

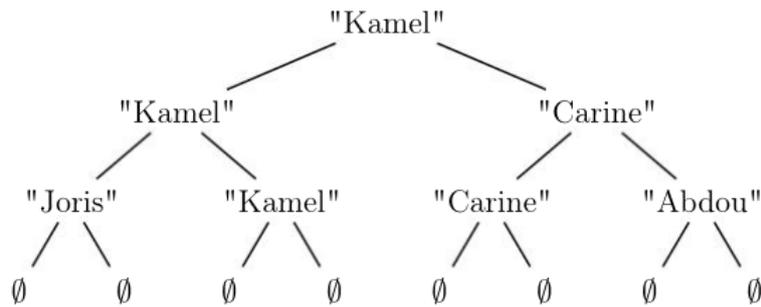
# Exercices Bac - Arbres binaires

## Exercice 1 :

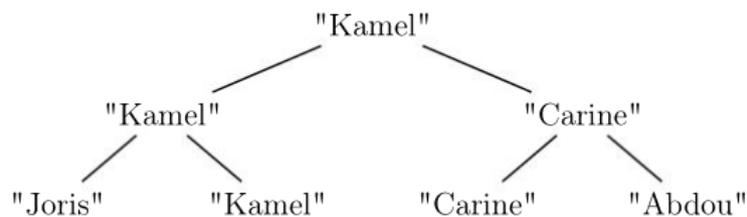
La fédération de badminton souhaite gérer ses compétitions à l'aide d'un logiciel.

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  $\emptyset$ , 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.

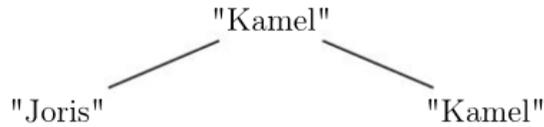
Les quatre fonctions suivantes pourront être utilisées :

- 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".

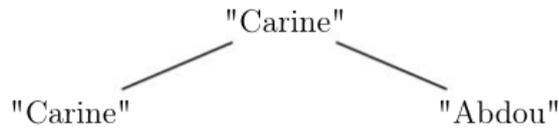
- La fonction **gauche** qui prend en paramètre un arbre de compétition **arb** et renvoie son sous-arbre gauche.

**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 :

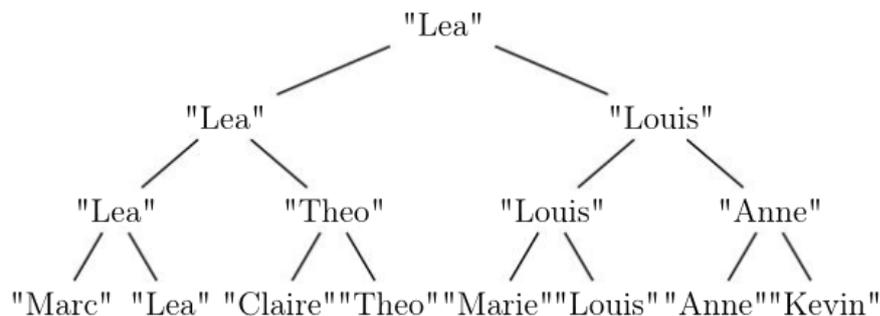


- La fonction `est_vide` qui prend en argument un arbre de compétition et renvoie `True` si l'arbre est vide et `False` sinon.

**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.

- (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.

