur
```

Q5

```
SELECT Acteur.acteur_nom
FROM Acteur, Film, Joue_role
WHERE Film.titre = 'Intouchables'
AND Joue_role.Idfilm = Film.Idfilm AND Joue_role.Idacteur = Acteur.Idacteur
ORDER BY Acteur.acteur_nom
```

Exercice 2

Partie 1

Q1

8 Gio

Q2

stockage de données (ROM)

Q3

consommation énergétique plus faible

coût plus faible

Partie 2

Q1

192.186.254.0

Q2

256 – 2 – 2 = 252

Q3

192.168.254.2

Partie 3

Q1

8A:FD:54:49:D0:CC

Q2

UDP

Q3

192.168.254.201

Partie 4

Q1

Table de routage du routeur R4		
Destination	**Routeur suivant**	**Distance**
R1	R2	2
R2	R2	1
R3	R2	2
R5	R6	2

R6	R6	1

Q2

R1 → R2 → R4 → R6

Exercice 3

Partie A

Q1.a

A

Q1.b

ouverture fichier en lecture (read)

Q2.a

list (de dict)

Q2.b

dict

{ 'dep': '56', 'nom': 'Vannes', 'nb_hab_1999': '51759', 'nb_hab_2012': '53000', 'densite': '1625', 'superficie': '32.3', 'alt_min': '0', 'alt_max': '56' }

Q2.c

```
print(listeVilles[5]['nb_hab_2012'])
```

Partie 2

Q1

Une liste de tuple (nom, altitude max) des villes dont l'altitude max est > 100 m

[('Pontivy', 192), ('Saint-Avé', 136)]

Q2

```
def variations(listeV : list) -> int :
    """
    @param        listV – liste de dictionnaire de villes
    @return       différence cumulée entre le nombre d'habitants en 2012 et 1999 des villes
    """
    difference = 0
    for ville in listeV :
            difference += ville['nb_hab_2012'] - ville['nb_hab_1999']
    return difference
```

Q3

```
def surmax(listeV : list) -> str:
```

```
    """
    @param      listV – liste de dictionnaire de villes
    @return     ville ayant la superficie la plus grande (pas de superficies identiques)
    """
    nom = ''
    if len(listeV) :    # attention si fichier < 2 villes
            surface_max = listeV[0]['superficie']
            nom = listeV[0]['nom']
    for ville in range(1, len(listeV)) :
            if surface_max < listeV[ville]['superficie'] :
                    surface_max = listeV[ville]['superficie']
                    nom = listeV[ville]['nom']
    return nom
```

Exercice 4

Q1.a

3

Q1.b

```
def hauteur(A) :
    if estVide(A):
            return -1
    else :
            g = hauteur(gauche(A))
            d = max(g, hauteur(droite(A)))
    return 1 + d
```

Q1.c

int

Q2.a

['AnneB', 'Pedro', 'FredB', 'Sophia', 'Malik', 'Marc', 'AstridM', 'KevinH', 'Nico']

Q2.b

préfixe

Q3.a

'FredB', 'Marc', 'KevinH', 'Nico'

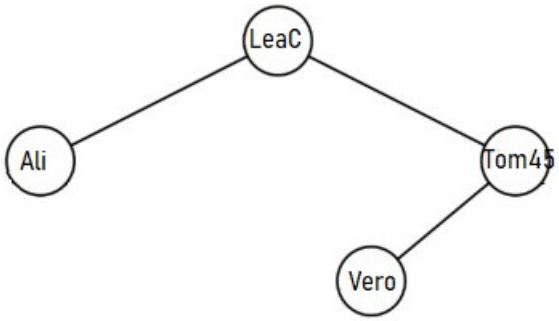
Q3.b

C

Q3.c

```
def profil(A : object) -> str :
    """renvoie le profil du membre dont le pseudonyme est l'étiquette de la racine de A,
    sous la forme d'une chaîne de caractères : 'membre_or', 'membre_argent'
    ou 'membre_bronze'."""
    if hauteur(gauche(A)) :
            if hauteur(droite(A)) :
                    return 'membre_or'
    elif hauteur(droite(A)) :
            return 'membre_argent'
    else :
            return 'membre_bronze'
```

Q4



Q5

```
def cotisation(A : object) ->  int :
    """renvoie le montant total des cotisations reçues par le club"""
    total = 0
    membres = [ ]
    membres_profiles(A, membres)
    for (_, profil) in membres :
            if profil == 'membre_or' :
                    total += 20
            elif profil == 'membre_argent' :
                    total += 30
            else :
                    total += 40
    return total
```

Exercice 5

Partie 1

Q1

```
def crediter(self, montant) :
    """Ajoute montant au solde."""
    self.solde += int(montant)        # le solde est en sou entier
```

Q2

```
def debiter(self, montant) :
    """Enlève montant au solde."""
    self.solde -= int(montant)        # le solde est en sou entier
```

Q3

```
def est_positif(self, montant) :
    """Renvoie Vrai si le solde du compte est positif ou nul."""
    return self.solde >= 0
```

Partie 2

Q1

```
cpt_0123456A = Compte("0123456A", "MARTIN Dominique", "12 rue des sports")
```

Q2

```
cpt_0123456A.crediter(200)
```

Q3

```
def transferer(self, autre_compte, montant)
    """transfère le montant vers autre_compte.
    On suppose que le compte courant est suffisamment approvisionné. """
    self.debiter(montant)
    autre_compte.crediter(montant)
```

Partie 3

Q1

```
    def rechercher_debiteurs (liste_comptes)
    """renvoie une liste de dictionnaires dont la première clé est le nom de l'adhérent
    et la seconde clé le solde de son compte en négatif."""
    debiteur = []
    for compte in liste_comptes :
            if compte.est_positif() == False :
            debiteur.append({'Nom': compte.adherent, 'solde': compte.solde})
```

```
    return debiteur
```

Q2

```
def urgent_debiteur(liste_debiteurs)
    """renvoie le nom de l'adhérent le plus endetté.
    On suppose qu'aucun adhérent n'a la même dette."""
    nom = ''
    if len(liste_debiteurs) :              # attention si dictionnaire < 2 débiteurs
            dette_max = liste_debiteurs[0]['solde']
            nom = liste_debiteurs[0]['Nom']
    for debiteur in range(1, len(liste_debiteurs)) :
            if dette_max > liste_debiteurs[debiteur]['solde'] :
                    dette_max = liste_debiteurs[debiteur]['solde']
                    nom = liste_debiteurs[debiteur]['Nom']
    return nom
```