Exercice 1 (6 points)

Cet exercice porte sur la programmation en Python en général, la programmation orientée objet et la récursivité.

Partie A – Programmation orientée objet

1.

- un attribut de la classe Chemin : self.itineraire ou self.longueur ou self..largeur ou self.grille ou itineraire ou longueur ou largeur ou grille
- une méthode de la classe Chemin : remplir_grille

2.

```
chemin_1 = Chemin("DDBDBBDDDDB")
a = chemin_1.largeur
b = chemin_1.longueur
```

```
>>> a
4
>>> b
7
```

- valeur de a : 4
- valeur de b : 7

3.

```
15      def remplir_grille(self):
16          i, j = 0, 0 # Position initiale
17          self.grille[0][0] = 'S' # Case de départ marquée d'un S
18          for direction in self.itineraire:
19              if direction == 'D':
20                  j = j + 1 # Déplacement vers la droite
21              elif direction == 'B':
22                  i = i + 1 # Déplacement vers le bas
23              self.grille[i][j] = '*' # Marquer le chemin avec '*'
24          self.grille[self.largeur][self.longueur] = 'E' #Case d'a
```

4.

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

```
>>> chemin_1.get_dimensions()
(7, 4)
```

5.

```
29      def tracer_chemin(self):
30          marge = 0
31          for i in range(len(self.grille)):
32              print(' '*marge, end='')
33              for j in range(len(self.grille[i])):
34                  if self.grille[i][j] != '.':
35                      print(self.grille[i][j], end='')
36                      marge += 1
37              print()
```

```
>>> chemin_1.tracer_chemin()
 S**
    **
      *
       *****
            E
```

Partie B – Génération aléatoire d'itinéraires

6.

```
def itineaire_aleatoire(m, n):
    itineraire = ''
    i, j = 0, 0
    while i != m and j != n:
        deplace = choice(['D', 'B']) # il y a pl
        itineraire += deplace
        if deplace == 'D':
            j += 1
        elif deplace == 'B':
            i += 1
    if i == m:
        itineraire = itineraire + 'D'*(n-j)
    if j == n:
        itineraire = itineraire + 'B'*(m-i)
    return itineraire
```

```
>>> itineaire_aleatoire(7, 4)
'BBBDDBDDBBB'
```

Partie C – Calcul du nombre de chemin possibles

7.

N(1, n) : Nombre de chemins de longueur 1 et de largeur n de dimension 1×n
si longueur = 1 avec S en position de coordonnées (0 ; 0)
alors tous les déplacements sont vers le bas (B), donc n fois B
donc il n'y a qu'un seul chemin vertical vers la bas.

De même, N(m, 1) = 1
si largeur = 1 avec S en position de coordonnées (0 ; 0)
alors tous les déplacements sont vers la droite D, donc m fois D
donc il n'y a qu'un seul chemin horizontal vers la droite.

8. Justifier que N(m, n) = N(m-1, n) + N(m, n-1)
N(m, n) : m+n déplacement : m à droite D et n en bas B
m+n-1 plus le dernier déplacement soit à droite D, soit en bas
N(m-1, n) : m-1+n déplacement : m-1 à droite D et n en bas B → le dernier est à droite D
N(m, n-1) : m+n-1 déplacement : m à droite D et n-1 en bas B → le dernier est en bas Bdonc 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
    return nombre_chemins(m-1, n) + nombre_chemins(m, n-1)
```

```
>>> nombre_chemins(1, 4)
1
>>> nombre_chemins(7, 1)
1
>>> nombre_chemins(2, 4)
4
```

```
def nombre_chemins(m, n):
    if m==1 or n==1:
        return 1
    print(f"nombre_chemins({m-1, n}) + nombre_chemins({m, n-1})")
    return nombre_chemins(m-1, n) + nombre_chemins(m, n-1)
```

```
>>> nombre_chemins(2, 4)
  nombre_chemins((1, 4)) + nombre_chemins((2, 3))
  nombre_chemins((1, 3)) + nombre_chemins((2, 2))
  nombre_chemins((1, 2)) + nombre_chemins((2, 1))
4
```

