Mayotte - mai 2022 - sujet 1

Exercice 1 (Piles et files - 4 points)
On cherche ici à mettre en place des algorithmes qui permettent de modifier l'ordre des informations contenues dans une file. On considère pour cela les structures de données abstraites de Pile et File définies par leurs fonctions primitives suivantes :

⋆ **Pile :**

- `creer_pile_vide()` renvoie une pile vide ;
- `est_pile_vide(p)` renvoie `True` si la pile `p` est vide, `False` sinon ;
- `empiler(p, element)` ajoute `element` au sommet de la pile `p` ;
- `depiler(p)` renvoie l'élément se situant au sommet de la pile `p` en le retirant de la pile `p` ;
- `sommet(p)` renvoie l'élément se situant au sommet de la pile `p` sans le retirer de la pile `p`.

⋆ **File :**

- `creer_file_vide()` renvoie une file vide ;
- `est_file_vide(f)` renvoie `True` si la file `f` est vide, `False` sinon ;
- `enfiler(f, element)` ajoute `element` dans la file `f` ;
- `defiler(f)` renvoie l'élément à la tête de la file `f` en le retirant de la file `f`.

On considère de plus que l'on dispose d'une fonction permettant de connaître le nombre d'éléments d'une file :

- `taille_file(f)` renvoie le nombre d'éléments de la file `f`.

On représentera les files par des éléments en ligne, l'élément de droite étant la tête de la file et l'élément de gauche étant la queue de la file. On représentera les piles en colonnes, le sommet de la pile étant le haut de la colonne. Par exemple :

La file suivante est appelée `f` :

4	3	8	2	1

La pile suivante est appelée `p` :

5
8
6
2

1. Les quatre questions suivantes sont indépendantes. Pour chaque question, on repartira de la pile `p` et de la file `f` initiales (présentées ci-dessus).
   (a) Représenter la file `f` après l'exécution du code suivant :

```
enfiler(f, defiler(f))
```

   (b) Représenter la pile `p` après l'exécution du code suivant :

```
empiler(p, depiler(p))
```

   (c) Représenter la pile `p` et la file `f` après l'exécution du code suivant :

```
for i in range(2):
    enfiler(f, depiler(p))
```

   (d) Représenter la pile `p` et la file `f` après l'exécution du code suivant :

```
for i in range(2):
    empiler(p, defiler(f))
```

2. On donne ici une fonction `mystere` qui prend une file en argument, qui modifie cette file, mais qui ne renvoie rien.

```
def mystere(f):
    p = creer_pile_vide()
    while not est_file_vide(f):
        empiler(p, defiler(f))
    while not est_pile_vide(p):
        enfiler(f, depiler(p))
    return p
```

Préciser l'état de la variable `f` après chaque boucle de la fonction `mystere` appliquée à la file | 1 | 2 | 3 | 4 |. Indiquer le contenu de la pile renvoyée par la fonction.

3. On considère la fonction `knuth` suivante dont le paramètre est une file :

```
def knuth(f):
    p=creer_pile_vide()
    N=taille_file(f)
    for i in range(N):
        if est_pile_vide(p):
            empiler(p, defiler(f))
        else:
            e = defiler(f)
            if e >= sommet(p):
                empiler(p, e)
            else:
                while not est_pile_vide(p) and e < sommet(p):
                    enfiler(f, depiler(p))
                empiler(p, e)
    while not est_pile_vide(p):
        enfiler(f, depiler(p))
```

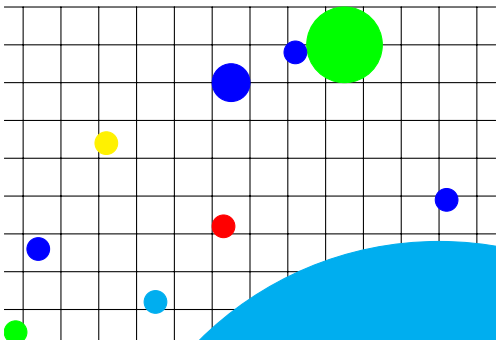
(a) Recopier et compléter le tableau ci-dessous qui détaille le fonctionnement de cet algorithme étape par étape pour la file | 2 | 1 | 3 |. Une étape correspond à une modification de la pile ou de la file. Le nombre de colonnes peut bien sûr être modifié.

f	| 2 | 1 | 3 |	| 2 | 1 |						
p	⌊ ⌋	| 3 |						

(b) Que fait cet algorithme ?

