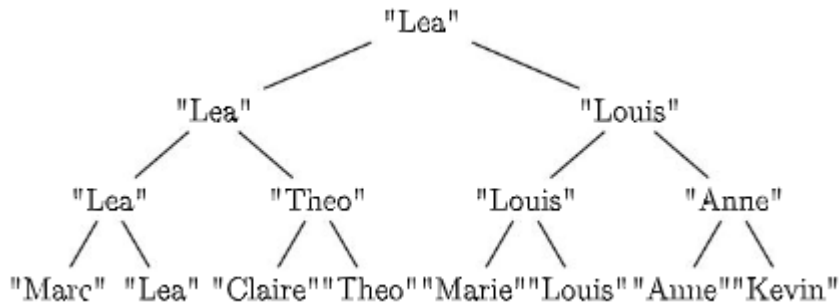


2021 - J1 - Amérique du Nord

Exercice 4



1.(a) Indiquer la racine de cet arbre.

La racine de cet arbre vaut "Lea".

1.(a) (suite) Donner l'ensemble des valeurs des feuilles de cet arbre.

Il y a 8 feuilles : "Marc", "Lea", "Claire", "Theo", "Marie", "Louis", "Anne", "Kevin".

1.(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"

```
[1]: def vainqueur(arb):  
      return racine(arb)
```

1.(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"]

```
[3]: def finale(arb):  
      arbre_gauche = gauche(arb)  
      finaliste_gauche = vainqueur(arbre_gauche)  
  
      arbre_droit = droit(arb)
```



```
finaliste_droit = vainqueur(arbre_droit)

return [finaliste_gauche, finaliste_droit]
```

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 : `occurrences(B, "Anne")` vaut 2.

On va utiliser une fonction récursive. C'est plus simple!

Le cas de base d'un arbre, c'est l'arbre vide. Et là, l'occurrence est toujours égale à 0

Sinon, le nombre d'occurrences est égal à la somme du nombre d'occurrences du sous arbre gauche, du nombre d'occurrence du sous-arbre droit. On ajoute 1 si la racine est égale au nom recherché.

```
[4]: def occurrences(arb, nom):
    # cas de base : l'arbre vide
    if est_vide(arb):
        return 0

    # cas récursif
    nb_occurrences = 0

    nb_occ_gauche = occurrences(gauche(arb), nom)
    nb_occurrences += nb_occ_gauche

    nb_occ_droite = occurrences(droit(arb), nom)
    nb_occurrences += nb_occ_droite

    if racine(arb) == nom:
        nb_occurrences += 1

    return nb_occurrences
```

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`

Il suffit d'utiliser la fonction précédente `occurrences` qui nous permet de répondre facilement à cette question.



Une personne a au moins une victoire si son nom apparaît au moins 2 fois dans l'arbre!

```
[5]: def a_gagne(arb, nom):  
      return occurrences(arb, nom) > 1
```

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

```
[11]: def nombre_matches(arb, nom):  
      """ arbre_competition , str -> int """  
      return occurrences (arb , nom)
```

Cette fonction est correcte pour tout le monde, sauf pour le vainqueur du tournoi. En effet, le vainqueur a son nom qui apparaît une fois de plus, à la racine de l'arbre.

3.(b) Proposer une correction pour la fonction `nombre_matches`.

```
[10]: def nombre_matches(arb, nom):  
      """ arbre_competition , str -> int """  
      nb_occurences = occurrences(arb, nom)  
  
      if nom != vainqueur(arb):  
          return nb_occurences  
      else:  
          return nb_occurences - 1
```

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 $[4, 6, 2, 3, 5, 1]$.

```
[9]: def liste_joueurs(arb):  
      """ arbre_competition -> tableau """  
      if est_vide(arb):  
          return ...  
      elif ... and ... :  
          return [racine(arb)]  
      else :  
          return ...+ liste_joueurs(droit(arb))
```

```
[12]: def liste_joueurs(arb):  
      """ arbre_competition -> tableau """  
      # cas de base : arbre vide  
      if est_vide(arb):
```



```
# aucun nom à renvoyer
return []

# cas de la feuille
elif est_vide(gauche(arb)) and est_vide(droit(arb)) :
    return [racine(arb)]

# cas d'un nœud : il faut descendre plus bas dans l'arbre
else :
    return liste_joueurs(droit(arb)) + liste_joueurs(droit(arb))
```