Arbres et applications

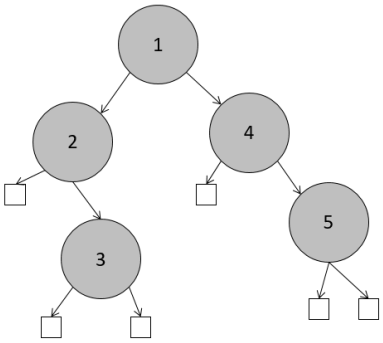
Exercices supplémentaires et compléments de théorie

1 Représentation « Liste de listes »

1) Dessiner l'arbre correspondant à la liste de listes

```
[4, [5, [6, [], []], []], 10]
```

2) Donner la liste de listes correspondant à l'arbre ci-contre



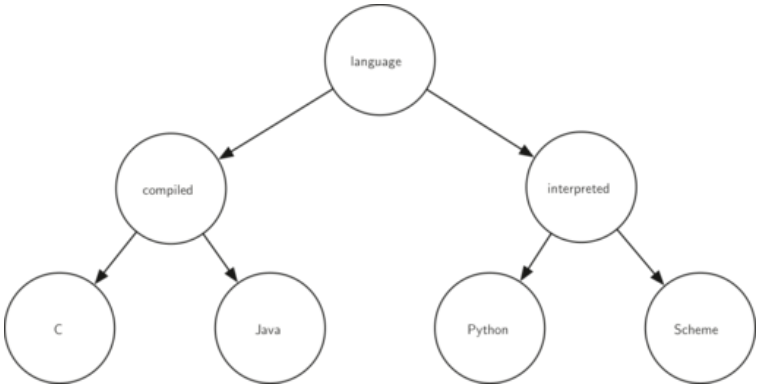
2 Exercice (Arbres)

Dessiner l'arbre résultant des appels suivants avec la représentation « Liste de listes »

```
>>> r = BinaryTree(3)
>>> insertLeft(r,4)
[3, [4, [], []], []]
>>> insertLeft(r,5)
[3, [5, [4, [], []], []], []]
>>> insertRight(r,6)
[3, [5, [4, [], []], []], [6, [], []]]
>>> insertRight(r,7)
[3, [5, [4, [], []], []], [7, [], [6, [], []]]]
>>> setRootVal(r,9)
>>> insertLeft(r,11)
[9, [11, [5, [4, [], []], []], []], [7, [], [6, [], []
]]]
```

3 Exercice (Représentation objet d'un arbre binaire)

Déterminer les appels nécessaires pour créer l'arbre suivant en utilisant la représentation **nœuds et références**



4 Parcours d'un arbre

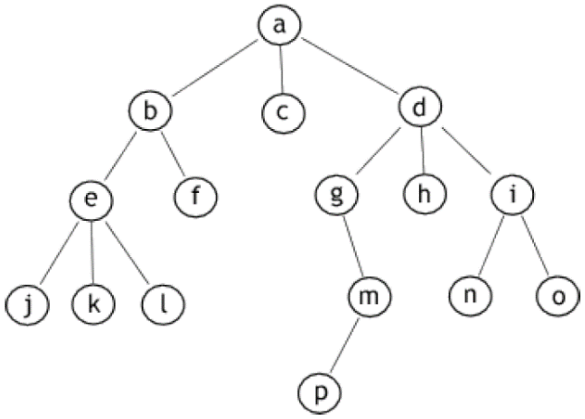
Pour l'arbre ci-contre, noter l'ordre d'affichage des nœuds lors de chaque type de parcours mentionné

1) Parcours en profondeur préfixé

2) Parcours en profondeur infixé

3) Parcours en profondeur postfixé (suffixé)

