



1. Implémentation d'un arbre : Version 1

1.1. Avec la Programmation Orientée Objet ↴

Le but est d'obtenir l'interface ci-dessous.

Il est à remarquer que ce que nous allons appeler «Arbre» est en fait un nœud et ses deux fils gauche et droit.

Script Python

```
>>> a = Arbre(4) # pour créer l'arbre dont le nœud a pour valeur 4,
                # et dont les sous-arbres gauche et droit sont None
>>> a.gauche = Arbre(3) # pour donner la valeur 3 au nœud du sous-arbre gauche de a
>>> a.droit = Arbre(1) # pour donner la valeur 1 au nœud du sous-arbre droit de a
>>> a.droit.etiquette # pour accéder à la valeur du fils droit de a
```

Dessinez l'arbre créé par les instructions suivantes :

Script Python

```
>>> a = Arbre(4)
>>> a.gauche = Arbre(3)
>>> a.droit = Arbre(1)
>>> a.droit.gauche = Arbre(2)
>>> a.droit.droit = Arbre(7)
>>> a.gauche.gauche = Arbre(6)
>>> a.droit.droit.gauche = Arbre(9)
```

■ Implémentation :

► **Principe** : nous allons créer une classe `Arbre`, qui contiendra 3 attributs :

- `etiquette` : la valeur du nœud (de type `Int`)
- `gauche` : le sous-arbre gauche (de type `Arbre`)
- `droit` : le sous-arbre droit (de type `Arbre`).

Par défaut, les attributs `gauche` et `droit` seront à `None`, qui représentera l'arbre vide (ce qui n'est pas très rigoureux, car `None` n'est pas de type `Arbre` ...).

► **Encapsulation ou pas ???** :

Afin de respecter le paradigme de la Programmation Orientée Objet, nous devrions jouer totalement le jeu de l'**encapsulation** en nous refusant d'accéder directement aux attributs.

Pour cela il faut construire des méthodes permettant d'accéder à ces attributs (avec des **getters**, ou **accesseurs** en français) ou de les modifier (avec des **setters**, ou **mutateurs** en français) .

1.2. Classe Arbre avec encapsulation ♥

Script Python

```
class Arbre:
    def __init__(self, etiquette):
        self.etiquette = etiquette
        self.gauche = None
        self.droit = None

    def ajout_gauche(self, sousarbre): # mutateur
        self.gauche = sousarbre

    def ajout_droit(self, sousarbre): # mutateur
        self.droit = sousarbre

    def get_gauche(self): # accesseur
        return self.gauche

    def get_droit(self): # accesseur
        return self.droit

    def get_etiquette(self): # accesseur
        return self.etiquette

    def affiche(self):
        """permet d'afficher un arbre"""
        if self==None:
            return None
        else :
            return [self.etiquette,Arbre.affiche(self.gauche),Arbre.affiche(self.droit)]
```

Utilisation

Script Python

```
a = Arbre(4)
a.ajout_gauche(Arbre(3))
a.ajout_droit(Arbre(1))
a.get_droit().ajout_gauche(Arbre(2))
a.get_droit().ajout_droit(Arbre(7))
a.get_gauche().ajout_gauche(Arbre(6))
a.get_droit().get_droit().ajout_gauche(Arbre(9))
a.affiche()
```

Texte

```
[4, [3, [6, None, None], None], [1, [2, None, None], [7, [9, None, None], None]]]
```

Accès à l'étiquette :

Script Python

```
a.get_droit().get_gauche().get_etiquette()
```

Texte

2

Autre possibilité

Texte

```
class Arbre:
    def __init__(self,valeur):
        """Initialisation de l'arbre racine + sous-arbre gauche et sous-arbre droit"""
        self.v=valeur
        self.gauche=None
        self.droit=None

    def ajout_gauche(self,val):
        """On ajoute valeur dans le sous-arbre gauche sous la forme [val,None,None]"""
        self.gauche=Arbre(val)

    def ajout_droit(self,val):
        """ On ajoute valeur dans le sous-arbre droit sous la forme [val,None,None]"""
        self.droit=Arbre(val)

    def get_gauche(self):
        return self.gauche

    def get_droit(self):
        return self.droit

    def get_valeur(self):
        if self==None:
            return None
        else:
            return self.v

    def affiche(self):
        """permet d'afficher un arbre"""
        if self==None:
            return None
        else :
            return [self.v,Arbre.affiche(self.gauche),Arbre.affiche(self.droit)]
```

Texte

```
a = Arbre(4)
a.ajout_gauche(3)
a.ajout_droit(1)
a.droit.ajout_gauche(2)
a.droit.ajout_droit(7)
a.gauche.ajout_gauche(6)
a.droit.droit.ajout_gauche(9)
```

```
print(a.affiche())
a.get_droit().affiche()
```

Texte

```
[4, [3, [6, None, None], None], [1, [2, None, None], [7, [9, None, None], None]]]
```

2. Implémentation d'un arbre : version 2

2.1. Classe Arbre sans encapsulation ♥

Script Python

```
class Arbre:
    def __init__(self, etiquette):
        self.etiquette = etiquette
        self.gauche = None
        self.droit = None
```

C'est déjà fini !

