T NSI	Structures de données	*TP_POO : Les arbres*

Objectif : Implémenter la classe arbre binaire

I. Classe Nœud :

Créer une classe Noeud qui aura 3 propriétés (ou attributs) :
La propriété valeur contiendra la valeur associée au noeud.
Les propriétés gauche et droit, **qui sont des instances de la classe Nœud !!!**
Si il n'y a pas de sous arbre gauche ou droit, on indiquera la valeur None dans les propriétés correspondantes.

```
class Noeud():
    """Représente un noeud dans un arbre binaire"""

    def __init__(self,valeur):
        """ Constructeur """
        self.valeur = valeur
        self.gauche = None
        self.droit = None

    def tableau(self):
        """Renvoie un tableau qui représente le noeud"""
        if self.gauche is None:
            fils_gauche = []
        else:
            fils_gauche = self.gauche.tableau()

        if self.droit is None:
            fils_droite = []
        else:
            fils_droite = self.droit.tableau()

        return [self.valeur, fils_gauche, fils_droite]

    def __repr__(self):
        return str(self.tableau())
```

Q1. Implémenter la méthode ajouter_gauche(self,valeur_gauche) , qui ajoute un noeud fils à gauche.

Q2. Implémenter la méthode ajouter_droite(self,valeur_droite) , qui ajoute un noeud fils à droite.

Q3. Implémenter la méthode est feuille(self) , qui renvoie True si le nœud est une feuille .

Q4. Créer une instance arbre de la classe Nœud qui représente l'arbre de la figure 1

Q5. Implémenter la méthode hauteur(self) qui renvoie la hauteur de l'arbre.

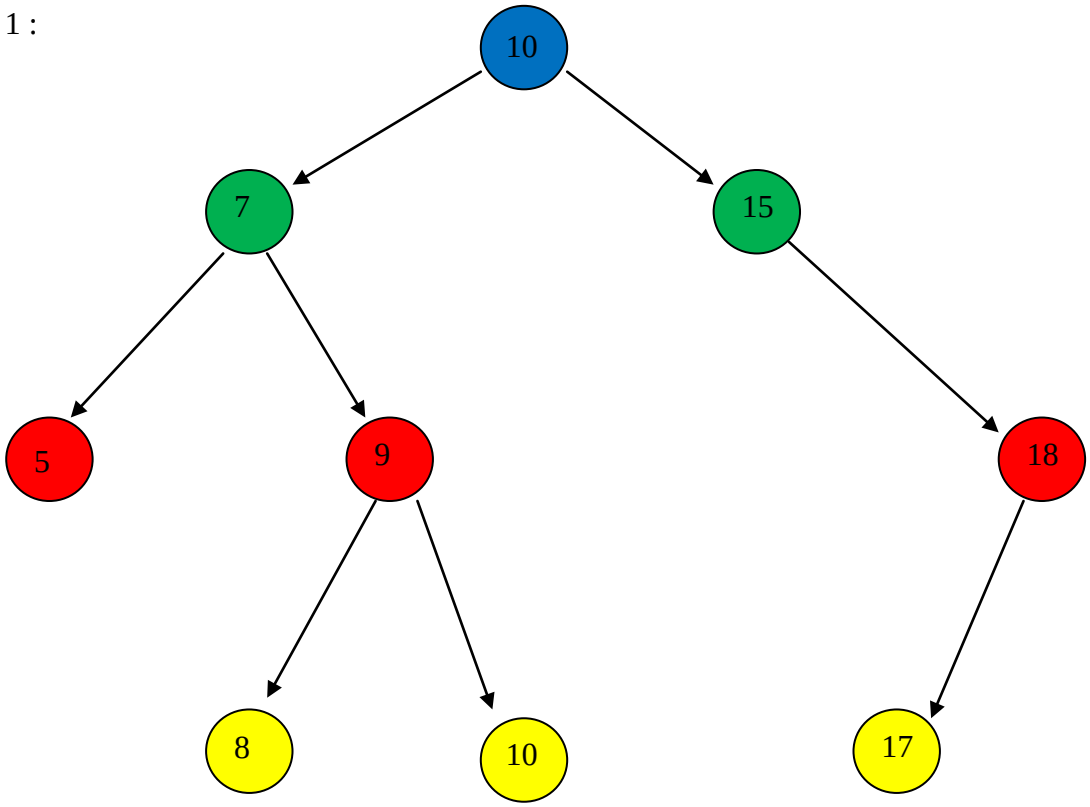
Q6. Implémenter la méthode taille(self) qui renvoie la taille de l'arbre.

Q7. Implémenter la méthode degré(self) qui renvoie le degré d'un Nœud.

Q8. Implémenter la méthode maximum(self) qui renvoie le maximum de l'arbre.

