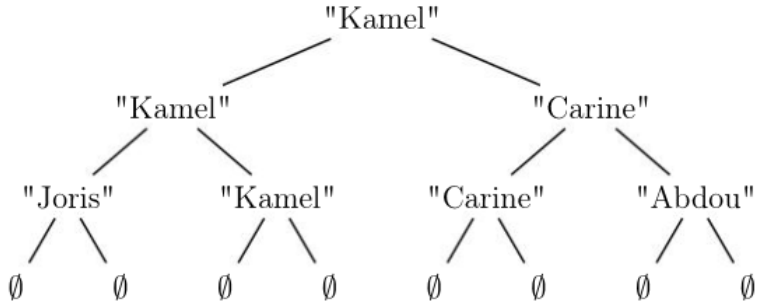
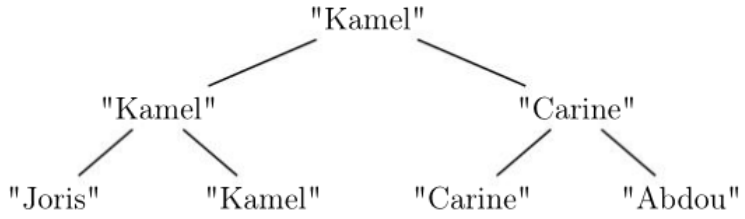
Exercices Bac - Arbres binaires

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.

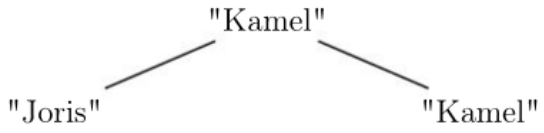
Les quatre fonctions suivantes pourront être utilisées :

- 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 :

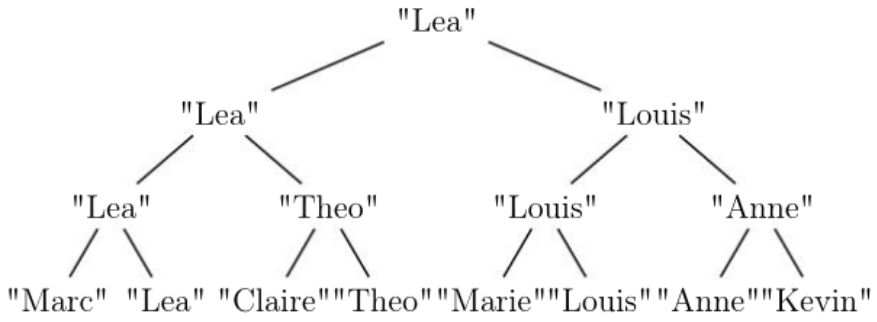


— La fonction `droit` qui prend en argument un arbre de compétition `arb` et renvoie son sous-arbre droit.
Exemple : en reprenant l'exemple d'arbre de compétition présenté ci-dessus, `droit(A)` vaut l'arbre représenté graphiquement ci-dessous :

— 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 = ["Lea",
        [  "Lea",
            [  "Lea",
               ["Marc",None,None],
               ["Lea",None,None]
            ],
            [  "Theo",
               ["Claire",None,None
               ["Theo",None,None]
            ]
        ],
        [  "Louis",
            ["Louis",
               ["Marie",None,None]
               ["Louis",None,None]
            ],
            ["Anne",
               ["Anne",None,None],
               ["Kevin",None,None]
            ]
        ]
    ]
```

```
def racine(arb) :
    return arb[0]

def gauche(arb) :
    return arb[1]

def droite(arb) :
    return arb[2]

def est_vide(arb) :
    return arb == None
```

```
assert racine(B) == "Lea" , "la fct racine() ne donne pas le bon résultat"
l =["Lea",["Lea",["Marc",None,None],["Lea",None,None]],["Theo",["Claire",None,None],["Theo",None,None]]]
assert gauche(B) == l , "la fct gauche() ne donne pas le bon résultat"
assert est_vide(None)  , "la fct est_vide() ne donne pas le bon résultat"
```

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)
```

```
assert vainqueur(B) == "Lea" , "la fct vainqueur() ne donne pas le bon résultat"
```