Exo

Représenter l'arbre défini ci-dessous :

Script Python

```
a = Arbre(4)
a.gauche = Arbre(3)
a.droit = Arbre(1)
a.droit.gauche = Arbre(2)
a.droit.droit = Arbre(7)
a.gauche.gauche = Arbre(6)
a.droit.droit.gauche = Arbre(9)
```

Exo

Ajouter à classe noeud ci-dessus une méthode `est_feuille` qui renvoie `True` si le noeud est une feuille et `False` sinon.

A rajouter dans la class pour visualier sous forme de liste

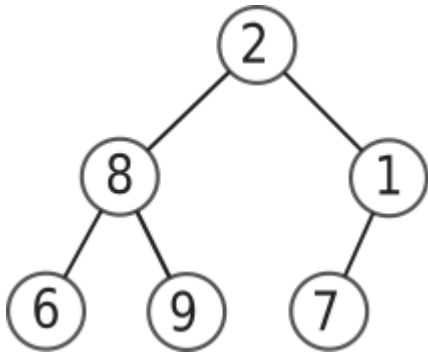
Script Python

```
def affiche(self):
    """permet d'afficher un arbre"""
    if self==None:
        return None
    else :
        return [self.etiquette,Arbre.affiche(self.gauche),Arbre.affiche(self.droit)]
```

3. Implémentation à partir de tuples imbriqués⚓

Considérons qu'un arbre peut se représenter par le tuple (valeur, sous-arbre gauche, sous-arbre droit).

L'arbre ci-dessous :



peut alors être représenté par le tuple :

Texte

```
a = (2, (8, (6,(),()), (9,(),())), (1, (7, (),()), ()))
```

Le sous-arbre gauche est alors a[1] et le sous-arbre droit est a[2].

Texte

```
a[1]
```

Texte

```
(8, (6, (), ()), (9, (), ()))
```

Texte

```
a[2]
```

Texte

```
(1, (7, (), ()), ())
```

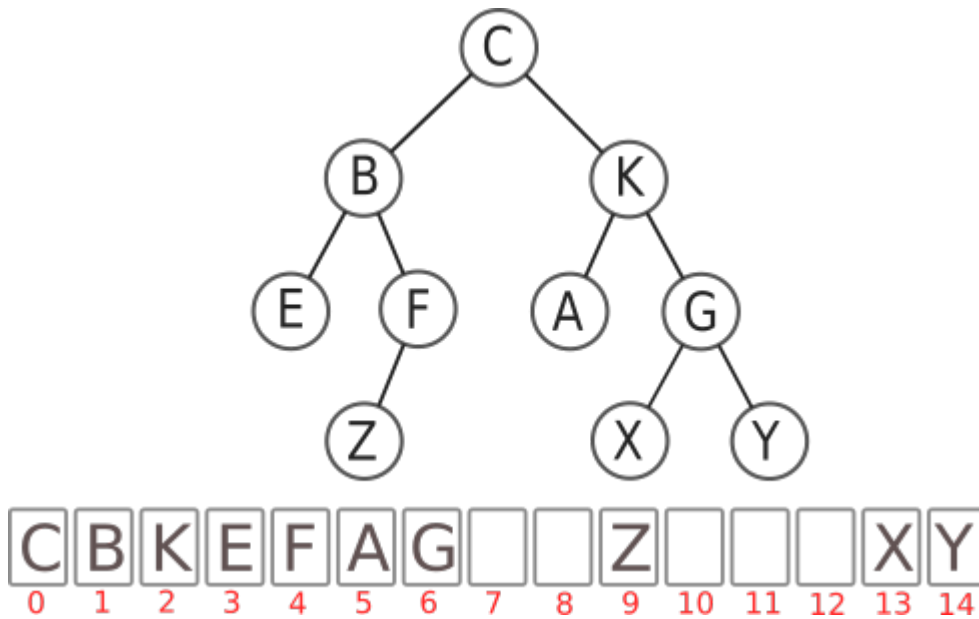
4. Implémentation à partir d'une «simple» liste↴

De manière plus surprenante, il existe une méthode pour implémenter un arbre binaire (qui est une structure hiérarchique) avec une liste (qui est une structure linéaire). Ceci peut se faire par le biais d'une astuce sur les indices :

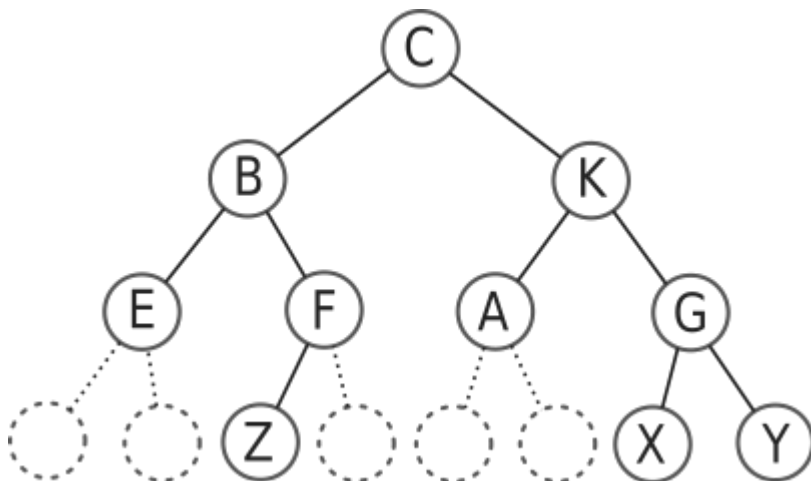
Les fils du nœud d'indice i sont placés aux indices $2i+1$ et $2i+2$.

