Correction

NSI - 2021 Métropole Jour 2 (--)

Exercice 1 - POO et arbres pour les données d'une agence immobilière

1. La console affiche :

```
class Bim:
    def __init__(self, nature, surface, prix_moy):
        self.nt = nature
        self.sf = surface
        self.pm = prix_moy

    def estim_prix(self):
        return self.sf * self.pm
```

2. L'instruction `b1.estim_prix()` renvoie 140 000.0 (70.0 x 2000.0). Ce résultat est de type « float ».

3.

```
def estim_prix(self):
    prix_brut = self.sf * self.pm
    if self.nt == 'maison':
        return prix_brut * 1.1
    elif self.nt == 'bureau':
        return prix_brut * 0.8
    else :
        return prix_brut
```

4.

```
def nb_maison(lst):
    total = 0
    for element in lst :
        if element.nt == 'maison':
            total = total + 1
    return total
```

5. a. Il s'agit du parcours infixe : b2, b4, b1, b5, b3 et b6.

5. b.

```
def contient(surface, abr):
    if abr.est_vide():
        return False
    elif abr.get_v().sf >= surface:
        return True
    else :
        return contient(surface, abr.get_d())
```

Correction

NSI - 2021 Métropole Jour 2 (--)

Exercice 2 - Base de données d'un restaurant

1. La requête R2 renvoie bien les valeurs de tous les attributs des plats de la catégorie 'entrée'.

2. a.

```
SELECT Client.nom, Client.avis FROM Client
JOIN Reservation ON Reservation.idClient = Client.idClient
WHERE Reservation.jour = '2021-06-05' AND Reservation.heure = '19:30:00';
```

On pourrait ajouter : `ORDER BY Client.nom` à la fin de la requête mais ce n'est pas demandé.

2. b.

```
SELECT DISTINCT Plat.nom FROM Plat
JOIN Commande ON Plat.idPlat = Commande.idPlat
JOIN Reservation ON Reservation.idReservation = Commande.idReservation
WHERE (Plat.categorie = 'plat principal' OR Plat.categorie = 'dessert')
AND Reservation.jour = '2021-04-12';
```

Remarque : `DISTINCT` permet de ne faire apparaître qu'une fois un même plat.
Il y a aussi la possibilité d'utiliser des alias (avec `AS`) pour alléger la requête :

```
SELECT DISTINCT P.nom FROM Plat AS P
JOIN Commande AS C ON P.idPlat = C.idPlat
JOIN Reservation AS R ON R.idReservation = C.idReservation
WHERE (P.categorie = 'plat principal' OR P.categorie = 'dessert')
AND R.jour = '2021-04-12';
```

3. Cette requête SQL permet d'ajouter un plat ayant pour identifiant `58`, pour nom `'Pêche Melba'`, pour catégorie `'dessert'`, pour description `'Pêche et glace vanille'` et pour prix 6,5€.

4. a.

```
DELETE FROM Commande WHERE idReservation = 2047;
```

4. b.

```
UPDATE Plat SET prix = prix * 1.05 WHERE prix < 20.0;
```

Correction

NSI - 2021 Métropole Jour 2 (--)

Exercice 3 - Les réseaux et les protocoles de routage

1. a. L'adresse du réseau local L1 est 192.168.1.0/24. L'adresse du réseau local L2 est 172.16.0.0/16.