(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)]
```

```
assert finale(B) == ["Lea","Louis"] , "la fct finale() ne donne pas le bon résultat"
```

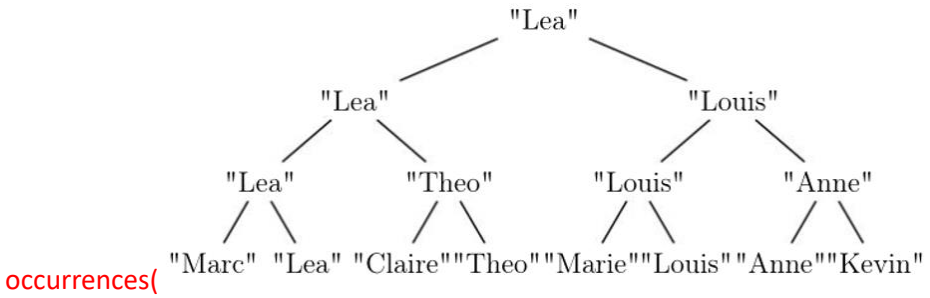
2. (a) Proposer une fonction Python `occurrences` ayant pour paramètre un arbre de compétition `arb` et le nom d'un joueur `nom` et qui renvoie le nombre d'occurrences (d'apparitions) du joueur `nom` dans l'arbre de compétition `arb`.

Exemple : `occurences(B, "Anne")` vaut 2.

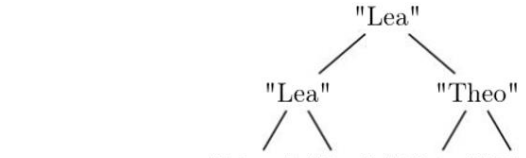
Script récursif : détail de l'exécution occurrences(B,''Anne'') :

```
def occurrences(arb,nom) :
    if est_vide(arb) : return 0
    n = 0
    if racine(arb) == nom : n = 1
    return n + occurrences(gauche(arb),nom) + occurrences(droite(arb),nom)
```

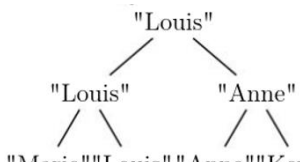
```
assert occurences(B,"Anne") == 2 , "la fct finale() ne donne pas le bon résultat"
```



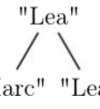
occurrences(,''Anne'')



0 + occurrences(,''Anne'') + occurrences(



,''Anne'')



0 + occ(,''Anne'') + occ (

,''Anne'') + 0 + occ (

,''Anne'') + occ (

,''Anne'')

0 + occ(,''A'') + occ (,''A'') + 0 + occ (,''A'') + occ (,''Anne'') + 0 + occ(,''A'') + occ (,''A'') + 1 + occ (,''A'') + occ (,''A'')

0 + occ(Ø,''A'') + occ (Ø,''A'') + 0 + occ (Ø,''A'') + occ (Ø,''Anne'') + 0 + occ(Ø,''A'') + occ (Ø,''A'') + 0+ occ (Ø,''A'') + occ (Ø,''A'') + 0 + occ(Ø,''A'') + occ (Ø,''A'') + 0 + occ (Ø,''A'') + occ (Ø,''Anne'') +1+ 1 + occ(Ø,''A'') + occ (Ø,''A'') + 0+ occ (Ø,''A'') + occ (Ø,''A'')

0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 1 + 1 + 0 + 0 + 0 + 0 = 2

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
```

```
assert a_gagne(B,"Louis")  , "la fct a_gagne() ne donne pas le bon résultat"
```

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)
```

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_matchs`.

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