Cette méthode est connue sous le nom de «méthode d'Eytzinger», et utilisée notamment en généalogie pour numéroter facilement les individus d'un arbre généalogique.

Exemple :



Pour comprendre facilement la numérotation, il suffit de s'imaginer l'arbre complet (en rajoutant les fils vides) et de faire une numérotation en largeur, niveau par niveau :



Remarque : parfois (comme dans le sujet 0...) la racine de l'arbre est placée à l'indice 1. Dans ce cas, les fils du nœud d'indice i sont placés aux indices $2i$ et $2i+1$.

5. Utilisation de l'implémentation : parcours, taille...📌

Dans toute la suite, sauf mention contraire, on utilisera l'implémentation en Programmation Orientée Objet, en version sans encapsulation (la plus simple). Nous allons créer des fonctions renvoyant les différents parcours d'un arbre, ou encore sa taille, sa hauteur, son nombre de feuilles... Toutes ses fonctions exploiteront la structure récursive d'un arbre.

Rappel de l'implémentation :

🐍 Script Python

```
class Arbre:
    def __init__(self, etiquette):
        self.etiquette = etiquette
        self.gauche = None
        self.droit = None
```

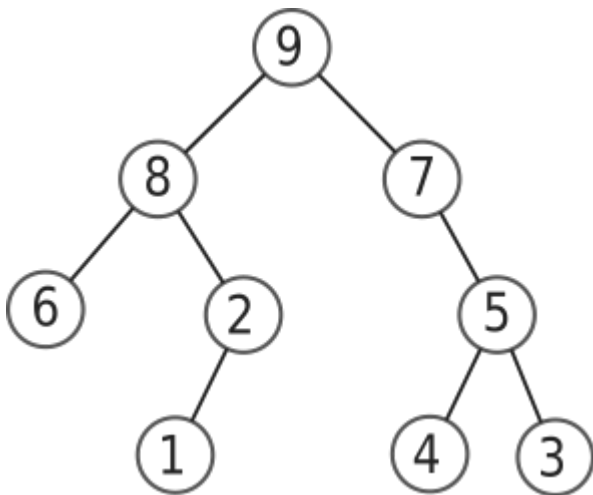
5.1. Parcours préfixe, infixe, postfixe📌

5.1.1. Parcours préfixe ♥

🐍 Script Python

```
def prefixe(arbre):
    if arbre is None :
        return None
    print(arbre.data, end = '-')
    prefixe(arbre.left)
    prefixe(arbre.right)
```

Exemple avec l'arbre



Construire l'arbre `a` à l'aide de la class précédente puis tester `prefixe(a)`

5.1.2. Parcours infixe ♥

🐍 Script Python


```
def infixe(arbre):
    if arbre is None :
        return None
    infixe(arbre.left)
    print(arbre.data, end = '-')
    infixe(arbre.right)
```



5.1.3. Parcours postfixe ou suffixe ♥


Script Python

```
def postfixe(arbre):
    if arbre is None :
        return None
    postfixe(arbre.left)
    postfixe(arbre.right)
    print(arbre.data, end = '-')
```

5.1.4. Pause vidéo

Regardez et appréciez cette vidéo

 **Video**




À l'aide de la vidéo, codez le parcours infixe en itératif

5.2. Calcul de la taille d'un arbre📌

Rappel : la taille d'un arbre est le nombre de ses nœuds.

Script Python

```
def taille(arbre):
    if arbre is None:
        return ...
    else:
        return ... + taille( ...) + taille( ...)
```

5.3. Calcul de la hauteur d'un arbre📌

Rappel : on prendra comme convention que l'arbre vide a pour hauteur 0.

Script Python

```
def hauteur(arbre):
    if arbre is None:
        return ...
```

```
else :  
    return ... + max(..., ...)
```

5.4. Calcul du nombre de feuilles d'un arbre⚓

Rappel : une feuille est un nœud d'arité 0, autrement dit sans fils gauche ni fils droit.

Script Python

```
def nbfeuilles(arbre):  
    if arbre is None:  
        return 0  
    if ... and ...:  
        return 1  
    else :  
        return ... + ...
```

5.5. Recherche d'une valeur dans un arbre⚓

On renverra True ou False en fonction de la présence ou non de la valeur dans l'arbre.

Script Python

```
def recherche(arbre, valeur):  
    if arbre is None:  
        return ...  
    if ...:  
        return True  
    else :  
        return ... or ...
```