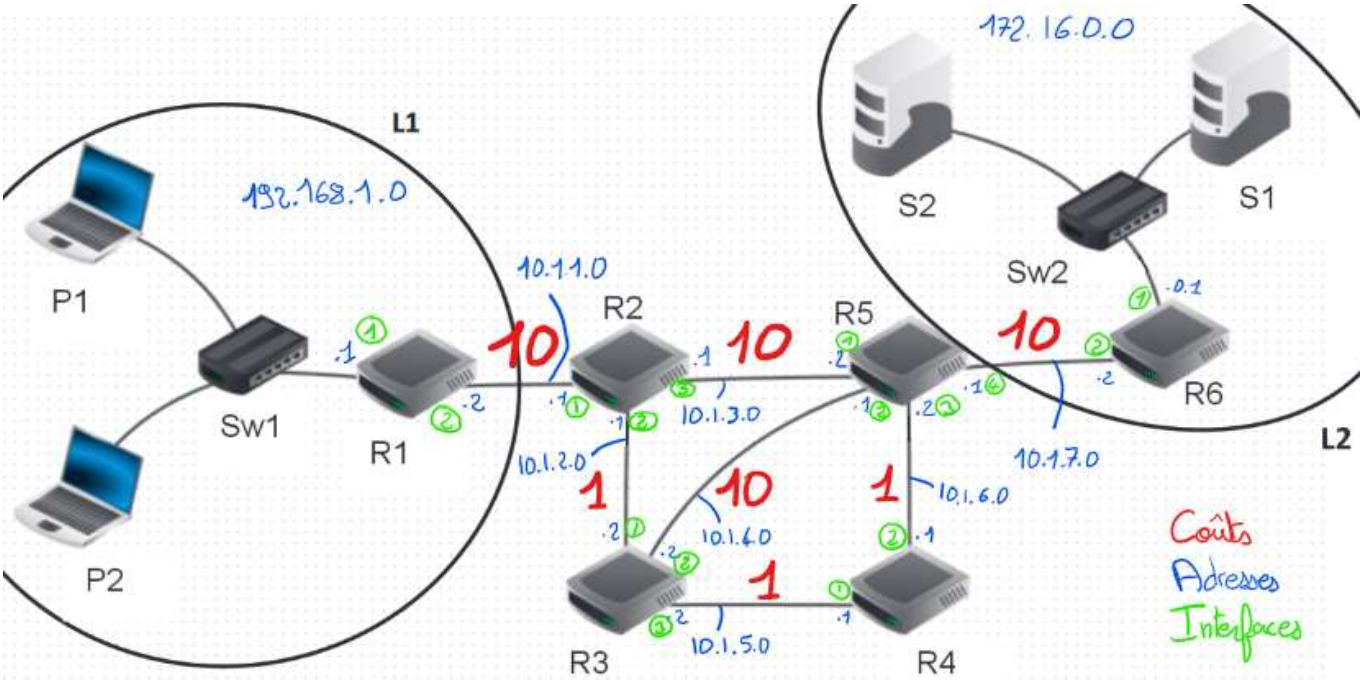
1.b. Sur le réseau L1, les adresses machines peuvent aller de 192.168.1.1 à 192.168.1.254.
Sur le réseau L2, les adresses machines peuvent aller de 172.16.0.1 à 172.16.255.254.

1.c. Sur le réseau L1, il est possible de connecter $2^8 - 2 = 254$ machines puisque 8 bits sont réservés pour l'adresse machine. Sur le réseau L2, avec 16 bits, il est possible de connecter $2^{16} - 2 = 65\,534$ machines.

2.a. Il est utile d'avoir plusieurs chemins en cas de panne ou de surcharge d'un routeur.

2.b. Le chemin le plus court en nombre de saut (protocole RIP) est : $R1 \rightarrow R2 \rightarrow R5 \rightarrow R6$ avec 3 sauts.

2.c. Le coût d'une liaison Ether est de $\frac{10^8}{10^7} = 10$ tandis que le coût d'une liaison FastEther est de $\frac{10^8}{10^8} = 1$.
Le chemin qui a le plus petit coût (protocole OSPF) est $R1 \rightarrow R2 \rightarrow R3 \rightarrow R4 \rightarrow R5 \rightarrow R6$ pour un coût total de $10 + 1 + 1 + 1 + 10 = 23$.



