Proposition de correction

Exercice 1

Partie A

Q1

Protocole

Q2.a

Routeur

Q2.b

Commutateur

Q3

Matériel	Adresse IP	Masque	Passerelle
Routeur Port 1	172.16.0.1	255.255.0.0	
Routeur Port 2	192.168.11.1	255.255.255.0	
Routeur Port 3	192.168.11.254	255.255.255.0	
Serveur fichiers	192.168.11.10	255.255.255.0	192.168.11.1
Serveur données	192.168.11.11	255.255.255.0	192.168.11.1
Poste 1	192.168.11.20	255.255.255.0	192.168.11.1
Poste 2	192.168.11.21	255.255.255.0	192.168.11.1
Poste 3	192.168.11.22	255.255.255.0	192.168.11.1

Partie B

Q1

- 10.0.0.0
- 172.16.0.0
- 192.168.0.0

NB : par convention ici saut direct = 0

Q2

Adresse IP destination	Interface Machine ou Port
192.168.1.55	192.168.0.1
172.18.10.10	172.15.0.1

Q3

Table de routage simplifiée du Routeur1		
Routeur destination	**Métrique**	**Route**
R2 : Routeur2	0	R1 – R2
R3 : Routeur3	0	R1 – R3
R4 : Routeur4	1	R1 – R2 – R4
R5 : Routeur5	1	R1 – R3 – R5
R6 : Routeur6	1	R1 – R3 – R6
R7 : Routeur7	2	R1 – R2 – R4 – R7

Exercice 2

Q1

["Toulouse","Auch"] : ok

["Luchon","Muret"] : n'est pas une liaison directe (via St Gaudens)

["Quillan","Limoux"] : ok

Q2.a

```
liaisonsJoueur2 = [
    ["Toulouse","Castres"],
    ["Toulouse","Castelnaudary"],
    ["Castres","Mazamet"],
    ["Castelnaudary","Carcasonne"]
    ]
```

Q2.b

```
DictJoueur2 = {
    "Toulouse":["Castres","Castelnaudary"],
    "Castres":["Toulouse","Mazamet"],
    "Castelnaudary":["Toulouse","Carcasonne"],
    "Mazamet":["Castres"],
    "Carcasonne":["Castelnaudary"]
    }
```

Q3.a

assert len(listeLiaisons)

Q3.b

dictionnaire obtenu : { "Toulouse": ["Muret", "Montauban"], "Gaillac": ["St Sulpice"], "Muret": ["Pamiers"] }

au lieu de : { "Toulouse": ["Muret", "Montauban"], "Montauban": ["Toulouse"], "Gaillac": ["St Sulpice"], "St Sulpice": ["Gaillac"], "Muret": ["Toulouse", "Pamiers"], "Pamiers": ["Muret"] }

le programme ne prend en compte que la première ville du couple [villeA, villeB].

Q3.c

```
def construireDict(listeLiaisons : list) -> dict:
    """
    listeLiaisons est un tableau de tableaux représentant la
    liste des liaisons d'un joueur comme décrit dans le problème
    """
    assert len(listeLiaisons)

    def construire(villeA : str, villeB : str) -> dict:
        if not villeA in Dict.keys() :
            Dict[villeA] = [villeB]
        else :
            destinationsA = Dict[villeA]
            if not villeB in destinationsA :
                destinationsA.append(villeB)

    Dict = {}
    for liaison in listeLiaisons :
        construire(liaison[0], liaison[1])
        construire(liaison[1], liaison[0])
    return Dict
```

Exercice 3

Q1

SQL (Structured Query Language)

Q2.a

Table Atomes	
Attribut	**Type**
Z	INTEGER
nom	VARCHAR(15)
Sym	CHAR(2)
L	INTEGER
C	INTEGER
masse_atom	FLOAT

Table Valence	
Attribut	**Type**
Col	INTEGER
couche	CHAR(1)

Q2.b

- Z : clé primaire car identifie l'atome de manière unique
- C : clé étrangère se référant à l'attribut Col de la relation Valence

Q2.c

Atomes(#Z, nom, Sym, L, C, masse_atom)

Valence(#Col, couche)

Q3.a

aluminium, argon, chlore, magnesium, sodium, phosphore, soufre, silicium

Q3.b

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18

Q4.a

```
SELECT nom, masse_atom
FROM Atomes
ORDER BY nom
```

Q4.b

```
SELECT Atomes.Sym
FROM Atomes, Valence
```

WHERE Valence.Col = Atomes.C
AND Valence.couche = ‘s’
ORDER BY Atomes.Sym

Q5

UPDATE Atomes
SET masse_atom = 39.948
WHERE Z = 18

Exercice 4

Q1.a

- assert self.__arome in ['fraise', 'abricot', 'vanille', 'aucun']
- assert 0 < self.__duree < 366

Q1.b

aromatise

Q1.c

```
def GetArome(self):
    return self.__arome
```

Q2

```
def SetArome(self, arome : str):
    self.__arome = arome
    if arome == 'aucun':
            self.__genre = 'nature'
    else:
            self.__genre = 'aromatise'
```

Q3.a

```
def empiler(p : list, Yaourt : object) -> list :
    p.append(Yaourt)
    return p
```

Q3.b

```
def depiler(p : list) -> object :
    return p.pop()
```

Q3.c

```
def estVide(p : object) -> bool :
    return len(p) == 0
```

Q3.d

>>24

>>False

Exercice 5

Q1.a

Le Comma-separated values est un format texte représentant ligne par ligne des enregistrements de données ; chaque donnée est séparée par une virgule.

Q1.b

argument : str

retour : str

Q2.a

import csv

Q2.b

assert type(prenom) is str

Q2.c

```
if type(prenom) is not str :
    return None
```

Q3

```
  if len(prenom) > 1: # attention au débordemnt d'index
    lettres = prenom[len(prenom)-2].lower() + prenom[len(prenom)-1].lower()
    if lettres in liste_M2 :
      return "M"
    elif lettres in liste_F2 :
      return "F"
    else :
      return "I"
  return None
```