Exercice 2 (POO - 4 points)

Dans un jeu de plateforme, des bulles de couleurs et de diamètres différents se déplacent de manière aléatoire. A chaque fois qu'une bulle touche une bulle plus grande, la petite cède son contenu à la plus grande, et donc celle-ci augmente de surface. Par exemple, si une bulle de 1 cm^2 rencontre une bulle de 4 cm^2, la petite bulle disparait et la plus grande a désormais une surface de 5 cm^2. A chaque collision, la vitesse de la grande bulle est réduite de moitié.



Le développeur a choisi de coder en Python. Chaque bulle est un objet disposant entre autre des attributs suivants :

- `xc`, `yc` sont deux entiers : les coordonnées du pixel placé au centre de la bulle ;
- `rayon` est un entier : le rayon de la bulle en pixels ;
- `couleur` est un entier : la couleur de la bulle ;
- `dirx`, `diry` sont deux décimaux (float) qui déterminent les déplacements à l'horizontale et à la verticale à chaque fois que la bulle se déplace. Ces deux valeurs déterminent donc la direction et la vitesse de la bulle. Par exemple si `dirx` vaut `0.5` et `diry` vaut `0.0`, la bulle se déplace vers la droite uniquement alors que si `dirx` vaut `-1.0` et `diry` vaut `0.0`, la bulle se déplace vers la gauche et deux fois plus vite que précédemment.

On suppose que toutes les fonctions de la bibliothèque `math` ont déjà été importées par l'instruction `from math import *`.

La fonction `randint` de la bibliothèque `random` prend en paramètre deux entiers et renvoie un entier alétoire dans la plage définie par les deux paramètres. Par exemple, `randint (-1, 5)` peut renvoyer une des valeurs suivantes : `-1`, `0`, `1`, `2`, `3`, `4`, `5`.

1. Pour simplifier, on se limitera à un jeu de six bulles. Au départ, on crée une liste appelée `Mousse` de longueur six contenant six emplacements vides :

   `Mousse = [None, None, None, None, None, None].`

   Le code ci-dessous montre le début du programme et notamment la structure définition de la classe nommée `Cbulle` ainsi que le code permettant le déplacement d'une bulle.

```
from random import randint
from math import *

class Cbulle:
    def __init__(self):
        self.xc = randint(0, 100)
        self.yc = randint(0, 100)
        self.rayon = randint(0, 10)
        self.dirx = float(randint(-1, 1)) # dirx et diry valent
        self.diry = float(randint(-1, 1)) # -1.0 ou 0.0. ou 1.0
        self.couleur = randint(1,65535)

    def bouge(self):
        # déplace la bulle
        self.xc = self.xc + self.dirx
        self.yc = self.yc + self.diry
```

   On crée les six bulles une à une et ces objets sont stockés dans les emplacements vides de la liste `Mousse`.

   `Mousse = [bulle1, bulle2, bulle3, bulle4, bulle5, bulle6].`

Lors d'une collision, la bulle la plus petite disparait et est remplacée dans la liste par la valeur `None` tandis que la plus grosse a sa surface qui augmente.

Au cours d'une partie, si une ou plusieurs bulles ont disparue, le programme peut en introduire de nouvelles dans le jeu : dans ce cas, lorsqu'une nouvelle bulle apparaît, elle remplace le premier `None` de la liste `Mousse`.

