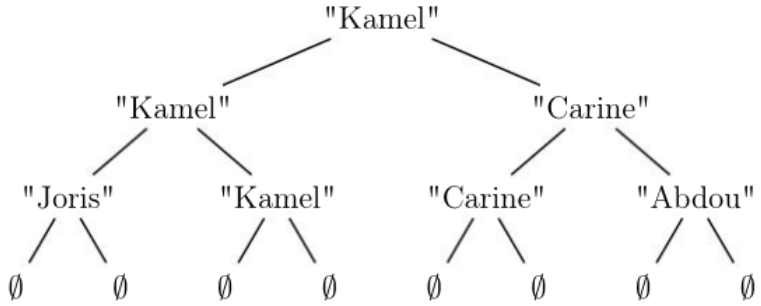
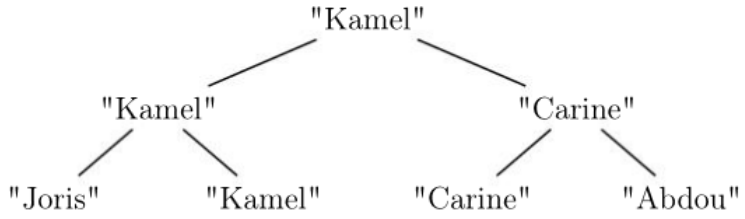
Exercices Bac - Arbres binaires

Exercice 1 :

La fédération de badminton souhaite gérer ses compétitions à l'aide d'un logiciel.
Pour ce faire, une structure **arbre de compétition** a été définie récursivement de la façon suivante : un arbre de compétition est soit l'arbre vide, noté ∅, soit un triplet composé d'une chaîne de caractères appelée valeur, d'un arbre de compétition appelé sous-arbre gauche et d'un arbre de compétition appelé sous-arbre droit.
On représente graphiquement un arbre de compétition de la façon suivante :



Pour alléger la représentation d'un arbre de compétition, on ne notera pas les arbres vides, l'arbre précédent sera donc représenté par l'arbre `A` suivant :



Cet arbre se lit de la façon suivante :

- 4 participants se sont affrontés : Joris, Kamel, Carine et Abdou. Leurs noms apparaissent en bas de l'arbre, ce sont les valeurs de feuilles de l'arbre.
- Au premier tour, Kamel a battu Joris et Carine a battu Abdou.
- En finale, Kamel a battu Carine, il est donc le vainqueur de la compétition.

Pour s'assurer que chaque finaliste ait joué le même nombre de matchs, un arbre de compétition a toutes ces feuilles à la même hauteur.

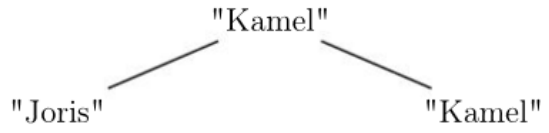
Les quatre fonctions suivantes pourront être utilisées :

- La fonction `racine` qui prend en paramètre un arbre de compétition `arb` et renvoie la valeur de la racine.

  Exemple : en reprenant l'exemple d'arbre de compétition présenté ci-dessus, `racine(A)` vaut `"Kamel"`.

- La fonction `gauche` qui prend en paramètre un arbre de compétition `arb` et renvoie son sous-arbre gauche.

  Exemple : en reprenant l'exemple d'arbre de compétition présenté ci-dessus, `gauche(A)` vaut l'arbre représenté graphiquement ci-après :

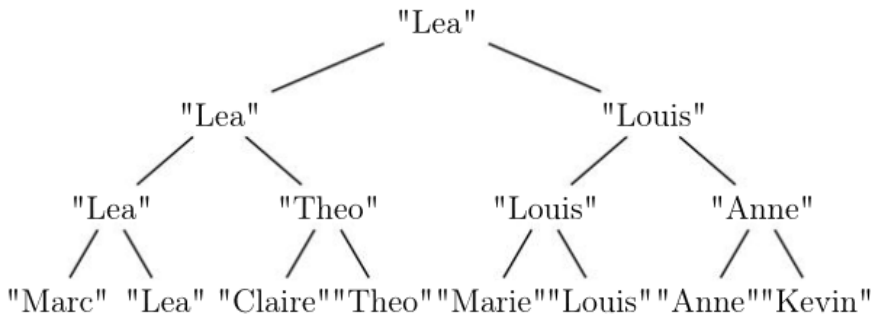


— La fonction `droit` qui prend en argument un arbre de compétition `arb` et renvoie son sous-arbre droit.
Exemple : en reprenant l'exemple d'arbre de compétition présenté ci-dessus, `droit(A)` vaut l'arbre représenté graphiquement ci-dessous :