4) Parcours en largeur



5 Exercice (Arbres binaires de recherche)

On considérer les listes d'entiers suivantes `[1,2,3,4,5,6,7,8,9,10]`. Représenter l'arbre binaire de recherche résultant de l'insertion de ces nombres dans l'ordre indiqué

a) `[9, 1, 8, 6, 10, 5, 4, 7, 2, 3]`

b) `[1,2,3,4,5,6,7,8,9,10]`

c) `[10,9,8,7,6,5,4,3,2,1]`

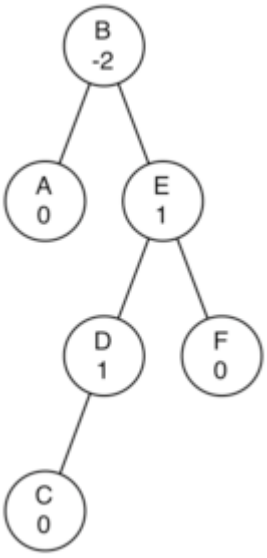
6 Entrainement (Arbres binaires de recherche)

1. Dessiner l'arbre de recherche résultant de l'insertion des clés suivantes dans l'ordre indiqué 68,88,61,89,94,50,4,76,66, et 82.

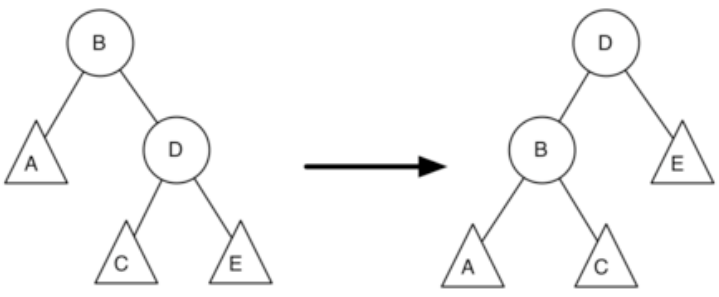
2. Même question pour une liste aléatoire de 10 nombres aléatoires.

7 Exercice (Arbre AVL)

Étant donné l'arbre suivant, effectuer les opérations de rotation nécessaires pour le ramener dans un état équilibré



3. Exprimer le facteur d'équilibre du nœud D :



8 Implémentation des arbres binaires

Compléter l'implémentation suivante de la représentation « Nœuds et références » d'un arbre binaire

```
class BinaryTree(object):
    ''' classe permettant de représenter un arbre binaire '''

    def __init__(self,rootObj):
        self.key = rootObj
        self.leftChild = None
        self.rightChild = None

    def insertLeft(self,newNode):
        if self.leftChild == None:
            self.leftChild = BinaryTree(newNode)
        else:
            t = BinaryTree(newNode)
            t.leftChild = self.leftChild
            self.leftChild = t

    def insertLeft(self,newNode):
        ''' à compléter '''









    def getRightChild(self):
        return self.rightChild

    def getLeftChild(self):
        ''' à compléter '''



    def setRootVal(self,obj):



    def getRootVal(self):
```

9 Implémentation des arbres binaires de recherche

- `BinarySearchTree()` : Créer un nouvel arbre binaire de recherche
- `put(key,val)` Ajoute une nouvelle paire (clé,valeur) dans l'arbre. Si la clé existe déjà, remplacer l'ancienne valeur par la nouvelle
- `get(key)` Étant donné une clé, retourner la valeur stockée dans l'arbre pour la clé donnée
- `del` Efface une paire (clé, valeur) de l'arbre à l'aide d'une instruction de la forme `del tree[key]`.
- `len()` Le nombre d'éléments présents dans l'arbre
- `in` retourne `True` pour une instruction de la forme `key in tree` si l'élément `key` se trouve dans l'arbre.

Indication

Une base de code se trouve sur GitHub dans le dépôt https://github.com/donnerc/ocialgos

10 Exercices (Arbres AVL)

On donne l'arbre binaire ci-contre. Les points rouges correspondent à des références **None**. Les nœuds ayant deux pointeurs None sont donc des feuilles.

a) Déterminer la hauteur du sous-arbre gauche et du sous-arbre droit de la racine de l'arbre

b) Calculez les facteurs d'équilibrage des nœuds de l'arbre suivant.