

## Éléments de correction sujet 06 (2024)

### Exercice 1

#### **Partie A**

1. attribut : self.itineraire ; méthode : remplir\_grille
2.  
a = 7 ; b = 4
3.  

```
def remplir_grille(self):  
    i, j = 0,  
    self.grille[0][0] = 'S'  
    for direction in self.itineraire:  
        if direction == 'D':  
            i = i + 1  
        elif direction == 'B':  
            j = j + 1  
        self.grille[i][j] = '*'  
    self.grille[self.largeur][self.longueur] = 'E'
```
4.  

```
def get_dimensions(self):  
    return (self.longueur, self.largeur)
```
5.  

```
def tracer_chemin(self):  
    self.remplir_grille()  
    for l in self.grille:  
        print(l)
```

#### **Partie B**

6.  

```
def itineraire_aleatoire(m, n):  
    itineraire = ''  
    i, j = 0, 0  
    while i != m and j != n:  
        dir = random.choice(['D', 'B'])  
        itineraire = itineraire + dir  
        if dir == 'D':  
            j = j + 1  
        else :  
            i = i + 1  
    if i == m:  
        itineraire = itineraire + 'D'*(n-j)  
    if j == n:  
        itineraire = itineraire + 'B'*(m-i)  
    return itineraire
```

## Partie C

7. Le tableau de dimension 1 x n est un tableau en une dimension. Le chemin est donc une ligne (on peut uniquement aller vers le bas), on a donc  $N(1,n) = 1$
8. Quand nous sommes sur la case de coordonnées (m,n), nous venons soit d'en haut (donc de la case de coordonnées (m, n-1), soit de la gauche (donc de la case de coordonnées (m-1, n)). Le nombre de chemins qui permettent d'atteindre la case de coordonnées (m,n) est donc égal au nombre de chemins qui permettent d'atteindre la case de coordonnées (m,n-1) plus le nombre de chemins qui permettent d'atteindre la case de coordonnées (m-1,n).  
D'où :  $N(m,n) = N(m-1,n) + N(m,n-1)$
9.

```
def nombre_chemins(m, n):  
    if m == 1 or n == 1:  
        return 1  
    else :  
        return nombre_chemins(m-1, n) + nombre_chemins(m, n-1)
```

## Exercice 2

1.

```
ls documents  
ou  
cd documents  
ls
```
2. le répertoire multimedia (et tout ce qu'il contient) se trouve désormais dans le répertoire documents
3. La classe Arbre permet de modéliser un arbre binaire (attributs self.gauche et self.droit). Comme l'arbre de la figure 1 n'est pas un arbre binaire (plus de 2 enfants), il n'est pas possible d'utiliser la classe Arbre pour modéliser l'arbre de la figure 1.
4. Nous avons un parcours de type préfixe.
5. parcours en largeur : home - documents - multimedia - cours - administratif - personnel - images - videos - films

## Partie B

6.

```
def est_vide(self):  
    return len(self.fils) == 0
```
7.

```
var_films = Dossier('films', [])  
var_videos = Dossier('videos',[var_films])  
var_images = Dossier('images', [])  
var_multimedia = Dossier('multimedia', [var_videos, var_images])
```

8.

```
def parcours(self):
    print(self.nom)
    for f in self.fils:
        f.parcours()
```

9.

Lors des appels récursifs, ont fini toujours par rencontrer un dossier vide (une feuille de l'arbre), ce qui provoque l'arrêt des appels récursifs. Donc la méthode parcours termine.

10.

```
def parcours(self):
    for f in self.fils:
        f.parcours()
    print(self.nom)
```

11.

La commande ls affiche uniquement les descendants directs alors que la méthode parcours affiche aussi les descendants des descendants...

12.

```
def mkdir(self, nom):
    nvx = Dossier(nom, [])
    self.fils.append(nvx)
```

13.

```
def contient(self, nom_dossier):
    file = []
    for d in self.fils:
        if d.nom == nom_dossier:
            return True
        file.append(d)
    while len(file) != 0:
        dossier = file.pop(0)
        for d in dossier.fils:
            if d.nom == nom_dossier:
                return True
            file.append(d)
    return False
```

14.

Il est possible de légèrement modifier le programme de la question 13 : à la place de renvoyer True, on renverrait le nom du dossier parent (self.nom pour la première boucle et dossier.nom pour la 2e boucle)

```
def parent(self, nom_dossier):
    file = []
    for d in self.fils:
        if d.nom == nom_dossier:
            return self.nom
        file.append(d)
    while len(file) != 0:
        dossier = file.pop(0)
        for d in dossier.fils:
            if d.nom == nom_dossier:
                return dossier.nom
            file.append(d)
    return False
```

15. On pourrait ajouter un attribut self.parent qui contiendrait le dossier parent.

### Exercice 3

#### **Partie A**

1. 11 en base 10 correspond à 1011 en binaire. Le bit de rang 0 (bit de poids faible nous indique que c'est un équipier, le bit de rang 1 nous indique que c'est un chef d'équipe, le bit de rang 2 nous indique que ce n'est pas un chef d'agrès et le bit de rang 3 nous indique que c'est un conducteur. Conclusion : c'est bien un chef d'équipe conducteur.
2. chef d'agrès conducteur : 1111 en binaire ce qui correspond à 15 en décimale
3. 4 en décimale correspond à 100. On aurait donc un chef d'agrès qui ne serait ni chef d'équipe et ni équipier, ce qui est impossible (voir énoncé)
4. En passant à un octet (8 bits), on aurait donc 4 bits supplémentaires, on pourrait donc coder 4 aptitudes supplémentaires.
5. Si on part du principe qu'il faut 10 caractères pour une aptitude et qu'un caractère est codé sur 1 octet, il faut donc 10 octets pour une aptitude et donc 40 octets pour 4 aptitudes. L'utilisation d'un octet pour coder 4 aptitudes nous fait donc "économiser" 39 octets, soit environ 98% d'économie mémoire (si on avait économisé 20 octets on aurait eu une économie de 50%, ici nous avons plus. Le seul choix possible est donc 98%).

#### **Partie B**

6. La clé primaire permet de distinguer chaque entrée car elle est unique pour chaque entrée. La clé étrangère permet d'effectuer une jointure entre 2 tables (permet de relier 2 tables entre elles).
7. Cette requête génère une erreur car il n'existe pas d'agrès ayant pour identifiant 1 dans la table agrès.
8.

```
UPDATE intervention
SET heure = '10:44:06'
WHERE jour = '2024-02-15' AND heure = '01:44:06'
```
- 9.

'Charlot'
'Red'
'Kevin'

10.

```
SELECT nom
FROM personnel
WHERE qualif >= 16 AND actif = 1
```

11.

Requête A : 2

Requête B : 1

**Attention** la requête B utilise un alias (utilisation du AS). Cela permet de remplacer dans le reste de la requête moyen par m et agres par a.

12.

```
SELECT DISTINCT(nom)
FROM personnel
JOIN agres ON idchefagres = matricule
WHERE jour = '2024-02-15'
```

13.

```
SELECT DISTINCT(nom)
FROM moyen
JOIN agres ON moyen.idagres = agres.id
JOIN intervention ON moyen.idinter = intervention.id
JOIN personnel ON agres.idchefagres = personnel.id
WHERE intervention.jour = '2024-06-11'
```