

## Exercice 1 : Base de données – Musiciens – Groupes – Concerts

1. Les attributs de la table groupes sont : idgrp, nom, style, nb\_pers.

2. Le même nom peut apparaître plusieurs fois (dans le cas d'homonymes), ce qui est interdit pour une clé primaire.

3. Cette requête renvoie : 'Weather Report' et 'Return to Forever'.

4.

```
UPDATE concerts SET heure_fin = '22h30' WHERE idconc = 36;
```

5.

```
SELECT nom FROM groupes  
JOIN concerts ON concerts.idgrp = groupes.idgrp  
WHERE scene = 1;
```

ou avec des alias :

```
SELECT g.nom FROM groupes AS g  
JOIN concerts AS c ON c.idgrp = g.idgrp  
WHERE c.scene = 1;
```

6.

```
INSERT INTO groupes  
VALUES (15, 'Smooth Jazz Fourplay', 'Free Jazz', 4);
```

7.

```
def recherche_nom(tab):  
    rep = []  
    for d in tab:  
        if d['nb_concerts'] >= 4 and d['nom'] not in t:  
            rep.append(d['nom'])  
    return rep
```

ou

# un set est une séquence ne contenant que des valeurs différentes

```
def recherche_nom(tab):  
    return set([d['nom'] for d in tab if d['nb_concerts'] >= 4])
```

## Exercice 2 : Architecture réseau – Routage

1. Nous avons un hôte d'adresse IP 172.16.2.3/24 qui appartient au réseau d'adresse réseau 172.16.2.0. C'est l'ordinateur d'Alice qui appartient à ce réseau (voir schéma).

2. Calcul du coût : coût = 10000/1000 = 10

3.

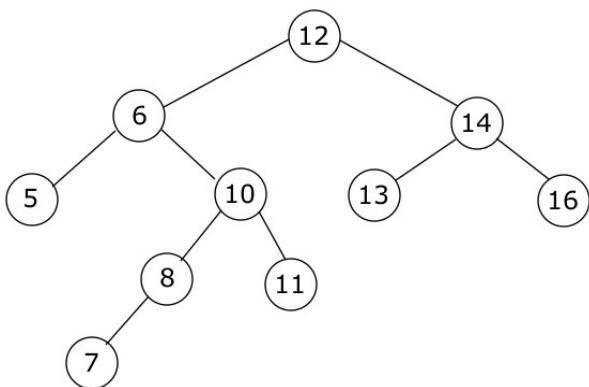
Routeur R6		
Destination	Pass.	Coût
LAN1	R5	21
LAN2	-	-
WAN1	R5	11
WAN2	R5	20
WAN3	R5	11
WAN4	R5	12
WAN5	R5	10
WAN6	-	-
WAN7	-	-
WAN8	R5	10

4. Bob → R1 → R2 → R5 → R6 → Alice

5. Le nouveau chemin est R1 → R2 → R4 → R6. On évite le routeur R5, c'est donc le routeur 5 qui est en panne.

### Exercice 3 : Architecture réseau - Routage

1. Il s'agit d'une structure FIFO (First-In-First-Out), c'est-à-dire une file.
2. a. Le nombre de nœuds d'un arbre correspond à sa taille.
2. b. La tâche la plus ancienne à effectuer est la première arrivée et constitue la racine de l'arbre.
2. c. La dernière tâche arrivée correspond à une feuille de l'arbre.
3. a. Les attributs de la classe Noeud sont : tache, indice, gauche et droite
3. b. La méthode `insere` est dite récursive, car elle s'appelle elle-même. Dans cette méthode récursive, on trouve bien le traitement du cas de base `if self.est_vide()`, ce qui permet d'affirmer que cette méthode se termine.
3. c. Il s'agit du signe `>` (strictement supérieur).  
(En effet si l'indice du nœud courant est strictement supérieur à l'indice de priorité de la tâche à ajouter, il faut insérer la nouvelle tâche dans le sous-arbre gauche.)
3. d. Arbre complété avec les tâches d'indice 11, 5, 16 et 7 :



4.
 

```

def est_present(self, indice_recherche) :
    """renvoie True si l'indice de priorité indice_recherche
    (int) passé en paramètre est déjà l'indice d'un nœud
    de l'arbre, False sinon"""
    if self.est_vide():
        return False
    if self.racine.indice == indice_recherche:
        return True
    if self.racine.indice > indice_recherche :
        return self.racine.gauche.est_present(indice_recherche)
    else :
        return self.racine.droite.est_present(indice_recherche)
      
```

5. a. Le parcours infixe de l'arbre de la figure 1 est : 6 - 8 - 10 - 12 - 13 - 14

5. b. Le parcours infixe permet d'obtenir les valeurs des nœuds d'un arbre binaire de recherche dans un ordre croissant. Le parcours infixe va donc permettre d'obtenir les tâches à accomplir dans l'ordre des priorités.

6.

```
def tache_prioritaire(self):
    """renvoie la tache du noeud situé le plus
    à gauche de l'ABR supposé non vide"""
    if self.racine.gauche.est_vide(): #pas de nœud plus à gauche
        return self.racine.tache
    else:
        return self.racine.gauche.tache_prioritaire()
```

7.

Rappel : Plus l'indice est faible, plus la tâche doit être traitée prioritairement.