```
>>> nombre_chemins(5, 3)
  nombre_chemins((4, 3)) + nombre_chemins((5, 2))
  nombre_chemins((3, 3)) + nombre_chemins((4, 2))
  nombre_chemins((2, 3)) + nombre_chemins((3, 2))
  nombre_chemins((1, 3)) + nombre_chemins((2, 2))
  nombre_chemins((1, 2)) + nombre_chemins((2, 1))
  nombre_chemins((2, 2)) + nombre_chemins((3, 1))
  nombre_chemins((1, 2)) + nombre_chemins((2, 1))
  nombre_chemins((3, 2)) + nombre_chemins((4, 1))
  nombre_chemins((2, 2)) + nombre_chemins((3, 1))
  nombre_chemins((1, 2)) + nombre_chemins((2, 1))
  nombre_chemins((4, 2)) + nombre_chemins((5, 1))
  nombre_chemins((3, 2)) + nombre_chemins((4, 1))
  nombre_chemins((2, 2)) + nombre_chemins((3, 1))
  nombre_chemins((1, 2)) + nombre_chemins((2, 1))
15
```

```
>>> nombre_chemins(7, 4)
  nombre_chemins((6, 4)) + nombre_chemins((7, 3))
  nombre_chemins((5, 4)) + nombre_chemins((6, 3))
  nombre_chemins((4, 4)) + nombre_chemins((5, 3))
  nombre_chemins((3, 4)) + nombre_chemins((4, 3))
  nombre_chemins((2, 4)) + nombre_chemins((3, 3))
  nombre_chemins((1, 4)) + nombre_chemins((2, 3))
  nombre_chemins((1, 3)) + nombre_chemins((2, 2))
...
  nombre_chemins((4, 2)) + nombre_chemins((5, 1))
  nombre_chemins((3, 2)) + nombre_chemins((4, 1))
  nombre_chemins((2, 2)) + nombre_chemins((3, 1))
  nombre_chemins((1, 2)) + nombre_chemins((2, 1))
84
```

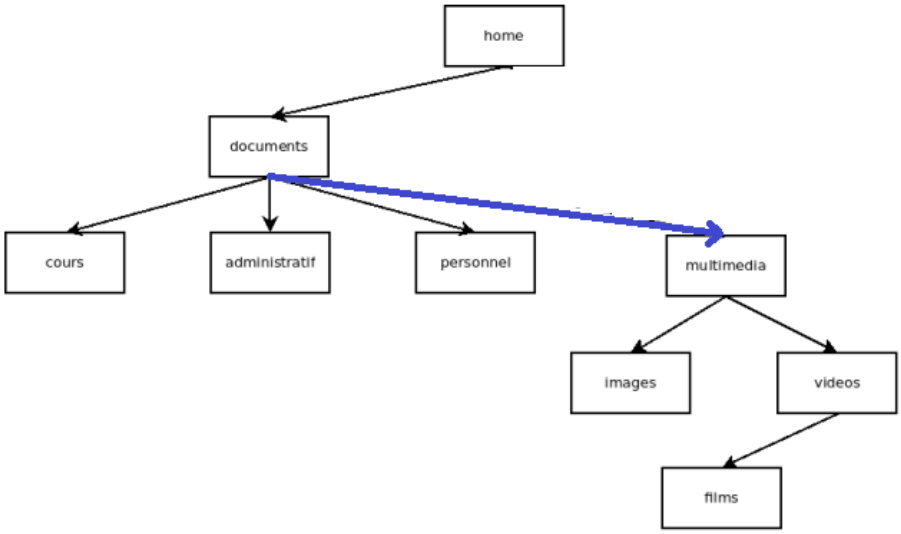
Exercice 2 (6 points)

Cet exercice porte sur la programmation objet, la récursivité, les arbres binaires et les systèmes d'exploitation

Partie A

1. **ls documents** ou ls home/documents

2. mv .../../multimedia /home/documents :
La modification apportée à l'arborescence est que le dossier multimedia situé dans le dossier(home) situés 2 niveaux/branches au-dessus du dossier courant(cours) est déplacé vers le répertoire documents situé dans le répertoire home.
Le sous-arbre de home de racine multimedia est déplacé en sous-arbre de racine documents



3.
Le code ne permet de modéliser que des arbres binaires de nœud à maximum deux branches/fils. Or l'arborescence de fichiers de la figure1 à des nœuds(dossiers) à plus de deux branches(ex le nœud dossier documents à 3 branches/fils).

4. parcours **préfixe** (racine→sag→sad)

5. parcours **en largeur** :
home → documents → multimedia → cours → administratif → personnel → images → videos → films

Partie B

6.