(a) Recopier les quatre dernières lignes et compléter les `.....` du code Python ci-dessous.

```
def donnePremierIndiceLibre(Mousse):
    """
    Mousse est une liste.
    La fonction doit renvoyer lindice du premier
    emplacement libre (contenant None) dans la liste Mousse
    ou renvoyer 6 en labsence dun emplacement libre dans
    Mousse.
    """
    i = 0
    while ..............  and Mousse[i] != None :
        ..............
    return i
```

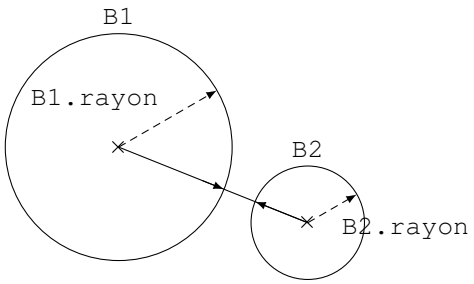
(b) Lorsque le jeu crée une bulle (instance de la classe `Cbulle`), il doit ensuite la placer dans la liste `Mousse` à la place d'un `None`.
Ecrire la fonction `placeBulle` qui reçoit en paramètre un objet `B` de type `Cbulle` et qui place cet objet dans la liste `Mousse`. Cette fonction ne renvoie rien, mais la liste `Mousse` est modifiée. Si aucun emplacement n'est disponible, la fonction ne modifie rien.

2. Pour le bon déroulement du jeu, on a besoin aussi d'une fonction `bullesEnContact` qui prend en paramètre deux objets `B1` et `B2` de type `Cbulle` et qui renvoie `True` si la bulle `B2` touche la bulle `B1` et `False` dans le cas contraire.

On peut remarquer que deux bulles sont en contact si la distance qui sépare leur centre est inférieure ou égale à la somme de leurs rayons.

On dispose de la fonction `distanceEntreBulles(B1, B2)` qui calcule et renvoie la distance entre les centres de bulles `B1` et `B2`.

Ecrire la fonction `bullesEnContact(B1, B2)`.



3. Quand une petite bulle touche une plus grosse bulle, on appelle la fonction `collision`, ci-dessous, où `indPetite` est l'indice de la petite bulle et `indGrosse` l'indice de la grosse bulle dans `Mousse`.
Recopier et compléter les `.....` de la fonction `collision`.

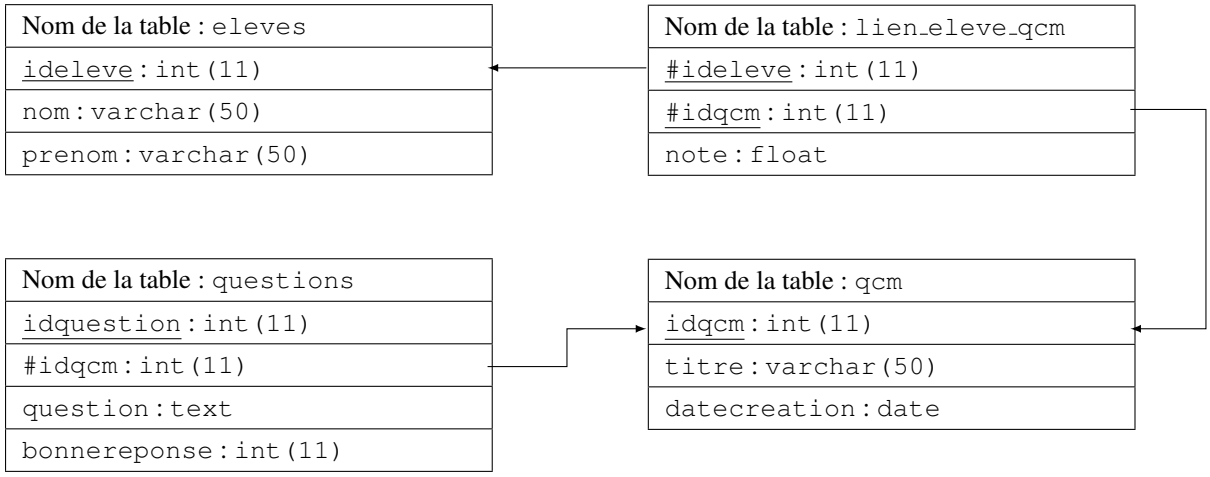
```
def collision(indPetite, indGrosse, mousse) :
    """
    Absorption de la plus petite bulle dindice indPetite
    par la plus grosse bulle dindice indGrosse. Aucun test
    nest réalisé sur les positions.
    """
    # calcul du nouveau rayon de la grosse bulle
    surfPetite = pi*Mousse [indPetite].rayon**2
    surfGrosse = pi*Mousse [indGrosse].rayon**2
    surfGrosseApresCollision = ..............................
    rayonGrosseApresCollision = sqrt(surfGrosseApresCollision/pi)
    #réduction de 50% de la vitesse de la grosse bulle
    Mousse[indGrosse].dirx = ..............................
    Mousse [indGrosse].diry = ..............................
    #suppression de la petite bulle dans Mousse
    ........................................
```