Q9. Implémenter la méthode minimum(self) qui renvoie le minimum

figure 1 :



II. Bonus : Affichage d'un Arbrebinaire

```
from graphviz import Digraph

class Arbrebin:
    """Représente un objet arbre binaire    """

    def __init__(self, nd = None):
        # Initialise l'arbre à vide par défaut, sinon avec un noeud passé en paramètre otionnel
        self.racine = nd

    def importe(self, tableau):
        """Importe un arbre depuis un tableau"""
        def importe_tableau(tableau):
            # Cas particuliers
            if tableau == []:
                return None
            if len(tableau) == 1:
                return Noeud(tableau[0])

            # tableau a une longueur >= 2
            nd = Noeud(tableau[0])
            nd.gauche = importe_tableau(tableau[1])
            nd.droit  = importe_tableau(tableau[2]) if len(tableau) > 2 else None
            return nd

        self.racine = importe_tableau(tableau)
```

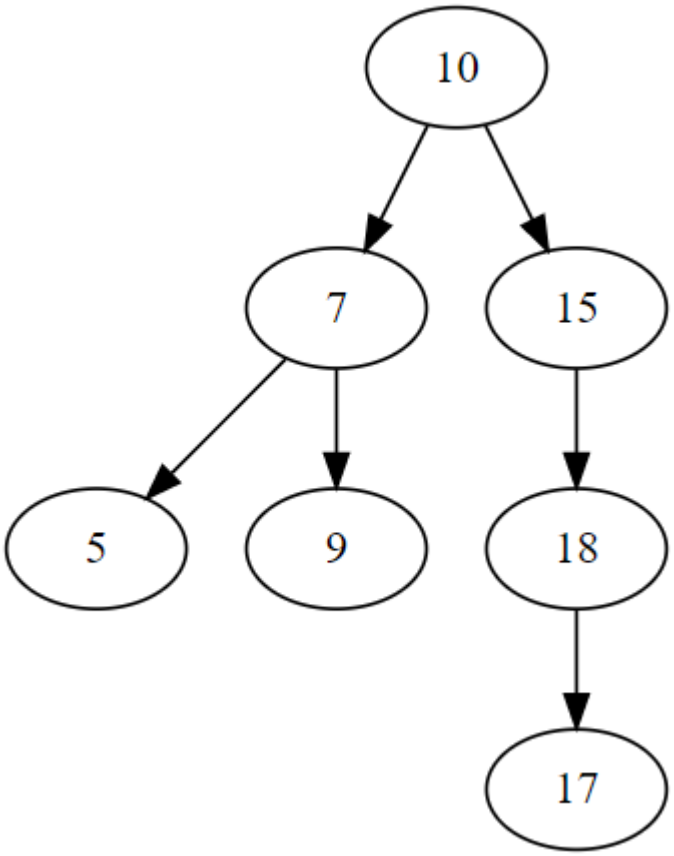
```
    def show(self):
        """Renvoie un objet graphviz pour la visualisation graphique de l'arbre"""
        def representation(dot, noeud, aretes):
            # Ajoute la représentation du noeud à la représentation dot de l'arbre
            if noeud is not None:
                dot.node(str(id(noeud)), str(noeud.valeur))
                # Appel récursif de la fonction representation
                if noeud.gauche is not None:
                    representation(dot, noeud.gauche, aretes)
                    aretes.append((str(id(noeud)) , str(id(noeud.gauche))))
                if noeud.droit is not None:
                    representation(dot, noeud.droit, aretes)
                    aretes.append((str(id(noeud)) , str(id(noeud.droit))))

        dot = Digraph(comment="Arbre binaire", format='svg')
        aretes = []
        representation(dot, self.racine, aretes)
        dot.edges(aretes)
        return dot

tableau=arbre.tableau()

# On crée une instance vide de notre arbre
arbre = Arbrebin()
# On importe le tableau ci-dessus
arbre.importe(tableau)

# On visualise l'arbre graphiquement
arbre.show()
```



Correction :

```
    def ajouter_gauche(self,valeur_gauche):
        """Ajoute un noeud fils à gauche"""
        self.gauche=Noeud(valeur_gauche)

    def ajouter_droite(self,valeur_droite):
        """Ajoute un noeud fils à droite"""
        self.droit=Noeud(valeur_droite)

    def est_feuille(self):
        """ Renvoie True si gauche = None et droit = None"""
        return self.gauche is None and self.droit is None



arbre=Noeud(10)
arbre.ajouter_gauche(7)
arbre.ajouter_droite(15)
arbre.gauche.ajouter_gauche(5)
arbre.gauche.ajouter_droite(9)
arbre.droit.ajouter_droite(18)
arbre.droit.droit.ajouter_gauche(17)
print(arbre)


    def hauteur(self):
        """Renvoie la hauteur"""
        if self.valeur is None:
            return 0
        elif self.est_feuille():
            return 1
        elif self.gauche is None:
            return 1 + self.droit.hauteur()
        elif self.droit is None:
            return 1 + self.gauche.hauteur()
        else:
            return 1 + max(self.droit.hauteur(), self.gauche.hauteur())

    def maximum(self):
        """Renvoie la valeur maximum"""
```