```
assert nombre_matchs(B,"Lea") == 3  , "la fct nombre_matchs() ne donne pas le bon résultat"
assert nombre_matchs(B,"Marc") == 1  , "la fct nombre_matchs() ne donne pas le bon résultat"
```

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.

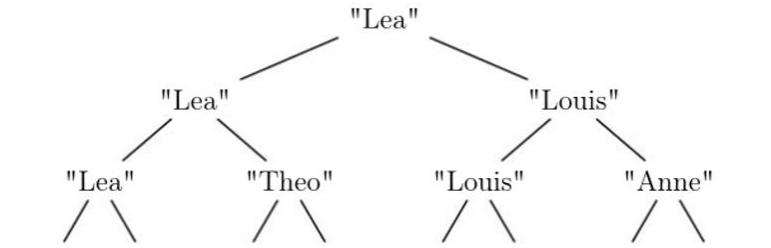
   L'opération `+` à la ligne 8 permet de concaténer deux tableaux.

   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]`

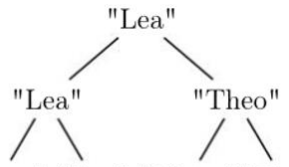
```
1 def liste_joueurs(arb):
2     """arbre_competition -> tableau"""
3     if est_vide(arb):
4         return ...
5     elif ... and ... :
6         return [racine(arb)]
7     else :
8         return ...+liste_joueurs(droit(arb))
```

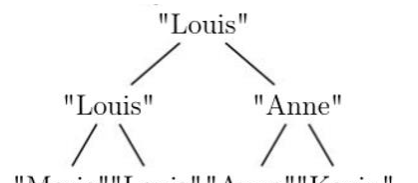
```
def liste_joueurs(arb) :
    if est_vide(arb) :
        return []
    elif est_vide(gauche(arb)) and est_vide(droite(arb)) :
        return [racine(arb)]
    else :
        return liste_joueurs(gauche(arb)) + liste_joueurs(droite(arb))
```

```
l = ['Marc', 'Lea', 'Claire', 'Theo', 'Marie', 'Louis', 'Anne', 'Kevin']
assert liste_joueurs(B) == l  , "la fct liste_joueurs() ne donne pas le bon résultat"
```

Pour l'arbre B, exécution de liste_joueurs(

)

liste_joueurs(

) + liste_joueurs(

)

liste_joueurs(

) + liste_joueurs(

)+ liste_joueurs(

)+liste_joueurs(

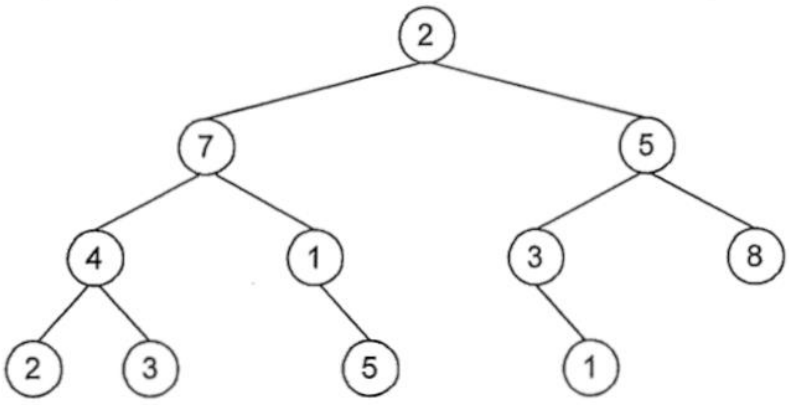
)

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"]

Exercice 2 :

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

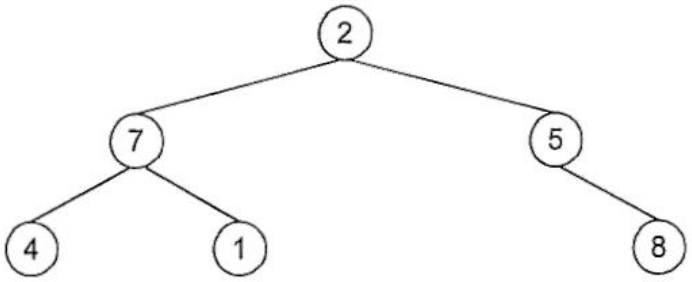


La plus grande somme est 2 + 7 + 4 + 3 = 16

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.




