2021 Métropole septembre sujet 2

Exercice 1

Partie A

1. [0.25 point] Protocole

2.a. [0.25 point] A : routeur

2.b. [0.25 point] B : switch

3. [0.75 point]

Poste 3	192.168.11.22	255.255.255.0	192.168.11.1

Partie B

1. [0.75 point]
- 10.0.0.0
- 172.16.0.0
- 192.168.0.0

2. [0.5 point]
- 192.168.1.55 : 192.168.0.1
- 172.18.10.10 : 172.15.0.1

3. [1.25 point]

Routeur destination	Métrique	Route
R2	0	R1 - R2
R3	0	R1 - R3
R4	1	R1 - R2 - R4
R5	1	R1 - R3 - R5
R6	1	R1 - R3 - R6
R7	2	R1 - R2 - R4 - R7

Exercice 2

1. [0.5 point] La liaison Luchon-Muret passe par St Gaudens.

2.a. [0.5 point]

```
[["Toulouse", "Castres"],
 ["Castres", "Mazamet"],
 ["Toulouse", "Castelnaudary"],
 ["Castelnaudary", "Carcassonne"],
 ["Tarbes", "St Gaudens"]]
```

2.b. [0.5 point]

```
{"Toulouse": ["Castres", "Castelnaudary"],
 "Castres": ["Toulouse", "Mazamet"],
 "Mazamet": ["Castres"],
 "Castelnaudary": ["Toulouse", "Carcassonne"],
 "Carcassonne": ["Castelnaudary"],
 "Tarbes": ["St Gaudens"],
 "St Gaudens": ["Tarbes"]}
```

3.a. [0.5 point]

```
assert listeLiaisons != [], "La liste des liaisons est vide"
```

3.b. [1 point]

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

Il oublie les trajets dont la ville de départ est `villeB`.

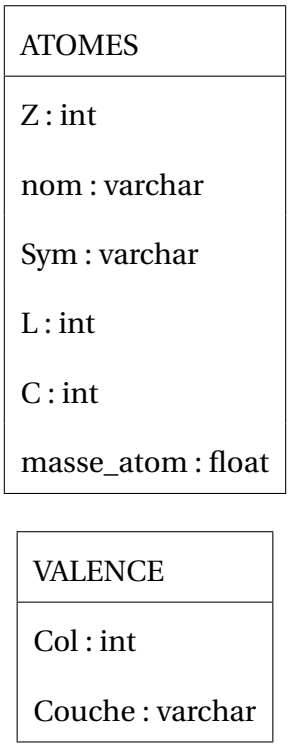
3.c. [1 point]
On ajoute entre la ligne 15 et la ligne 16 :

```
if not villeB in Dict.keys():
    Dict[villeB] = [villeA]
else:
    destinationB = Dict[villeB]
    if not villeA in destinationB:
        destinationB.append(villeA)
```

Exercice 3

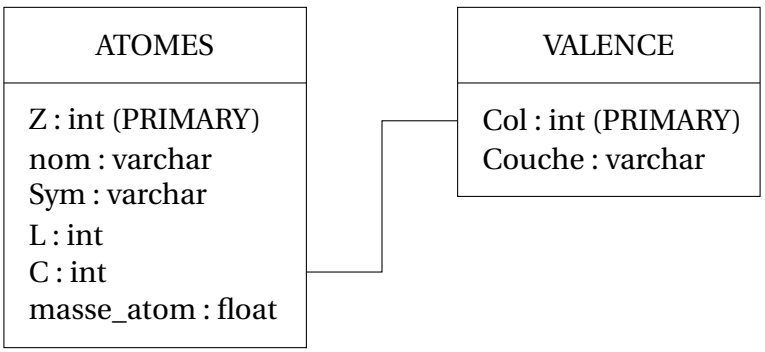
1. [0.25 point] SQL

2.a. [0.5 point]



2.b. [0.75 point]
`Z` peut être une clé primaire de la table `ATOMES` car elle est unique. `Nom`, `Sym` ou `masse_atom` peuvent également être des clés primaires mais les manipulations sur les entiers étant plus efficaces, on préférera `Z`.
`C` peut être une clé étrangère car elle correspond à l'attribut `Col` de la table `VALENCE`. `Col` étant la clé primaire de la table `VALENCE`.

2.c. [0.5 point]



3.a. [0.5 point]

aluminium
argon
chlore
magnesium
sodium
phosphore
soufre
silicium

3.b. [0.25 point] L'ordre n'est pas déterminé.

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

4.a. [0.25 point] `SELECT nom, masse_atom FROM ATOMES;`

4.b. [0.5 point]

```
SELECT Sym
FROM ATOMES
JOIN VALENCE
ON ATOMES.C = VALENCE.Col
WHERE Couche = "s";
```

5 [0.5 point]

```
UPDATE ATOMES
SET masse_atom = 39,948
WHERE Z = 18;
```

Exercice 4

1.a. [0.75 point]

```
assert arome == "aucun" or arome == "fraise" or arome == "abricot" or \
        arome == "vanille", "Arôme non valide"
assert duree > 0 and duree < 366, "Durée non valide"
```

1.b. [0.25 point] `'aromatise'`

1.c. [0.25 point]

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

2. [0.75 point]

```
def SetArome(self, arome):
    self.__arome = arome
    self.__setGenre(arome)
```

3.a. [0.5 point]

```
def empiler(p, Yaourt):
    p.append(Yaourt)
    return p
```

3.b. [0.5 point]

```
def depiler(p):
    return p.pop()
```

3.c. [0.5 point]

```
def estVide(p):
    return p == []
```

3.d. [0.5 point]

```
24
False
```

Exercice 5

1.a. [0.75 point] CSV signifie Coma Separated Values. C'est un fichier texte avec des données séparée par des virgules (ou points virgule ou tabulations) sur plusieurs lignes.

1.b. [0.5 point] Les deux sont de type `String`.

2.a. [0.5 point]
```
import csv
```

2.b. [0.5 point]

```
assert type(prenom) == str, "Type d'argument non valide"
```

2.c. [0.75 point]

```
if type(prenom) != str:
    return "I"
else:
    ...le reste du programme...
```

3. [1 point]

```
if prenom[len(prenom)-2].lower() + prenom[len(prenom)-1].lower() in liste_M2:
    return "M"
elif prenom[len(prenom)-2].lower() + prenom[len(prenom)-1].lower() in liste_F2:
    return "F"
else:
    return "I"
```