Éléments de correction sujet 09 (2022)

Exercice 1

1.
    a.
    l'utilisateur est *gestion*
    b.
    ls ./Contrats
2.
    a.
    mkdir ./Contrats/TURING_Alan
    b.
    chmod ug+rxw,o+r ./Contrats/TURING_Alan
3.

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

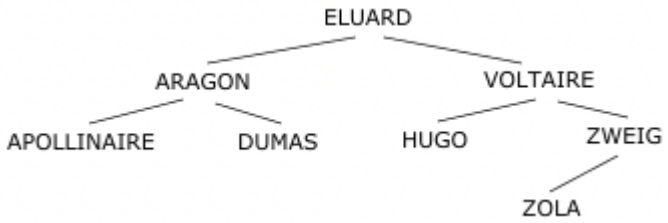
ou encore :

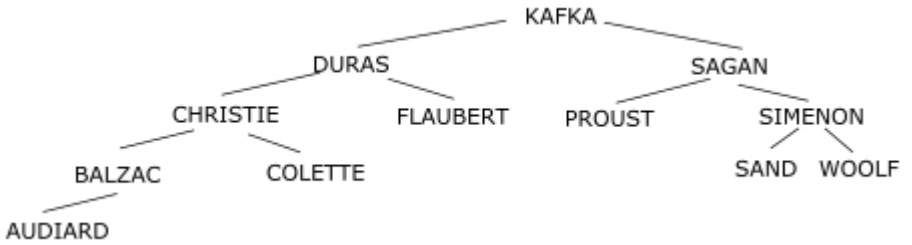
```
def formatage(tab):
    return [v[0]+"_"+v[1] for v in tab]
```

4.

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

Exercice 2

1.
   a.
   

   b.
   taille de l'arbre : 8 ; hauteur : 4
   c.
   Pour une hauteur h, il est possible d'enregistrer jusqu'à 2^h - 1 auteurs.
2.
   

3.
   La fonction mystère renvoie VRAI si l'auteur t est présent dans l'arbre ABR et FAUX dans le cas contraire. SIMENON est bien présent dans l'arbre A2, la fonction renvoie donc VRAI.
4.

```
Fonction hauteur(ABR) :
  SI ABR ≠ NULL :
       RENVOYER 1 + MAX(hauteur(fils_gauche(ABR)), hauteur(fils_droit(ABR)))
  SINON :
      RENVOYER 0
```

Exercice 3

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 la modification de toutes les autres lignes.
   b. L'instruction permettant de modifier le tableau : `jeu[5][2]=1`.

2.
   a.

```
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 (il y a 64 cases dans le tableau)

3.

```
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.

```
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

1.
    a. clé primaire de la relation matchs : 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.

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

    b.

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

4.

```
INSERT INTO joueurs
VALUES
(5, 'MAGID', 'Zora', 'zora', 2021)
```

5.

```
SELECT date
FROM matchs
JOIN joueurs ON matchs.id_joueur1 = joueurs.id_joueur OR
matchs.id_joueur2 = joueurs.id_joueur
WHERE prenom_joueur = 'Alice'
```

Exercice 5

1.

```
def somme(n) :
    total = 0
    for i in range(1,n) :
        total = total  + 1/i
    return total
```

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 5 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 chaine de caractère ("Joueur ") avec un nombre (i) ce qui va provoquer une erreur (on doit avoir 2 chaines de caractères pour effectuer une concaténation). Il est donc nécessaire de transformer le nombre en chaine de caractère grâce à la fonction str.

```
def genere(n):
    L = []
    for i in range(1, n+1) :
        L.append('Joueur '+str(i))
    return L
```

4.

a. 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…., 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. (5, [10])

b. 4 [10]