Exercice 3 (Bases de données et SQL - 4 points)

Un rappel sur la syntaxe de quelques fonctions SQL est donné en annexe 1 en fin de sujet.

Un enseignant a mis en place un site web qui permet à ses élèves de faire des QCM (questionnaire à choix multiples) de NSI en ligne.

L'enseignant a créé une base de données nommée `QCM_NSI` pour gérer ses QCM, contenant les quatre relations (appelées aussi communément « tables ») du schéma relationnel ci-dessous :



Dans le schéma relationnel précédent, un attribut souligné indique qu'il s'agit d'une clé primaire. Un attribut précédé du symbole `#` indique qu'il s'agit d'une clé étrangère et la flèche associée indique l'attribut référencé. Ainsi, par exemple, l'attribut `ideleve` de la relation `lien_eleve_qcm` est une clé étrangère qui fait référence à l'attribut `ideleve` de la relation `eleves`.

Dans le cas de la relation `lien_eleve_qcm` la clé primaire est composée de l'association des deux attributs `ideleve` et `idqcm`, eux-mêmes étant des clés étrangères.

On donne ci-dessous le contenu exhaustif des relations :

Table **`eleves`**

ideleve	nom	prenom
2	Dubois	Thomas
3	Dupont	Cassandra
4	Marty	Mael
5	Bikila	Abebe

Table **`lien_eleve_qcm`**

ideleve	idqcm	note
2	1	12
2	3	18
2	4	13
2	5	15
3	1	20
3	2	9
3	3	18
3	5	13
4	4	15
4	5	20
5	4	15

Table **`qcm`**

idqcm	titre	datecreation
1	Base de données	2021-09-20
2	POO	2022-04-08
3	Arbre binaire	2022-01-09
4	Arbre parcours	2022-02-15
5	Piles-Files	2021-12-05

1. (a) Que renvoie la requête suivante ?

```
SELECT titre FROM qcm WHERE datecreation > '2022-01-10' ;
```

   (b) Ecrire une requête qui donne les notes de l'élève qui a pour identifiant `4`.

2. (a) Sachant que la clé primaire de la relation `lien_eleve_qcm` est composée de l'association des deux attributs `ideleve` et `idqcm`, expliquer pourquoi avec ce schéma relationnel, un élève ne peut pas faire deux fois le même QCM.

(b) L'élève Marty Mael vient de faire le QCM sur la POO et a obtenu une note de 18.
Comment est/sont modifiée(s) le(s) table(s) ? Il n'est pas demandé d'écrire une requête SQL.

(c) Un nouvel élève (nom : Lefèvre, prénom : Kevin) est enregistré. Ecrire la requête permettant la mise à jour du/des relation(s).

(d) L'élève Dubois Thomas quitte l'établissement et toutes les références à cet élève doivent être supprimées des relations. Pour la relation `lien_eleve_qcm`, écrire la requête pour supprimer toutes les références à l'élève Dubois Thomas qui a pour identifiant `2`.