```
def est_vide(self):
    return self.fils == []
```

```
>>> films = Dossier('films', []) ; films.est_vide()
True
>>> videos = Dossier('videos', [Dossier('films', [])]) ; videos.est_vide()
False
```

7.

```
var_multimedia = Dossier('multimedias',
                         [Dossier('images', []),
                          Dossier('videos', [Dossier('films', [])])])
```

8.

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

```
>>> var_multimedia.parcours()
multimedias
images
videos
films
```

9. parcours est une fonction récursive qui se termine si la liste self.fils des dossiers enfants est vide
parcours parcourt tous les Dossiers donc termine sur une arborescence de fichiers

10.

```
def parcours_s(self):
    if not self.est_vide():
        for f in self.fils:
            f.parcours()
    print(self.nom)
```

```
>>> var_multimedia.parcours_s()
images
videos
films
multimedias
```

11.

• appel de la méthode parcours : affiche les noms de tous les dossiers et sous-dossiers de l'arborescence à partir du dossier actuel self parcouru récursivement dans l'ordre postfixe | méthode de la classe Dossier en Python

• execution de la commande UNIX ls : affiche/liste le contenu d'un répertoire/dossier dans un ordre spécifique(alphabétique …) | commande intégrée au terminal de l'OS type UNIX

ls -R pour un parcours récursif de l'arborescence

12.

```
def mkdir(self, nom):
    """ var_videos.mkdir("documentaires")
    crée un dossier documentaires vide dans le dossier var_videos"""
    dossier = Dossier(nom, [])
    self.fils.append(dossier)
```

```
>>> var_videos = Dossier('videos', [Dossier('films', [])])
>>> var_videos.mkdir("documentaires")
>>> var_videos.parcours()
videos
films
documentaires
```

13.

```
def contient(self, nom_dossier):
    """ ▸ True si arborescence de racine self contient au moins nom_dossier,
    False sinon"""
    if self.nom == nom_dossier:
        return True
    else:
        for f in self.fils:
            return f.contient(nom_dossier)
    return False
```

```
>>> var_videos.contient('films')
True
>>> var_videos.contient('NSI')
False
```

14. Avec l'implémentation de la classe Dossier de cette partie, expliquer comment il serait possible de déterminer le dossier parent d'un dossier donné dans une arborescence donnée. On attend ici l'idée principale de l'algorithme décrite en français. On ne demande pas d'implémenter cet algorithme en Python.

• méthode get_parent(self, nom_dossier) :
pour chaque dossier de self.fils
- si dossier.nom est nom_dossier :
  alors retourner self.nom # le parent de dossier.nom
- sinon :
  appliquer la methode get_parent à dossier avec le même nom_dossier
  #appel récursif

```
    def get_parent(self, nom_dossier):
        for dossier in self.fils:
            if dossier.nom == nom_dossier:
                return self
            else:
                parent = dossier.get_parent(nom_dossier)
                if parent:
                    return self
        return None
```

```
>>> var_videos.get_parent('films').nom
'videos'
>>> var_videos.get_parent('documentaires').nom
'videos'
```

15.