```
s2 = Noeud(2)
s7 = Noeud(7)
s5 = Noeud(5)
s2.modifier_sag(s7)
s2.modifier_sad(s5)
s4 = Noeud(4)
s1 = Noeud(1)
s7.modifier_sag(s4)
s7.modifier_sad(s1)
s8 = Noeud(8)
s5.modifier_sad(s8)
```

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

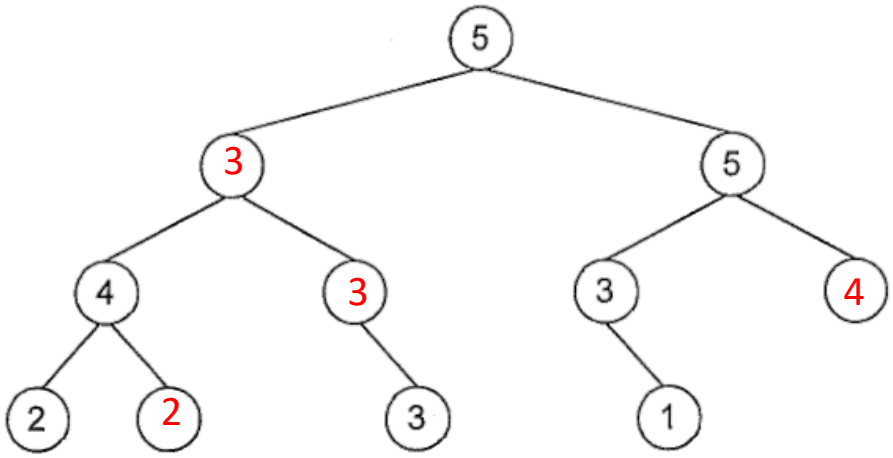
```
>>> A.niveau()
2
```

3. S'inspirer du code de la méthode `niveau` pour écrire une méthode récursive `pgde_somme` qui renvoie la plus grande somme racine-feuille d'un arbre.

```
def pgde_somme(self) :
    if self.sag != None and self.sad != None :
        g = self.sag.pgde_somme()
        d = self.sad.pgde_somme()
        return self.etiquette + max(g , d)
    if self.sag != None :
        return self.etiquette + self.sag.pgde_somme()
    if self.sad != None :
        return self.etiquette + self.sad.pgde_somme()
    return self.etiquette
```

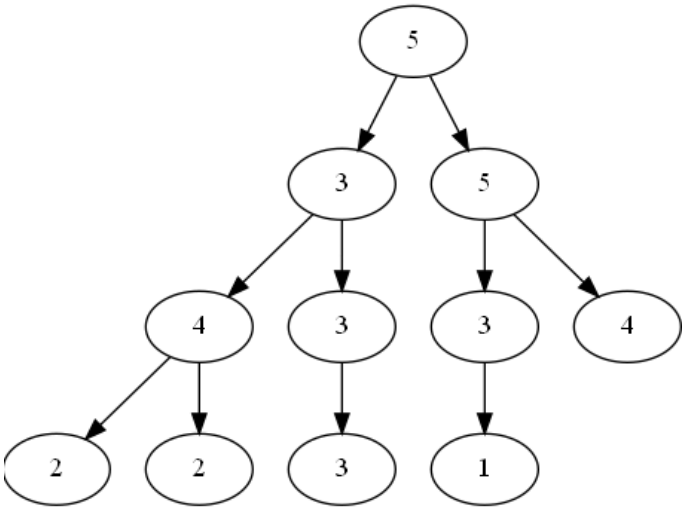
4. On appelle arbre magique un arbre binaire dont toutes les sommes des chemins racine-feuille sont égales.

a. Recopier et compléter l'arbre ci-dessous pour qu'il soit magique.



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.

```python
def est_magique(self) :
    if self.sag != None and self.sad != None :
        g = self.sag.pgde_somme()
        d = self.sad.pgde_somme()
        return g == d and self.sag.est_magique() and self.sad.est_magique()
    if self.sag != None :
        return self.sag.est_magique()
    if self.sad != None :
        return self.sad.est_magique()
    return True
```




```python
A = Noeud(5)
A.modifier_sag(Noeud(3))
A.modifier_sad(Noeud(5))
A.sag.sag = Noeud(4)
A.sag.sad = Noeud(3)
A.sad.sag = Noeud(3)
A.sad.sad = Noeud(4)
A.sag.sag.sag = Noeud(2)
A.sag.sag.sad = Noeud(2)
A.sag.sad.sad = Noeud(3)
A.sad.sag.sad = Noeud(1)
```

```
>>> A.est_magique()
True
```