Asie - mai 2022 - sujet 2 (corrigé)

Exercice 1 (Système d'exploitation)

1. (a) L'utilisateur est `gestion`.
   (b) Il faut saisir l'instruction : `ls ./Contrats`
2. (a) Il faut saisir l'instruction : `mkdir ./Contrats/TURING_Alan`
   (b) Il faut saisir l'instruction : `chmod ug+rxw,o+r ./Contrats/TURING_Alan`
3. Les deux fonctions suivantes conviennent :

```
def formatage(tab):
    t = []
    for v in tab:
        t.append(v[0]+"_"+v[1])
    return t
```

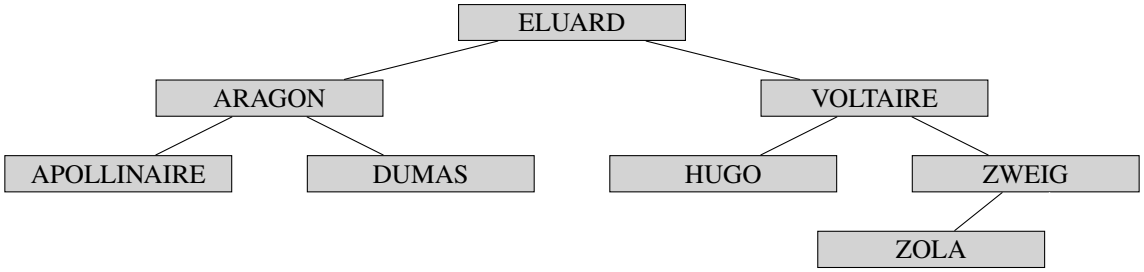
```
def formatage(tab):
    return [v[0]+"_"+v[1] for v in tab]
```

4. La fonction suivante convient :

```
def creation_dossier(tab):
    for v in tab:
        os.mkdir("Contrats/"+v)
        os.chmod("Contrats/"+v, 774)
```

Exercice 2 (Arbres binaires de recherche)

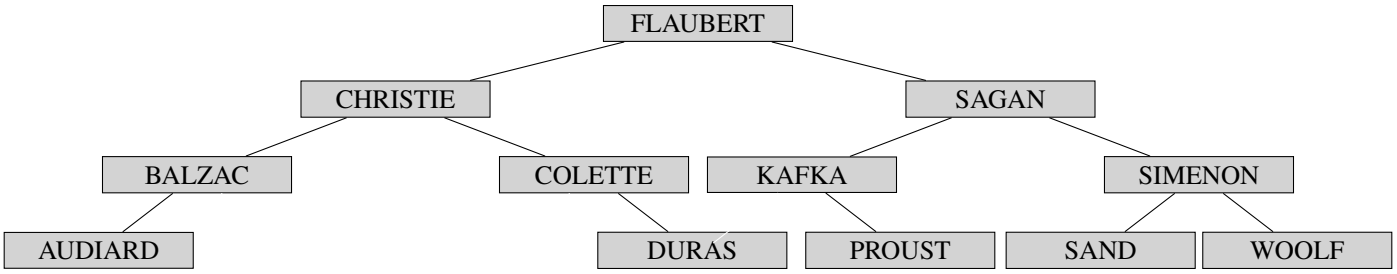
1. (a) On a l'arbre suivant :



   (b) La taille de l'arbre est 8 et sa hauteur est 4.

   (c) Pour une hauteur h, il est possible d'avoir $1 + 2 + 2^2 + ... + 2^{h-1} = 2^h - 1$.

2. On a l'arbre suivant :



3. La fonction `mystere` renvoie `VRAI` si l'auteur `t` est présent dans l'arbre `ABR` et `FAUX` dans le cas contraire. L'auteur `SIMENON` étant présent dans l'arbre `A2`, la fonction renvoie donc `VRAI`.

4. L'algorithme suivant convient :

```
fonction hauteur(ABR)
  si ABR ≠ NULL alors
    renvoyer 1 + MAX(hauteur(fils_gauche(ABR)), hauteur(fils_droit(ABR)))
  sinon
    renvoyer 0
  fin si
fin fonction
```

Exercice 3 (Programmation générale)

1. (a) Le choix 2 est le plus adapté. En effet, dans les choix 1, toutes les lignes du tableau seront liées les unes aux autres (toutes les lignes seront identiques). La modification d'une ligne entrainera donc la modification de toutes les autres lignes.

   (b) L'instruction permettant de modifier le tableau est `jeu[5][2]=1`.