3. (a) Recopier et compléter les `.....` de la requête suivante pour qu'elle affiche la liste des noms et prénoms des élèves ayant fait le QCM d'`idqcm` égal à `4`.

```
SELECT ............................. FROM eleves
JOIN lien_eleve_qcm ON eleves.ideleve = ..............................
WHERE ............................. ;
```

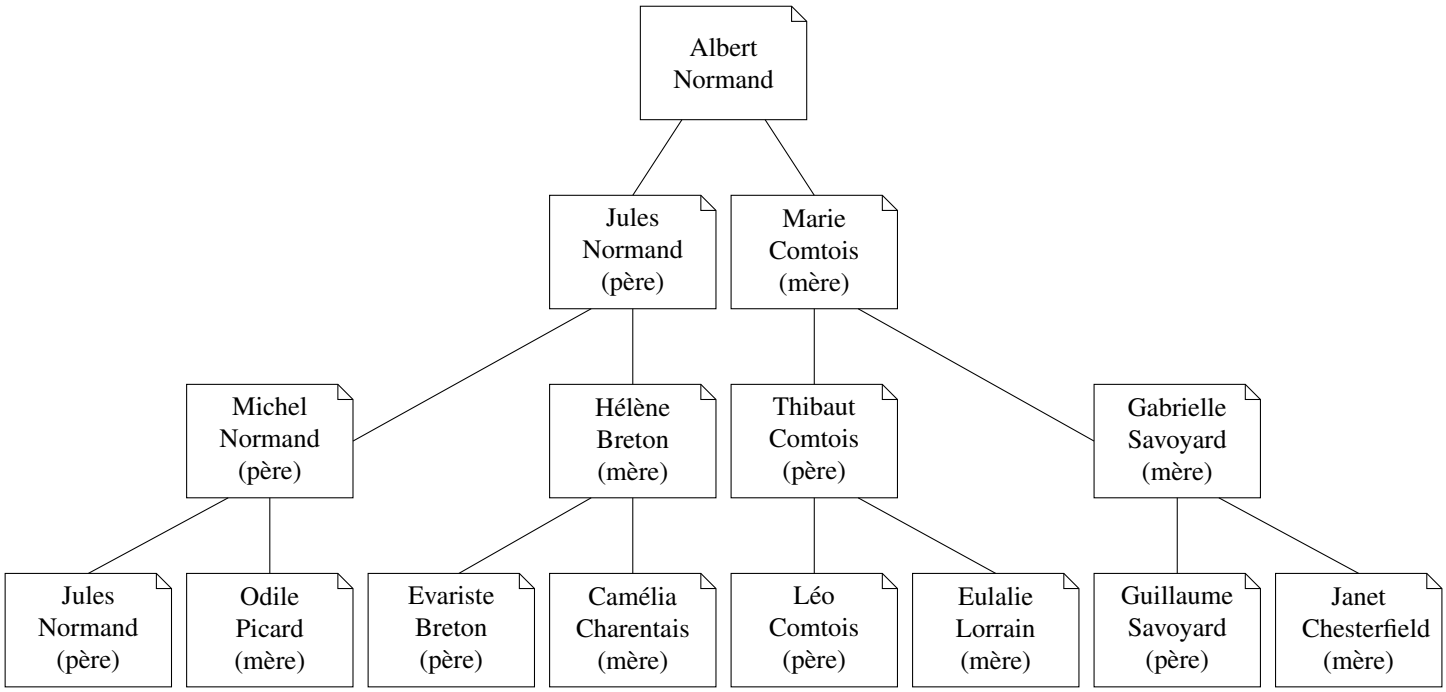
(b) Donner le résultat de la requête de la question 3.(a).

4. Ecrire une requête qui affiche le nom, le prénom et la note des élèves ayant fait le QCM `Arbre binaire`. L'utilisation des trois tables dans la requête est attendue.

Exercice 4 (Arbres binaires - 4 points)
On s'intéresse dans cet exercice à l'étude d'un arbre généalogique.
Voici un extrait de l'arbre généalogique fictif d'une personne nommée Albert Normand.

L'arbre généalogique est présenté avec les parents vers le bas et les enfants vers le haut.

Albert Normand est considéré comme la génération 0. On considère ses parents comme la génération 1, ses grands-parents comme la génération 2 et ainsi de suite pour les générations précédentes.



