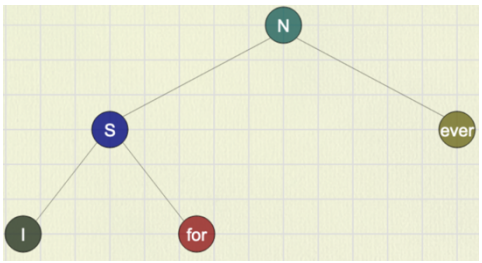
PARCOURS D´ARBRES BINAIRES

Pour implémenter un arbre binaire, nous avons utilisé la POO en utilisant la classe Node.

```
class Node():
    def __init__(self, val):
        self.val = val
        self.gauche = None
        self.droit = None
```

- val contient la valeur associée au nœud de l´arbre.
- gauche et droit pointent vers les fils du nœud, leur valeur par défaut est **None**.



Par exemple l´implémentation de cet arbre est réalisée ainsi :

```
arbre = Node('N');
arbre.gauche = Node('S');
arbre.droit = Node('ever');
arbre.gauche.gauche = Node('I');
arbre.gauche.droit = Node('for');
```

Pour parcourir un arbre binaire, nous avons vu deux approches :

- Le **parcours en profondeur** qui utilise la récursivité et se fait selon trois méthodes différentes que vous devez connaitre : préfixe, infixe ou suffixe.
- Le **parcours en largeur** qui consiste à afficher les valeurs des nœuds niveau par niveau de gauche à droite en commençant par la racine. Nous avons ici utilisé une file pour garder en mémoire le fils gauche et le fils droit du nœud visité jusqu´à ce que la file soit vide.

Nous avons également étudié les **arbres binaires de recherche** (ABR en français, BST en anglais). C'est un arbre binaire qui va être utilisé pour réaliser efficacement des opérations de recherche d'une valeur, mais aussi des opérations d'insertion et suppression de valeurs.

Un arbre binaire de recherche est un arbre binaire tel que pour tout nœud de valeur **e**:

- les valeurs de tous les nœuds de son sous-arbre gauche sont Inférieures ou égales à **e**.
- les valeurs de tous les nœuds de son sous-arbre droit sont supérieures à **e**.

Nous avons remarqué que le parcours **infixe** d´un arbre binaire de recherche retourne la liste des éléments triés dans l´ordre croissant.