La racine de cet arbre est "Léa" : `racine(B) = "Léa"`

- (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"`

```
def vainqueur(arb) :
    return racine(arb)
```

- (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"]`

```
def finale(arb) :
    f1 = gauche(arb)
    f2 = droite(arb)
    return [racine(f1) , racine(f2)]
```



Script itératif (utilise un file f)

```
def occurences(arb,nom) :
    f = [arb]
    n = 0
    while len(f) > 0 :
        nd = f.pop(0)
        if racine(nd) == nom : n = n + 1
        if not est_vide(gauche(nd)) : f.append(gauche(nd))
        if not est_vide(droite(nd)) : f.append(droite(nd))
    return n
```

- (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`

```
def a_gagne(arb,nom) :
    n = occurences(arb,nom)
    return n > 1
```

3. On souhaite programmer une fonction Python `nombre_matches` 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_matches(B,"Lea")` doit valoir 3 et `nombre_matches(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_matches(arb,nom):
2     """arbre_competition , str -> int"""
3     return occurences(arb,nom)
```

La valeur est erronée pour le vainqueur du tournoi. Son nom inscrit sur la racine ne correspond à un nouveau match.

- (b) proposer une correction pour la fonction `nombre_matches`.

```
def nombre_matches(arb,nom) :
    if vainqueur(arb) == nom :
        return occurences(arb,nom) - 1
    else :
        return occurences(arb,nom)
```

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]`

```

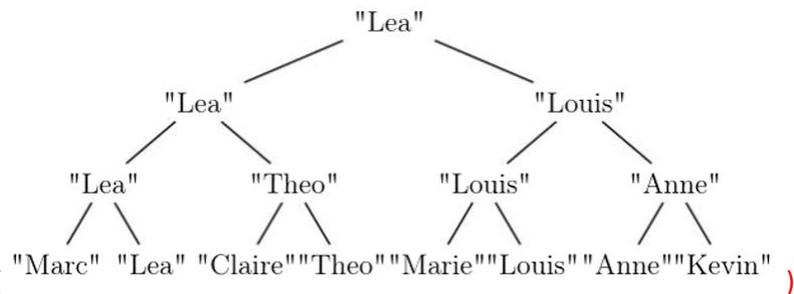
1 def liste_joueurs(arb):
2     """arbre_competition -> tableau"""
3     if est_vide(arb):
4         return ...
5     elif ... and ... :
6         return [racine(arb)]
7     else :
8         return ...+liste_joueurs(droit(arb))

```

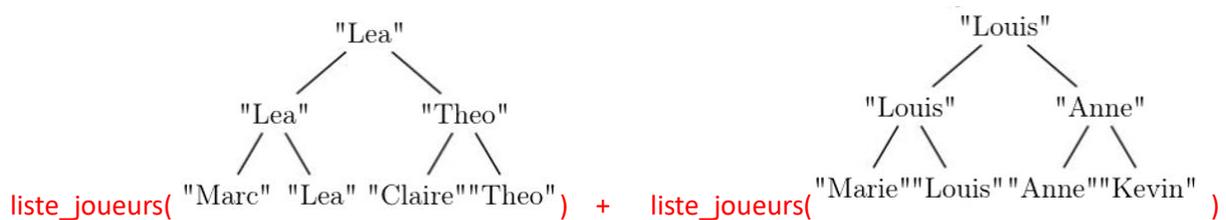
```

def liste_joueurs(arb) :
    if est_vide(arb) :
        return []
    elif est_vide(gauche(arb)) and est_vide(droite(arb)) :
        return [racine(arb)]
    else :
        return liste_joueurs(gauche(arb)) + liste_joueurs(droite(arb))

```



Pour l'arbre B, exécution de `liste_joueurs(` "Marc" "Lea" "Claire" "Theo" "Marie" "Louis" "Anne" "Kevin" )



`liste_joueurs( "Marc" "Lea" ) + liste_joueurs( "Claire" "Theo" ) + liste_joueurs( "Marie" "Louis" ) + liste_joueurs( "Anne" "Kevin" )`

`list_j( "Marc" ) + list_j( "Lea" ) + list_j( "Claire" ) + list_j( "Theo" ) + list_j( "Marie" ) + list_j( "Louis" ) + list_j( "Anne" ) + list_j( "Kevin" )`

`[ "Marc" ] + [ "Lea" ] + [ "Claire" ] + [ "Theo" ] + [ "Marie" ] + [ "Louis" ] + [ "Anne" ] + [ "Kevin" ]`