Éléments de correction sujet 10

Exercice 1

Partie A : Réseau

1

Protocole

2

a) élément A : Routeur
b) élément B : Switch

3

Matériel	Adresse IP	Masque	Passerelle
...	...	...	...
Poste 3	192.168.11.22	255.255.255.0	192.168.11.1

Partie B : Routage réseaux

1

Les adresses IP des réseaux directement connectés au routeur R1 (métrique égale à 0) sont : 10.0.0.0, 172.16.0.0 et 192.168.0.0

2

Adresse IP destination	Interface Machine ou Port
192.168.1.55	192.168.0.1
172.18.10.10	172.15.0.1

3

La question n'est pas très claire. Est-ce que l'on devrait considérer toutes les routes possibles pour atteindre un routeur donné ou seulement la route la plus courte ? La première solution étant très très longue, j'ai choisi de traiter la deuxième.

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 (ou R1-R3-R6-R7)

Exercice 2

1

La liste proposée n'est pas valide car la liaison ["Luchon", "Muret"] n'est pas directe.

2a

```
liaisonsJoueur2 = [["Toulouse", "Castres"],
                   ["Toulouse", "Castelnaudary"],
                   ["Castres", "Mazamet"],
                   ["Castelnaudary", "Carcassonne"],
                   ["Tarbes",  "St Gaudens"]]
```

2b

```
DictJoueur2 = {
                "Toulouse" : [ "Castres",  "Castelnaudary"],
                "Castres" : ["Toulouse",  "Mazamet"],
                "Castelnaudary" : ["Toulouse",  "Carcassonne"],
                "Mazamet" : ["Castres"],
                "Carcassonne" : ["Castelnaudary"],
                "Tarbes" : ["St Gaudens"],
                "St Gaudens" : ["Tarbes"]
    }
```

3a

```
assert len(listeLiaisons)!= 0, "la liste est vide"
```

3b

Résultat de l'exécution de la fonction construireDict :

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

La fonction gère la liaison A-B mais pas la liaison B-A. Par exemple, pour la clé "`Toulouse` " on retrouve bien "`Muret`" dans le tableau alors que pour la clé "`Muret`", on ne retrouve pas "`Toulouse` " dans le tableau.

3c

```
def construireDict(listeLiaisons):
    assert len(listeLiaisons)!= 0, "la liste est vide"
    Dict={}
    for liaison in listeLiaisons :
        villeA = liaison[0]
        villeB = liaison[1]
        if not villeA in Dict.keys() :
            Dict[villeA]=[villeB]
        else :
            destinationsA = Dict[villeA]
            if not villeB in destinationsA :
                destinationsA.append(villeB)
        if not villeB in Dict.keys() :
            Dict[villeB]=[villeA]
        else :
            destinationsB = Dict[villeB]
            if not villeA in destinationsB :
                destinationsB.append(villeA)
    return Dict
```

Exercice 3

1

Pour effectuer des requêtes sur une base de données relationnelle, on utilise le langage SQL

2a

ATOME (Z : INT, nom : TEXT, Sym : TEXT, L : INT, C : INT, masse_atom : FLOAT)
VALENCE (Col : INT, Couche : TEXT)

2b

l'attribut Z peut jouer le rôle de clé primaire car il existe un Z unique pour chaque élément chimique.
l'attribut C va jouer le rôle de clé étrangère car cet attribut va permettre d'établir une "liaison" avec l'attribut Col de la table VALENCE

2c

ATOME (Z : INT, nom : TEXT, Sym : TEXT, L : INT, #C : INT, masse_atom : FLOAT)
VALENCE (Col : INT, Couche : TEXT)

3a

On obtient la liste de nom d'atomes suivante :
aluminium, argon, chlore, magnesium, sodium, phosphore, soufre, silicium

3b

On obtient la liste des colonnes :
1, 2 ,3 ,4 ,5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18

4a

```
SELECT nom, masse_atom
FROM ATOMES
```

4b

```
SELECT Sym
FROM ATOMES
INNER JOIN VALENCE ON ATOMES.C = VALENCE.Col
WHERE Couche = 's'
```

5

```
UPDATE ATOMES
SET mass_atom = 39.948
WHERE nom = 'argon'
```

Exercice 4

1a

voici les 2 assertions dans la méthode `__init__` :

```
class Yaourt:
    def __init__(self,arome,duree):
        assert arome in ['fraise', 'abricot', 'vanille', 'aucun'], "Cet
arome est inconnu"
        assert duree > 0 and duree < 366, "la durée doit être comprise
entre 1 et  365"
        self.__arome = arome
        self.__duree = duree
        if arome == 'aucun':
            self.__genre = 'nature'
        else:
            self.__genre = 'aromatise'
```

1b

Le genre associé à Mon_Yaourt sera `aromatise`

1c

Voici la **méthode** GetArome :

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

2

```
def SetArome(self, arome):
       assert arome in ['fraise', 'abricot', 'vanille', 'aucun'], "Cet
arome est inconnu"
       self.__arome = arome
       self.__SetGenre(arome)
```

3a

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

3b

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

3c

```
def estVide(p):
    return len(p)==0
```

3d

```
24
False
```

Exercice 5

1a

Un fichier CSV est un fichier au format “texte” permettant de “stocker” des données tabulées. Les données sont séparées par des virgules, d’où l’acronyme CSV : Comma Separated Values

1b

- prenom est de type string
- la réponse renvoyée par la fonction est aussi de type string

2a

```
import csv
```

2b

```
assert isinstance(prenom, str)
```

2c

```
def genre(prenom):
    liste_M = ['f', 'd', 'c', 'b', 'o', 'n', 'm', 'l', 'k','j', 'é',
'h', 'w', 'v', 'u', 't', 's', 'r','q', 'p', 'i', 'þ', 'z', 'x', 'ç',
'ö', 'ã','â', 'ï', 'g']
    liste_F = ['e', 'a', 'ä', 'ü', 'y', 'ë']
    if not isinstance(prenom, str):
        return "erreur, le prénom doit être une chaîne de caractères"
    if prenom[len(prenom)-1].lower() in liste_M :
        return "M"
    elif prenom[len(prenom)-1].lower() in liste_F :
        return "F"
    else :
        return "I"
```

3

modification de la fonction genre (de la ligne 7 à la ligne 13) :

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