

[← Index des sujets 2021](#)

21-NSIJ2ME3 : Corrigé

Année : **2021**

Centre : **Métropole session de septembre**

Jour : **2**

Enoncé : [PDF](#)

1. Exercice 1

réseau, protocoles de communication et de routages

Partie A :

1. Protocole
2. a) élément A : Routeur
b) élément B : Switch M

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

Partie B :

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	172.15.0.13

3.	Routeur destination	Métrique	Route
	R2	0	R1-R2

Routeur destination	Métrique	Route
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

2. Exercice 2

structure de données, langages et programmation

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

Script Python

```
liaisonsJoueur2 = [["Toulouse", "Castres"],
["Toulouse", "Castelnau'dary"],
["Castres", "Mazamet"],
["Castelnau'dary", "Carcassonne"],
["Tarbes", "St Gaudens"]]
```

b.

Script Python

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

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

b. Résultat de l'exécution de la fonction `construireDict` :

Script Python

```
{'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.

c. python

```
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
```

3. Exercice 3

base de données

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

2. a.

Requête SQL

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

b. 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

c.

Requête SQL

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

1. a. On obtient la liste de nom d'atomes suivante :

aluminium, argon, chlore, magnésium, 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

1. a.

Requête SQL

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

- b.

Requête SQL

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

- 5.

Requête SQL

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

4. Exercice 4

programmation orientée objet, langages et programmation

1. a voici les 2 assertions dans la méthode `init` :

Script Python

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

b. Le genre associé à Mon_Yaourt sera aromatisé

c. Voici la méthode GetArome :

Script Python

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

2.

Script Python

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

1. a.

Script Python

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

b.

Script Python

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

c.

Script Python

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

d. 24

False

5. Exercice 5

traitement de données en table (CSV), langages 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 - la réponse renvoyée par la fonction est aussi de type string

```

1. a. import csv

b. assert isinstance(prenom, str)

c. python
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', 'b', 'z', 'x', 'ç', 'ö', 'ä', '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"

```

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

Script Python

```

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"

```