2. (a) La fonction suivante convient :

```
def remplissage(n, jeu):
    for i in range(n):
        x = random.randint(0,7)
        y = random.randint(0,7)
        while jeu[y][x] != 0:
            x = random.randint(0,7)
            y = random.randint(0,7)
        jeu[y][x] = 1
```

   (b) La variable `n` doit être un entier compris entre 0 et 64 car il y a 64 cases dans le tableau.

3. La fonction suivante convient :

```
def nombre_de_vivants(i, j, jeu) :
    nb=0
    voisins = [(i-1,j-1), (i-1,j), (i-1,j+1), (i,j+1),(i+1,j+1),
               (i+1,j), (i+1,j-1), (i,j-1)]
    for e in voisins :
        if 0 <= e[0] < 8 and 0 <= e[1] < 8 :
            nb = nb + jeu[e[0]][e[1]]
    return nb
```

4. La fonction suivante convient :

```
def transfo_cellule(i, j, jeu):
    nb_v = nombre_de_vivants(i, j, jeu)
    if jeu[i][j] == 0 and nb_v == 3 :
        return 1
    if jeu[i][j] == 1 and (nb_v == 3 or nb_v == 2) :
        return 1
    return 0
```

Exercice 4 (Bases de données et SQL)

1. (a) La clé primaire de la relation `matchs` est `id_match`.

   (b) La relation `matchs` possède plusieurs clés étrangères : `id_creneau`, `id_terrain`, `id_joueur1` et `id_joueur2`.

2. (a) Le match a eu lieu le 1er août 2020 de 10h à 11h.

   (b) Il s'agit de Dupont Alice et Durand Belina

3. (a) La requête suivante convient :

```
SELECT prenom_joueur FROM joueurs WHERE nom_joueur = 'Dupont'
```

   (b) La requête suivante convient :

```
UPDATE joueurs
SET mdp = 1976
WHERE prenom_joueur = 'Dorine' AND nom_joueur = 'Dupont'
```

4. La requête suivante convient :

```
def produire(self):
    res = 0
    while self.stock_suffisant_brioche():
        self.qt_beurre = self.qt_beurre - 175
        self.qt_farine = self.qt_farine - 350
        self.nb_oeufs = self.nb_oeufs - 4
        res = res + 1
    return res
```

5. La requête suivante convient :

```
def nb_brioches(liste_stocks):
    nb = 0
    for s in liste_stocks:
        nb = nb + s.produire()
    return nb
```

Exercice 5 (Programmation générale)

1. La fonction suivante convient :

```
def mystere(dep):
    x = 0
    y = 0
    for c in dep:
        if c == '0':
            x = x+1
        else:
            y = y+1
    return [x, y]
```

2. (a) Au lieu d'avoir `while indice <= len(L)`, on devrait avoir `while indice < len(L)`. En effet, quand `indice` est égal à `len(L)`, nous allons avoir une erreur du type `list index out of range` (par exemple, pour un tableau de cinq éléments, l'indice du dernier élément est 4.

   (b) La fonction renvoie 0 alors qu'elle devrait renvoyer -2. Voici la version corrigée de la fonction :

```
def maxi(L) :
    indice = 1
    maximum = L[0]
    while indice < len(L) :
        if L[indice] > maximum :
            maximum = L[indice]
        indice = indice + 1
    return maximum
```

3. La variable `i` est de type nombre. Dans le `append` on essaye de concaténer une chaîne de caractère (`'Joueur'`) avec un nombre (`i`) ce qui va provoquer une erreur (on doit avoir deux chaînes de caractères pour effectuer une concaténation). Il est donc nécessaire de transformer le nombre en chaîne de caractère grâce à la fonction `str`. D'où la fonction corrigée :

```
from random import randint
def chemin(arrivee):
    deplacement = '00000000'
    while ..... :
        .....
        for k in range(8):
            pas = str(randint(0,1))
            ..... = deplacement + .....
    return deplacement
```

4. (a) L'appel `suite(6)` renvoie `21`.

   (b) Dans le cas où on exécute `suite(7)`, au cours des différents appels récursifs, `n` prend les valeurs suivantes : `7`, `5`, `3`, `2`, `1`, `-1`, `-3`, etc. Nous n'aurons jamais le cas de base (`n=0`). Les appels récursifs vont avoir lieu jusqu'au moment où la pile de récursion sera pleine. Nous aurons donc une erreur : `RecursionError: maximum recursion depth exceeded`

5. (a) Le premier `print` renvoie `(5, [10])`.

   (b) Le second `print` renvoie `(4, [10])` (`x` n'est pas modifiable par la fonction, mais la liste `L` est modifiable).