— La fonction `est_vide` qui prend en argument un arbre de compétition et renvoie `True` si l'arbre est vide et `False` sinon.
Exemple : en reprenant l'exemple d'arbre de compétition présenté ci-dessus, `est_vide(A)` vaut `False`.

Pour toutes les questions de l'exercice, on suppose que tous les joueurs d'une même compétition ont un prénom différent.

1. (a) On considère l'arbre de compétition B suivant :



Indiquer la racine de cet arbre puis donner l'ensemble des valeurs des feuilles de cet arbre.

(b) Proposer une fonction Python `vainqueur` prenant pour argument un arbre de compétition `arb` ayant au moins un joueur. Cette fonction doit renvoyer la chaîne de caractères constituée du nom du vainqueur du tournoi.
Exemple : `vainqueur(B)` vaut `"Lea"`

(c) Proposer une fonction Python `finale` prenant pour argument un arbre de compétition `arb` ayant au moins deux joueurs. Cette fonction doit renvoyer le tableau des deux chaînes de caractères qui sont les deux compétiteurs finalistes.
Exemple : `finale(B)` vaut `["Lea", "Louis"]`

2. (a) Proposer une fonction Python `occurrences` ayant pour paramètre un arbre de compétition `arb` et le nom d'un joueur `nom` et qui renvoie le nombre d'occurrences (d'apparitions) du joueur `nom` dans l'arbre de compétition `arb`.
Exemple : `occurences(B, "Anne")` vaut 2.

(b) Proposer une fonction Python `a_gagne` prenant pour paramètres un arbre de compétition `arb` et le nom d'un joueur `nom` et qui renvoie le booléen `True` si le joueur `nom` a gagné au moins un match dans la compétition représenté par l'arbre de compétition `arb`.
Exemple : `a_gagne(B,"Louis")` vaut `True`

3. On souhaite programmer une fonction Python `nombre_matchs` qui prend pour arguments un arbre de compétition `arb` et le nom d'un joueur `nom` et qui renvoie le nombre de matchs joués par le joueur `nom` dans la compétition représentée par l'arbre de compétition `arb`.

   Exemple : `nombre_matchs(B,"Lea")` doit valoir 3 et `nombre_matchs(B,"Marc")` doit valoir 1.

   (a) Expliquer pourquoi les instructions suivantes renvoient une valeur erronée. On pourra pour cela identifier le noeud de l'arbre qui provoque une erreur.

```
1 def nombre_matchs(arb,nom):
2     """arbre_competition , str -> int"""
3     return occurrences(arb,nom)
```

   (b) proposer une correction pour la fonction `nombre_matchs`.

4. Recopier et compléter la fonction `liste_joueurs` qui prend pour argument un arbre de compétition `arb` et qui renvoie un tableau contenant les participants au tournoi, chaque nom ne devant figurer qu'une seule fois dans le tableau.

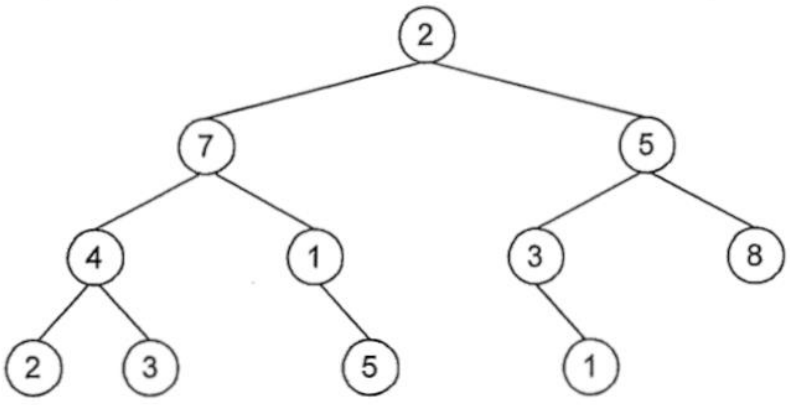
   L'opération `+` à la ligne 8 permet de concaténer deux tableaux.

