Contrôle de connaissances

Nom :..... **Prénom :**.....

On travaille avec la classe suivante :

Implémentation d'un arbre binaire :

```
class Arbre:
    def __init__(self,racine,gauche=None,droit=None):
  # Par defaut l'arbre est une feuille
        self.racine=racine # Valeur de la racine
        self.gauche=gauche # Pointeur vers le sous-arbre gauche
        self.droit=droit # Pointeur vers le sous-arbre droit
```

Dans cette définition, un arbre ne peut être vide. Il y a forcément au moins un élément, la racine. Si c'est le seul, l'arbre est alors une feuille.
Pour facilité les méthodes, nous ajoutons les contraintes suivantes :

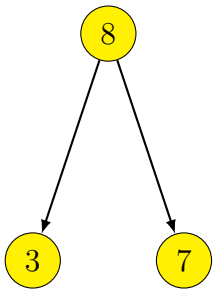
- tout arbre possède 0 ou deux fils.
- Dans un arbre, toutes les étiquettes sont uniques.

.

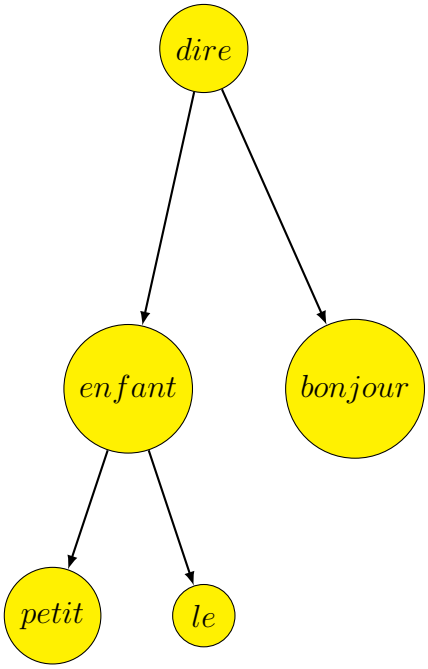
Partie I

1. Représenter les arbres `arb_1`, `arb_2` et `arb_3` suivants :

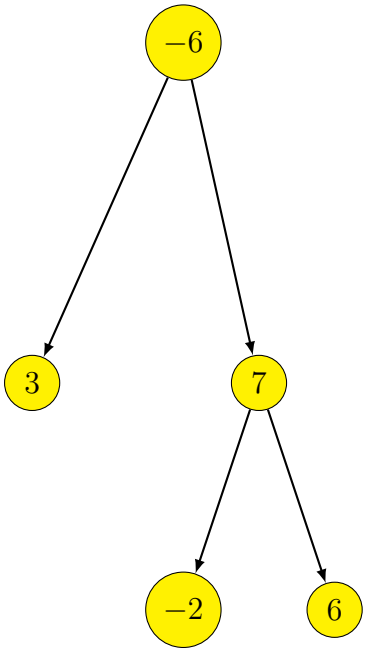
   - `arb_1 = Arbre(8,Arbre(3) , Arbre(7))`



   - `arb_2 = Arbre("dire" , Arbre("enfant" , Arbre("petit") ,\`
     `Arbre("le")), Arbre("bonjour"))`


dire
enfant
bonjour
petit
le

- ```
  arb_3 = Arbre(-6 , Arbre(3) , \
      Arbre(7 , Arbre(-2) , Arbre(6)))
  ```



2. Écrire la méthode `est_une_feuille` qui renvoie le booléen correspondant à l'arbre est une feuille.

3. Écrire la méthode `taille` qui renvoie la taille de l'arbre.
   Exemple :

   ```
   >>> arb_1.taille()
   3
   >>> arb_2.taille()
   5
   ```
   ```

4. Écrire la méthode `recherche` qui prend en argument un élément et retourne le booléen correspondant à l'appartenance de l'élément dans l'arbre.
Exemple :

```
>>> arb_1.recherche(7)
True
>>> arb_3.recherche(5)
False
```

5. Écrire la méthode `element_le_plus_a_gauche` qui retourne l'étiquette du nœud le plus à gauche dans l'arbre. **Exemple :**

```
>>> arb_2.element_le_plus_a_gauche()
'petit'
>>> arb_3.element_le_plus_a_gauche()
3
```

Correction

