Métropole - septembre 2021 (corrigé)

Exercice 1 (Réseaux et routage)
Partie A : réseau.

1. Il s'agit du protocole (réponse b).
2. (a) L'élément A est un routeur.
   (b) L'élément B est un commutateur (switch).
3. On a le tableau suivant :

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. On a le tableau suivant :

Adresse IP destination	Interface Machine ou Port
`192.168.1.55`	`192.168.0.1`
`172.18.10.10`	`175.15.0.1`

3. On a le tableau suivant :

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 (ou R1 – R3 – R6 – R7)

Exercice 2 (Dictionnaires, tableaux et programmation)

1. La liste proposée n'est pas valide car la liaison `["Luchon", "Muret"]` n'est pas directe.
2. (a) On a le tableau suivant :

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

   (b) On a le dictionnaire suivant :

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

3. (a) L'instruction suivante convient :

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

   (b) La fonction renvoie le résultat suivant :

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

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

   (c) La fonction suivante convient :

```
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 (SQL)

1. Pour effectuer des requêtes sur une base de données relationnelle, on utilise le langage SQL.
2. (a) Pour la relation `ATOMES` on a les attributs suivants :
   - `Z` : le numéro atomique qui est une valeur entière comprise entre 1 et 54 ;
   - `nom` : une chaîne de caractères décrivant l'élément chimique, son nom chimique ;
   - `Sym` : une chaîne de caractères, son symbole chimique ;
   - `L` : un entier compris entre 3 et 5 ;
   - `C` : un entier compris entre 1 et 18 ;
   - `masse_atom` : un nombre décimal (flottant) donnant sa masse atomique.

   Pour la relation `VALENCE` on a les attributs suivants :
   - `Col` : un entier compris entre 1 et 18 ;
   - `Couche` : une des chaînes de caractères `'s'`, `'p'` ou `'d '` correspondant à la couche de Valence.

   (b)
   - L'attribut `Z` peut jouer le rôle de clé primaire pour la relation `ATOMES` car il existe un `Z` unique pour chaque élément chimique. Les attributs `nom` et `Sym` étant également uniques peuvent également jouer le rôle de clé primaire pour la relation `ATOMES` .
   - L'attribut `Col` peut jouer le rôle de clé primaire pour la relation `VALENCE` car il existe un unique numéro de colonne dans cette relation.
   - L'attribut `C` va jouer le rôle de clé étrangère car cet attribut va permettre d'établir une « liaison » avec l'attribut `Col`, clé primaire de la table `VALENCE`.

   (c) En choisissant `Z` comme clé primaire (choix le plus naturel) on a le schéma relationnel suivant :

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

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

   (b) On obtient la liste des colonnes : 1, 2 ,3 ,4 ,5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18.
4. (a) La requête suivante convient :

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

   (b) La requête suivante convient :

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

5. La requête suivante convient :

```
def taille(self):
    if self.gauche == None and self.droit == None :
        return 1
    if self.gauche == None :
        return 1 + self.droit.taille()
    elif self.droit == None :
        return 1 + self.gauche.taille()
    else :
        return 1 + self.gauche.taille() + self.droit.taille()
```

Exercice 4 (POO)

1. (a) Les deux assertions sont les suivantes :

```
assert arome in ['fraise' , 'abricot', 'vanille', 'aucun'], "Cet arôme est inconnu"
assert duree > 0 and duree < 366, "la durée doit être comprise entre 1 et 365"
```

   (b) Le genre associé à `Mon_Yaourt` sera `aromatise`.

   (c) La méthode suivante convient :

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

2. La fonction suivante convient :

```
# Renvoie un entier aléatoire N tel que a <= N <= b.
# Alias pour randrange(a,b+1).}
```

3. (a) La fonction suivante convient :

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

   (b) La fonction suivante convient :

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

   (c) La fonction suivante convient :

```
lst = [v for v in range(5)]
melange(lst, 4)
```

   (d) Le bloc d'instructions affiche :

```
24
False
```

Exercice 5 (Données en table et programmation)

1. (a) 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*.

   (b) `prenom` est de type `string` et la réponse renvoyée par la fonction est aussi de type `string`.

2. (a) On saisit l'instruction suivante : `import csv`

   (b) L'assertion suivante convient : `assert isinstance(prenom, str)`

   (c) La fonction suivante convient :

```
def pgsp(lst):
    n = len(lst)
    somme_max = lst[0]
    i_max = 0
    j_max = 0
    for i in range(n):
        for j in range(i,n):
            s = somme_sous_sequence(lst,i,j)
            if s > somme_max:
                somme_max = s
                i_max = i
                j_max = j
    return (somme_max, i_max, j_max)
```

3. On a les modifications suivantes pour les lignes 7 à 12 :

```
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'
```