3. Tables de routage :

R5 :

IP réseau de destination	Passerelle suivante	Interface
10.1.3.0/24	10.1.3.2	Interface 1
10.1.4.0/24	10.1.4.2	Interface 2
10.1.6.0/24	10.1.6.2	Interface 3
10.1.7.0/24	10.1.7.1	Interface 4
172.16.0.0/16	10.1.7.2	interface 4
192.168.1.0/24	10.1.3.0	interface 1

R6 :

IP réseau de destination	Passerelle suivante	Interface
172.16.0.0/16	172.16.0.1	Interface 1
10.1.7.0/24	10.1.7.1	interface 2
192.168.1.0/24	10.1.7.1	interface 2

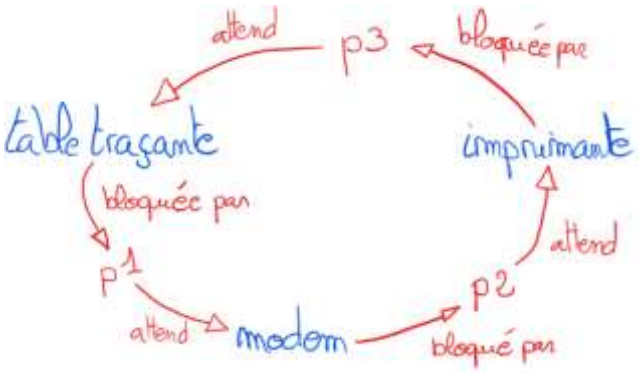
Correction

NSI - 2021 Métropole Jour 2 (--)

Exercice 4 - systèmes d'exploitation : gestion des processus

PARTIE A :

1. Le programme 1 bloque la table traçante et attend le modem bloqué par le programme 2 qui attend l'imprimante bloquée par le programme 3 qui attend la table traçante.
Tous les programmes attendent une ressource bloquée par un autre programme ce qui crée un interblocage.



2. Pour éviter l'interblocage, il faudrait inverser l'ordre des demandes pour un des trois programmes. Par exemple, le programme 3 devrait demander l'accès à la table traçante avant l'imprimante :

```
Programme 3
demander (imprimante)
demander (table traçante)
exécution
libérer (table traçante)
libérer (imprimante)
```

3. L'état du processus p1 tant que la table traçante ne sera pas disponible sera : bloqué. (Pour être prêt, il faudrait qu'il ait accès à toutes les ressources dont il a besoin.)

PARTIE B :

1. C'est la commande `ps -ef` qui permet cet affichage.
(`ls` permet de liste le contenu d'un répertoire, `cd` permet de changer de répertoire et `chmod` permet de changer les droits d'accès.)

2. L'identifiant du processus parent (PPID) à l'origine de tous les processus concernant le navigateur web est 831.

3. L'identifiant du processus (PID) dont le temps d'exécution est le plus long est 6 211 avec un temps d'exécution d'1min16.

Correction

NSI - 2021 Métropole Jour 2 (--)

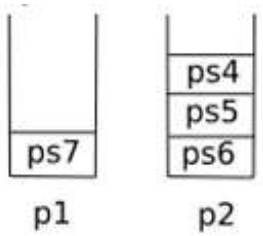
Exercice 5 - Structures de données linéaires : Piles et Files - POO

1. La structure de données la plus appropriée pour mettre en œuvre le mode FIFO (First In First Out) est la file.

2.

```
def ajouter(lst, proc):
    lst.append(proc)
```

3.



4. a.

Solution 1

```
def est_vide(f):
    # f est un tuple (p1, p2)
    p1, p2 = f
    return pile_vide(p1) and pile_vide(p2)
```

Solution 2

```
def est_vide(f):
    # f est un tuple (p1, p2)
    return pile_vide(f[0]) and pile_vide(f[1])
```

4. b.

```
def enfiler(f, elt):
    empiler(f[0], elt)
```

4. c.

```
def defiler(f):
    p1, p2 = f
    if pile_vide(p2):
        while not pile_vide(p1):
            empiler(p2, depiler(p1))
    return depiler(p2)
```