```
    def __init__(self, nom, liste, parent=None):
        self.nom = nom
        self.fils = liste # liste d'objets de la classe Dossier
        self.parent = parent
        for dossier in self.fils:
            dossier.parent = self
```

Choix de plus de plus de simplicité :
• On attribue le parent du dossier directement à l'instanciation de l'objet dans la classe Dossier

```
var_videos = Dossier('videos', [Dossier('films', [], 'videos')], 'multimedia')
>>> var_videos.parent
'multimedia'
```

Exercice 3 (8 points)

Cet exercice porte sur le codage binaire, les bases de données relationnelles et les requêtes SQL.

Partie A – Encodage binaire

1.

8	4	2	1
2^3	2^2	2^1	2^0
1 : conducteur	1 : chef agrès	1 : chef équipe	0 : non formé
			1 : équipier

qualification chef d'équipe conducteur : chef d'équipe donc équipier et conducteur :
codage binaire : **1011**
codage décimale : **11** (8+0+2+1)

2.
qualification chef d'agrès conducteur : chef d'équipe donc chef équipe et équipier et conducteur :
codage binaire : **1111**
codage décimale : **15** (8+4+2+1)

3.
codage décimale : 4
codage binaire : 0100
qualification uniquement chef d'agrès pas possible car aussi chef d'équipe et équipier :
(codage binaire : 0111 ; codage décimale 7)

4.
avec ce codage sur un octet on peut définir **4 autres aptitudes de rang 4, 5, 6, et 7**

5.
4 aptitudes codées chacune par une chaine de 10 caractères de 1 octet chacun :
donc codées sur 4×10×1 = 40 octets
donc l'encodage sur 1 octet permet une économie mémoire de 39 octets
soit 39/40*100 % donc environ **98% d'économie mémoire**

Partie B – Encodage binaire

6.
• **clé primaire** : attribut unique et non nul d'une table
• **clé étrangère** : attribut d'une table en référence à une clé primaire d'une autre table pour lier les deux tables

7.
```
INSERT INTO moyen (idagres, idinter) VALUES (1,5);
```
► génère une erreur car :
• on ne peut pas insérer la clé étrangère idagres avec la valeur 1 car cette valeur n'est pas affectée à la clé primaire id de la table agres à laquelle elle doit faire référence ;
• on ne peut pas insérer la clé étrangère idinter avec la valeur 5 car cette valeur n'est pas affectée à la clé primaire id de la table intervention à laquelle elle doit faire référence

8.
```
UPDATE intervention SET heure='10:44:06' WHERE id = 3 ;
```

9.

```
SELECT nom FROM personnel
WHERE actif = 0;
```

nom
'Charlot'
'Red'
'Kevin'

10.

```
SELECT nom FROM personnel
WHERE qualif >= 16;
```

11.

Requête A

```
SELECT COUNT(*) FROM agres
WHERE jour = '2024-03-27';
```

COUNT(*)
2

affiche le nombre de départ de véhicule (agrès) la journée du 27 mars 2024

Requête B

```
SELECT COUNT(*) FROM moyen AS m
INNER JOIN agres AS a ON a.id = m.idagres
WHERE a.jour = '2024-03-27';
```

m
1

affiche le nombre de véhicule (agrès) sortis la journée du 27 mars 2024

12. Proposer une requête qui renvoie sans répétition tous les noms des chefs d'agrès assignés à un véhicule le 15 février 2024.

```
SELECT peronnel.nom FROM personnel
JOIN agres ON agres.idchefA = peronnel.matricule
WHERE agres.jour = '2024-02-15';
```

ou

```
SELECT peronnel.nom FROM personnel, agres
WHERE agres.idchefA = peronnel.matricule
AND agres.jour = '2024-02-15';
```

13. Proposer une requête qui renvoie sans répétition tous les noms des chefs d'agrès engagés en intervention le 11 juin 2024.

```
SELECT peronnel.nom FROM personnel
JOIN agres ON agres.idchefA = peronnel.matricule
JOIN intervention ON intervention.jour = agres.jour
WHERE intervention.jour = '2024-06-11';
```

ou

```
SELECT peronnel.nom FROM personnel, agres, intervention
WHERE agres.idchefA = peronnel.matricule
AND intervention.jour = agres.jour
AND intervention.jour = '2024-06-11';
```