```
class Arbre:
    def __init__(self,racine,gauche=None,droit=None):
  # Par defaut l arbre est une feuille
        self.racine=racine # Valeur du noeud
        self.gauche=gauche # Pointeur vers le sous-arbre gauche
        self.droit=droit # Pointeur vers le sous-arbre droit

    def est_une_feuille(self) :
        return self.gauche==None and self.droit==None

    def taille(self) :
        if self.est_une_feuille() :
            return 1
        else :
            return 1 + self.gauche.taille() + self.droit.taille()

    def recherche(self,element) :
        if self.est_une_feuille() :
            return self.racine == element
        else :
            return self.gauche.recherche(element)\
                   or self.droit.recherche(element)

    def element_le_plus_a_gauche(self) :
        if self.est_une_feuille() :
            return self.racine
        else :
            return self.gauche.element_le_plus_a_gauche()


arb_1 = Arbre(8,Arbre(3) , Arbre(7))

arb_2 = Arbre("dire" , Arbre("enfant" , Arbre("petit") , Arbre("le")),\
```

```
                Arbre("bonjour"))

arb_3 = Arbre(-6 , Arbre(3) , \
  Arbre(7 , Arbre(-2) , Arbre(6)))

exemple_1 = Arbre("ou", Arbre("et" , Arbre(True),Arbre(False)),\
                  Arbre("ou" , Arbre("et",Arbre(False),Arbre(True)),\
                        Arbre("ou",Arbre(False),Arbre(True))))

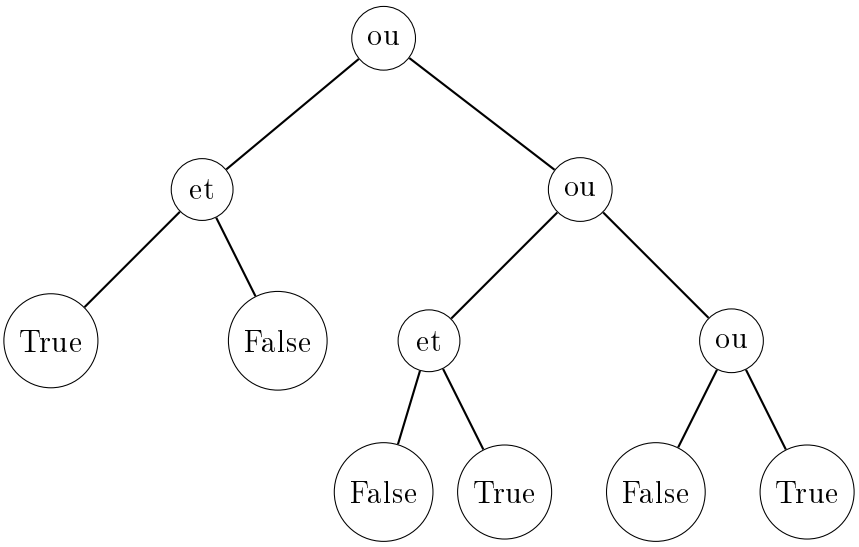
def simplification ( arb) :
    if arb.est_une_feuille() :
        return arb
    else :
        r = arb.racine
       # On simplifie les deux fils
        simp_arb_gauche = simplification(arb.gauche)
        simp_arb_droit = simplification(arb.droit)
        if r == "ou" :
            return Arbre(simp_arb_gauche.racine or \
                         simp_arb_droit.racine)
        if r =="et" :
            return Arbre(simp_arb_gauche.racine and \
                         simp_arb_droit.racine)

print(simplification(exemple_1).racine)
```

Partie II

Le but de cette partie est d'étudier les arbres booléens.

Exemple 1 :



1. Définir l'exemple 1.

Correction

```
exemple_1 = Arbre("ou", Arbre("et" , Arbre(True),Arbre(False)),\
                  Arbre("ou" , Arbre("et",Arbre(False),Arbre(True)),\
                        Arbre("ou",Arbre(False),Arbre(True))))
```

2. Donner le booléen associé à l'arbre exemple.

Correction

On procède en partant des feuilles, on trouve True.

3. Pour simplifier un arbre booléen, nous allons procéder à sa simplification de la manière suivante :

   - Si l'arbre est une feuille, elle contient comme étiquette une constante booléenne (`True` ou `Talse`), il est donc simplifié.
   - Si l'arbre est un nœud où les deux fils sont des feuilles, la simplification renvoie la feuille correspondante au résultat de l'opération booléenne.
   - Sinon, on procède à la simplification des deux arbres fils, et on renvoie la simplification.

   Compléter la fonction `simplification` qui renvoie la feuille équivalente à l'arbre donné en argument.

```
def simplification ( arb) :
    if arb.est_une_feuille() :
        return ....................................
    else :
        r = arb.racine
       # On simplifie les deux fils
        simp_arb_gauche = simplification(.............................

        simp_arb_droit = simplification(.............................

        if r == "ou" :
            return Arbre(.....................................

        if r =="et" :
            return Arbre(.....................................
```

Correction

```
def simplification ( arb) :
    if arb.est_une_feuille() :
        return arb
    else :
        r = arb.racine
       # On simplifie les deux fils
        simp_arb_gauche = simplification(arb.gauche)
        simp_arb_droit = simplification(arb.droit)
        if r == "ou" :
            return Arbre(simp_arb_gauche.racine or \
                         simp_arb_droit.racine)
        if r =="et" :
            return Arbre(simp_arb_gauche.racine and \
                         simp_arb_droit.racine)
```