MODELISATION DE L'ARBRE :
L'arbre généalogique d'un individu est modélisé par un arbre :

- chaque nœud de l'arbre représente un individu ;
- le premier nœud du sous-arbre gauche d'un individu est associé à son père ;
- le premier nœud du sous-arbre droit est associé à sa mère.

IMPLEMENTATION DE L'ARBRE :
Pour implémenter l'arbre, on utilise des notions de programmation orientée objet. Chaque nœud de l'arbre est représenté par un objet qui est l'instance d'une classe `Noeud` ayant trois attributs. Ainsi l'objet `N` de type `Noeud` aura les attributs suivants :

- `N.identite` de type tuple : `(prenom,nom)` de l'individu référencé par l'objet `N` ;
- `N.gauche` de type arbre binaire : le sous-arbre gauche de l'objet `N` ;
- `N.droit` de type arbre binaire : le sous-arbre droit de l'objet `N`.

Pour un individu, référencé par l'objet `N` de type `Noeud`, dont on ne connait pas les parents, on considèrera que `N.gauche` et `N.droit` ont la valeur `None`.

1. (a) Expliquer en quoi cet arbre généalogique est un arbre binaire.
   (b) Pourquoi un arbre généalogique n'est pas un arbre binaire de recherche (ABR) ?
2. On souhaite obtenir la liste de tous les ascendants (ancêtres) d'Albert Normand. Pour cela, on utilise un parcours en profondeur de l'arbre.
   (a) Ecrire les sept premières personnes (nom et prénom) rencontrées si on utilise le parcours en profondeur préfixe.
   (b) Ecrire les sept premières personnes (nom et prénom) rencontrées si on utilise le parcours en profondeur infixe.

   On donne ci-dessous le code incomplet de la fonction d'un parcours en profondeur de l'arbre, dans lequel il manque la ligne correspondant à l'instruction d'affichage du prénom et du nom de l'individu :

```
def parcours(racine_de_l_arbre) :
    if racine_de_l_arbre != None :
        noeud_actuel = racine_de_l_arbre
        parcours(noeud_actuel.gauche)
        parcours(noeud_actuel.droite)
```

(c) Recopier et compléter l'algorithme ci-dessus en y insérant au bon endroit la ligne contenant l'instruction d'affichage pour que cet algorithme corresponde à un parcours en profondeur préfixe.

(d) Recopier et compléter l'algorithme ci-dessus en y insérant au bon endroit la ligne contenant l'instruction d'affichage pour que cet algorithme corresponde à un parcours en profondeur infixe.

3. On souhaite maintenant préciser la génération d'un individu dans l'implémentation de l'arbre généalogique. Lors de la création de l'instance, on donnera la valeur `0` par défaut.

(a) Recopier et compléter la définition de la classe `Noeud` pour ajouter un attribut `generation` qui indique la génération d'un individu.

```
class Noeud() :
    def __init__(prenom, nom) :
        self.identite = (prenom, nom)
        self.gauche = None
        self.droite = None
        ..............................
```

(b) Ecrire la fonction récursive `numerotation` qui parcourt l'arbre et modifie l'attribut `generation` de tous les ancêtres d'Albert Normand, de sorte que les parents d'Albert Normand soient la génération 1, etc.
Cette fonction prend en paramètres `racine_de_l_arbre` de type `Noeud` et `num_gen` de type entier.

```
def numerotation(racine_de_l_arbre, num_gen=0) :
    ..........
```

4. On donne la fonction suivante qui prend en paramètres l'objet `N` de type `Noeud` et la variable `affiche` de type booléen :