   Exemple : Si `L1 = [4, 6, 2]` et `L2 = [3, 5, 1]`, l'instruction `L1 + L2` va renvoyer le tableau `[4, 6, 2, 3, 5, 1]`

```
1 def liste_joueurs(arb):
2     """arbre_competition -> tableau"""
3     if est_vide(arb):
4         return ...
5     elif ... and ... :
6         return [racine(arb)]
7     else :
8         return ...+liste_joueurs(droit(arb))
```

Exercice 2 :

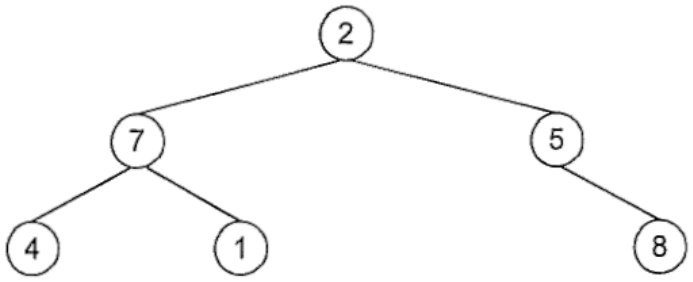
1. Déterminer la plus grande somme racine-feuille de l'arbre représenté ci-dessous.



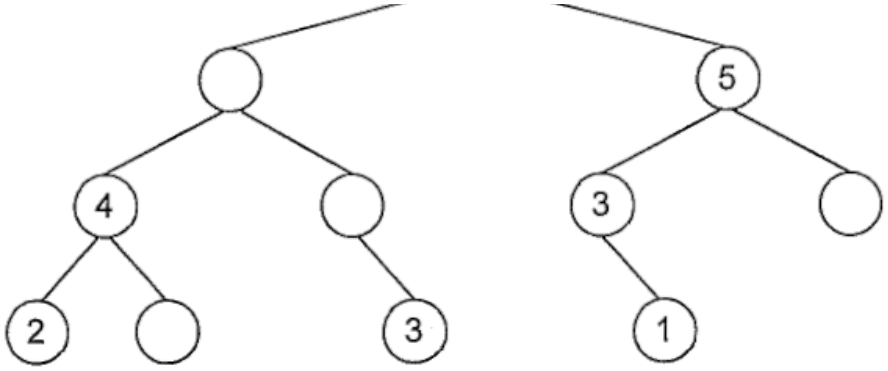
2. La classe `Noeud` ci-dessous implémente le type abstrait d'arbre binaire.

```
class Noeud:

    def __init__(self, v):
        self.etiquette = v
        self.sag = None
        self.sad = None

    def niveau(self):
        if self.sag!=None and self.sad!=None:
            hg = self.sag.niveau()
            hd = self.sad.niveau()
            return 1+max(hg, hd)
        if self.sag!=None:
            return self.sag.niveau()+1
        if self.sad!=None:
            return self.sad.niveau()+1
        return 0

    def modifier_sag(self, nsag) :
        self.sag = nsag

    def modifier_sad(self, nsad) :
        self.sad = nsad
```

a. Écrire une suite d'instructions utilisant la classe `Noeud` permettant de représenter l'arbre ci-dessous.



b. Que renvoie l'appel de la méthode `niveau` sur l'arbre ci-dessus ?



b. Un arbre est magique si ses sous-arbres sont magiques et qu'ils ont de plus la même plus grande somme racine-feuille. Écrire une méthode récursive `est_magique` qui renvoie `True` si l'arbre est magique et `False` sinon.