Amérique du nord - mars 2023 - sujet 1 (corrigé)

Exercice 1 (Données en table et bases de données)
Partie A : traitement de données en table

1. (a) Avec cette instruction, on accède à la valeur de la clé `'insee'` dans le premier dictionnaire (indice `0`) de la liste `tab_lieux`. On obtient donc la valeur `'1024'`.
   (b) Il faut saisir l'instruction `tab_lieux[1]['ad_lieu']`
2. Il s'agit des propositions 2 et 3 (la proposition 1 déclenche une erreur car la variable `dico` n'est pas définie).
3. Cet appel renvoie le nombre d'aire de stationnement située en sortie d'autoroute ayant au moins 100 places.
4. La fonction suivante convient :

```
def insee_max_places(table):
    maxi = table[0]['nb_places']
    code_insee = table[0]['insee']
    for dico in table:
        if dico['nb_places'] > maxi:
            maxi = dico['nb_places']
            code_insee = dico['insee']
    return code_insee
```

5. La fonction suivante convient :

```
def moyenne_par_type(table, nom_type):
    nb_lieux = 0
    nb_places = 0
    for dico in table:
        if dico['type'] == nom_type:
            nb_lieux += 1
            nb_places += dico['nb_places']
    if nb_lieux != 0:
        moyenne = nb_places/nb_lieux
    else:
        moyenne = 0
    return moyenne
```

Partie B : base de données

6. Dans la relation `COMMUNE`, l'attribut `code_insee` est une clé primaire qui garantie la contrainte d'unicité dans la relation. Ainsi, deux communes ne peuvent pas posséder le même code INSEE.
7. La requête suivante convient :

```
SELECT id_lieu, code_insee FROM LIEU WHERE type = 'Sortie autoroute'
```

8. La requête suivante convient :

```
SELECT COUNT(id_lieu) FROM LIEU
JOIN COMMUNE
ON LIEU.code_insee = COMMUNE.code_insee
WHERE COMMUNE.departement = 'JURA'
```

9. (a) Cette requête supprime dans la table `COMMUNE` la donnée dont l'attribut `code_insee` est `'40714'`. Dans la table `LIEU`, il existe des données dont l'attribut `code_insee` est `'40714'`. Or, cet attribut est une clé étrangère de la table `LIEU` qui fait référence à l'attribut `code_insee` de la table `COMMUNE` dont elle est une clé primaire. La contrainte d'inclusion n'étant pas vérifiée, on obtient une erreur.
   (b) La requête suivante convient :

```
UPDATE LIEU SET code_insee = '40146' WHERE code_insee = '40714'
```

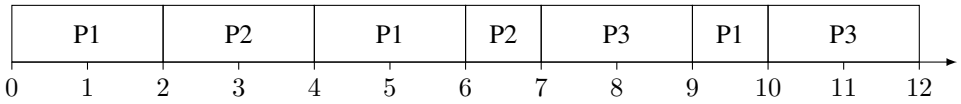
Exercice 2 (Processus et logique booléenne)

Partie A : processus

1. L'application associé au processus dont le `PID` est `5468` est le logiciel `gimp`.
2. Le processus qui a sollicité le processeur le plus longtemps a pour identifiant `1770` (51 secondes).
3. On recherche le `PPID` qui apparaît le plus de fois. Il s'agit du `PPID` valant `1611`.
4. La commande `ps` a un `PPID` et `5622` et le programme dont le `PID` est `5622` est `bash`. Ainsi, `bash` est le nom du programme dont est issue la commande `ps`.
5. On a les identifiants suivants : `1 - 1611 - 5571 - 5622 - 5629`

Partie B : ordonnancement

6. On a les définitions suivantes :
   - ⋆ prêt : le processus est en attente du processeur ;
   - ⋆ bloqué : le processus est en attente de ressources ;
   - ⋆ élu : le processeur est exécuté par le processeur.
7. On a le chronogramme d'exécution suivant :



Partie C : logique booléenne

8. On a la table de vérité suivante :

a	b	a ET b	NON(a ET b)
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

9. On a la table de vérité suivante :

a	b	NON a	NON b	(NON a) OU (NON b)
0	0	1	1	1
0	1	1	0	1
1	0	0	1	1
1	1	0	0	0

Les tables de vérité des deux expressions `NON(a ET b)` et `(NON a) OU (NON b)` étant égales, on en déduit que ces deux expressions sont égales (il s'agit d'une des deux lois de de Morgan).

Exercice 3 (POO et ABR)

Partie A : programmation orientée objet

1. On a le code suivant :

```
def __init__(self, nom_clan, pts_vie, pts_force):
    self.clan = nom_clan
    self.vie = pts_vie
    self.force = pts_force
```

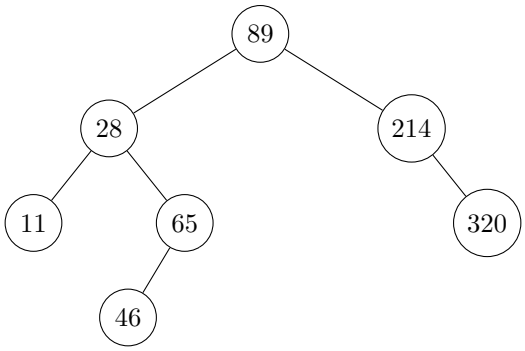
2. ⋆ Les objets de cette classe ont trois attributs : `clan`, `vie` et `force`.
   ⋆ Cette classe a deux méthodes en plus de la méthode d'initialisation `__init__` : `attaque` et `defense`.
3. Il faut saisir l'instruction suivante : `luther = Personnage('Umbrella', 100, 15)`
4. Le code suivant convient :

```
def attaque(self, autre_perso):
    autre_perso.vie -= self.force
    if autre_perso.vie > 0:
        return 1
    return 0
```

5. Il suffit qu'il utilise une structure de pile : la dernière action réalisée est au sommet de la pile ; si on l'annule, il suffit de la dépiler et on obtient l'état antérieur.

Partie B : arbres binaires de recherche

6. (a) Cet arbre a pour hauteur 3 et pour taille 6.
   (b) On obtient l'arbre suivant :



7. (a) Les nœuds sont traités dans l'ordre *gauche - racine - droite*, donc il s'agit du parcours en profondeur infixe.
   (b) Avec ce parcours, on obtient les valeurs suivantes : $11-28-65-89-214-320$, c'est-à-dire les valeurs des clés de l'arbre par ordre croissant.
8. La fonction suivante convient :

```
def recherche_dichotomique(arbre, ref_materiel):
    if est_vide(arbre):
        return -1
    elif ref_materiel < valeur_racine(arbre)[0]:
        return recherche_dichotomique(gauche(arbre), ref_materiel)
    elif ref_materiel > valeur_racine(arbre)[0]:
        return recherche_dichotomique(droit(arbre), ref_materiel)
    else:
        return valeur_racine(arbre)[1]
```