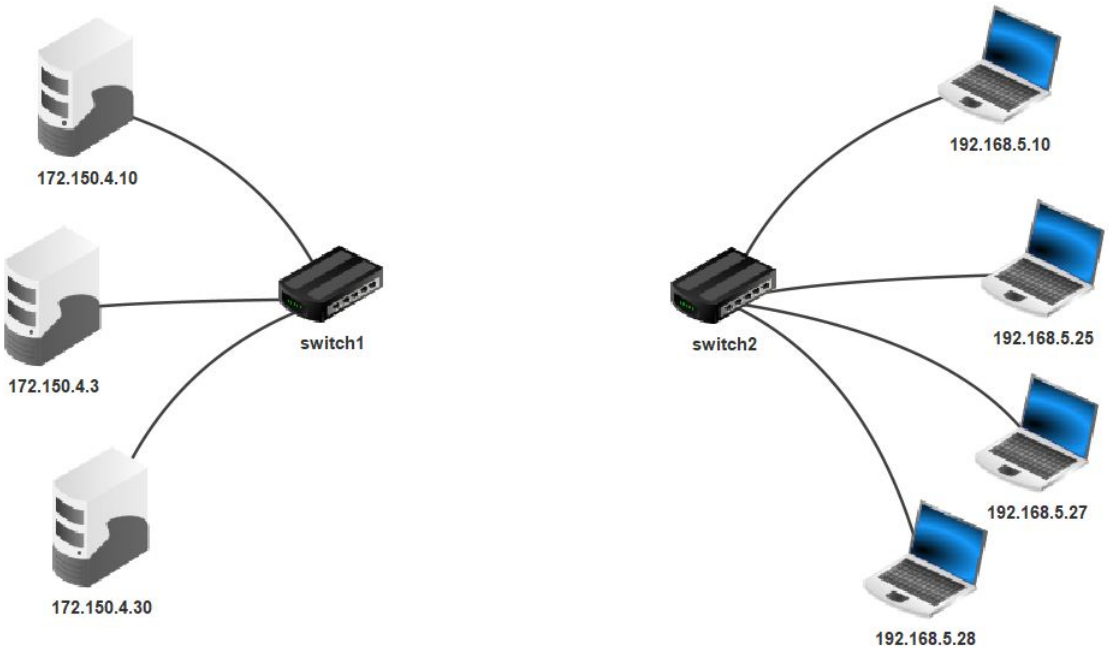
```
def mystere(N,affiche) :
    if N != None :
        if affiche :
            print(N.identite[0])
        mystere(N.gauche,False)
        mystere(N.droite,True)
```

Ecrire, dans l'ordre d'affichage, le résultat de l'exécution de `mystere(racine_de_l_arbre,False)` où `racine_de_l_arbre` est le nœud qui référence Albert Normand.

Exercice 5 (Réseau et protocoles de routage - 4 points)

Pour une « LAN PARTY », les organisateurs gèrent deux réseaux différents non liés physiquement suivant le schéma suivant :

- **Réseau 1 :** réseau contenant le commutateur 1 (switch1) ;
- **Réseau 2 :** réseau contenant le commutateur 2 (switch2).



Dans cet exercice, on exploitera la notation CIDR pour l'adressage des deux réseaux. En notation CIDR, l'adresse IP d'une machine est composée d'une adresse IPv4 et d'une indication sur le masque de sous réseau. Par exemple : `172.16.1.10 / 16` signifie :

- Adresse IP décimale : `172.16.1.10`
- Masque de sous-réseau en notation CIDR : `16`

La notation CIDR `/16` signifie que le masque de sous-réseau a les 16 bits de poids fort de son adresse IP à la valeur 1, c'est-à-dire, pour notre exemple : `11111111.11111111.00000000.00000000`.

Le PC3 du réseau 1 a pour adresse IPv4 `172.150.4.30/24`

1. (a) Combien d'octets sont nécessaires pour constituer une adresse IPv4 ?
   (b) Quelle est la notation décimale du masque de sous-réseau du PC3 du réseau 1 ?
2. Pour déterminer l'adresse IP du réseau, recopier le tableau ci-dessous et compléter les cases vides, en suivant l'ordre des instructions suivantes :
   (a) compléter la ligne 2 : conversion de l'adresse IP décimale en adresse IP binaire sur 4 octets ;
   (b) ligne 3 : compléter le masque de sous réseau en notation binaire ;
   (c) ligne 4 : compléter l'adresse du réseau en notation binaire suite à un `ET` logique entre chaque bit de la ligne 2 et la ligne 3 ;
   (d) ligne 5 : compléter l'adresse IP décimale du réseau 1.

