Proposition de correction

Exercice 1

Q1.a

Pouvoir identifier de manière unique un enregistrement

Q1.b

Mise en relation avec les coordonnées du client et le produit commandé

Q1.c

Meubles (id, intitule, prix, stock, description)

Q2

Id	stock	description
62	2	Armoire blanche 3 portes
63	3	Armoire noire 3 portes

Q3

```
SELECT nom, prenom
FROM Clients
WHERE ville = ‘Paris’
ORDER BY nom, prenom
```

Q4

```
UPDATE Meubles
SET stock = 50
WHERE id = 98
```

Q5

```
INSERT INTO Meubles
VALUES(65, 'matta', 95.99, 25, 'Tapis vert à pois rouges')
```

Q6

```
SELECT Clients.nom, Clients.prenom
FROM Clients, Commandes
WHERE Commandes.date = ‘2021-04-30’
AND Clients.id = Commandes.idClient
```

ORDER BY Clients.nom, Clients.prenom

Exercice 2

Q1

C'est un réseau maillé

Q2

SiteB → R2 → R3 → R4 → R5 → SiteC

Q3

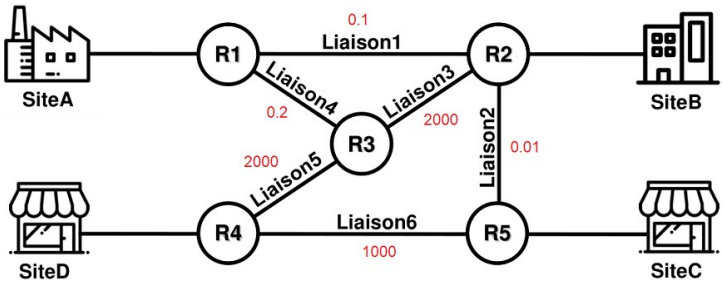
Routeur R1 (RIP)		
Destination	**Suivant**	**Nombre de sauts**
SiteA	Local	0
SiteB	R2	1
SiteC	R2	2
SiteD	R3	2

Q4

2 sites passent par R2 : risque de saturation

Q5.a

Liaison	Coût	D (Mbps)
Liaison1	100 000	0,1
Liaison2	1 000 000	0,01
Liaison3	5	2000
Liaison4	50 000	0,2
Liaison5	5	2000
Liaison6	10	1000



Q5.b

SiteA → R1 → R2 → R5 → SiteC — Coût = 100 000 + 1 000 000 = 1 100 000

SiteA → R1 → R3 → R2 → R5 → SiteC — Coût = 50 000 + 5 + 1 000 000 = 1 050 005

SiteA → R1 → R3 → R4 → R5 → SiteC — Coût = 50 000 + 5 + 10 = 50 015

SiteA → R1 → R2 → R3 → R4 → R5 → SiteC — Coût = 100 000 + 5 + 5 + 10 = 100 020

Q5.c

Routeur R1 (OSPF)		
Destination	**Suivant**	**Coût total du chemin**
SiteA	Local	0
SiteB	R1	50 005
SiteC	R1	50 015
SiteD	R1	50 005

Exercice 3

Partie 1

Q1

attribut

Q2

str

Q3

```
ge = Region('Grand Est ')
```

Q4

```
def renvoie_premiere_couleur_disponible(self):
    '''
    Renvoie la première couleur du tableau des couleurs
    disponibles supposé non vide.
    : return (str)
    '''
    return self.tab_couleurs_disponibles[0]
```

Q5

```
def renvoie_nb_voisines(self) :
    '''
    Renvoie le nombre de régions voisines.
    : return (int)
    '''
    return len(self.tab_voisines)
```

Q6

```
def est_coloriee(self):
    '''
    Renvoie True si une couleur a été attribuée à cette
    région et False sinon.
    : return (bool)
    '''
```

```
    return self.couleur_attribuee is not None
```

Q7

```
def retire_couleur(self, couleur):
    '''
    Retire couleur du tableau de couleurs disponibles de
    la région si elle est dans ce tableau. Ne fait rien sinon.
    : param couleur (str)
    : ne renvoie rien
    : effet de bord sur le tableau des couleurs disponibles
    '''
    if couleur in self.tab_couleurs_disponibles :
            self.tab_couleurs_disponibles.remove(couleur)
```

Q8

```
def est_voisine(self, region):
    '''
    Renvoie True si la region passée en paramètre est une
    voisine et False sinon.
    : param region (Region)
    : return (bool)
    '''
    return region in self.tab_voisines
```

Partie 2

Q9

```
def renvoie_tab_regions_non_coloriees(self):
    '''
    Renvoie un tableau dont les éléments sont les régions
    du pays sans couleur attribuée.
    : return (list) tableau d'instances de la classe Region
    '''
    return [ region for region in self.tab_regions if region.est_coloriee() ]
```

Q10.a

toutes les régions sont coloriées

Q10.b

- région non coloriée
- et qui possède le plus de régions limitrophes

Q11

```
def colorie(self):
    region = self.renvoie_max()
    while region is not None:
            couleur = region.renvoie_premiere_couleur_disponible()
            for region_voisine in region.tab_voisines:
                    if couleur == region_voisine.couleur_attribuee :
                            region.retire_couleur(couleur)
                    else :
                            region.couleur_attribuee = couleur
            region = self.renvoie_max()
```