Adresse IP (V4) du PC3	172								150								4								30							
	1	0	1	0	1	1	0	0	1	0	0	1	0	1	1	0																
Masque de S.R.	1	1	1	1	1	1	1	1																								
Pour obtenir l'adresse réseau binaire, on réalise un `ET` logique entre chaque bit de l'adresse IP (ligne 2) et du masque de sous-réseau (ligne 3)																																
Adresse du réseau 1	1	0	1	0	1	1	0	0																								
	172								150																							

3. (a) Parmi les propositions ci-dessous, déterminer, en justifiant, celle(s) qui pourrai(en)t être utilisée(s) pour associer un 4ème PC client au réseau 1 :
   1) `172.154.4.30`

2) `172.150.4.10`
3) `172.150.10.257`
4) `172.150.4.11`
5) `172.150.4.0`
6) `172.150.4.200`

(b) Quelle commande permettrait de connaître son adresse IP ?

Les organisateurs décident de faire une partie de jeu vidéo en connectant entre elles les machines des deux réseaux. Toutes les machines doivent être capables de communiquer entre elles.

4. On décide de connecter directement le switch 1 avec le switch 2 pour réaliser cette nouvelle configuration du réseau. Expliquer pourquoi cette solution n'est pas satisfaisante ? Proposer une alternative ?
5. Dans le cadre d'une future « LAN PARTY », l'organisateur veut gérer la liste des IPv4 pour éviter que deux machines aient la même adresse. Il décide de commencer son étude en créant une fonction Python `adresse`.
   Une liste de listes sera utilisée pour stocker les adresses IP des machines du réseau. Par exemple :

   ```
   liste_IP=[[192,168,10,1],[192,168,10,25],[192,168,10,13]]
   ```

   La fonction `adresse` prend en paramètres l'adresse IP (sous la forme d'une liste) que l'on souhaite tester, une liste de listes (comme `liste_IP`) et :

   - si l'adresse IP testée ne figure pas dans la liste, cette fonction l'ajoute à la liste des adresses IP du réseau et affiche le message `"pas trouvée, ajoutée"` ;
   - si l'adresse IP testée figure dans la liste, cette fonction se contente d'afficher le message `"trouvée"`.

   Exemple :

   ```
   >>> liste_IP=[[192,168,10,1],[192,168,10,25],[192,168,10,13]]
   >>> adresse([192,168,10,3],liste_IP)
   pas trouvée, ajoutée
   >>> liste_IP
   [[192,168,10,1],[192,168,10,25],[192,168,10,13],[192,168,10,3]]
   >>> adresse([192,168,10,25],liste_IP)
   trouvée
   >>> liste_IP
   [[192,168,10,1],[192,168,10,25],[192,168,10,13],[192,168,10,3]]
   ```

   Ecrire en langage Python la fonction `adresse`.

Annexe 1 (exercice 2)

(à ne pas rendre avec la copie)

⋆ **Types de données :**

`CHAR(t)`	Texte fixe de `t` caractères
`VARCHAR(t)`	Texte de `t` caractères variables
`TEXT`	Texte de $65\,535$ caractères maximum
`INT`	Nombre entier de -2^{31} à $2^{31}-1$ (signé) ou de 0 à $2^{32}-1$ (non signé)
`FLOAT`	Réel à virgule flottante
`DATE`	Date format `AAAA-MM-JJ`
`DATETIME`	Date et heure format `AAAA-MM-JJHH:MI:SS`

⋆ **Quelques exemples de syntaxe SQL :**

- Insérer des enregistrements :
```
INSERT INTO Table (attribut1, attribut2) VALUES(valeur1 , valeur2)
```
- Modifier des enregistrements :
```
UPDATE Table SET attribut1=valeur1, attribut2=valeur2 WHERE Selecteur
```
- Supprimer des enregistrements :
```
DELETE FROM Table WHERE Selecteur
```
- Sélectionner des enregistrements :
```
SELECT attributs FROM Table WHERE Selecteur
```
- Effectuer une jointure :
```
SELECT attributs FROM TableA JOIN TableB ON TableA.cle1=TableB.cle2 WHERE Selecteur
```