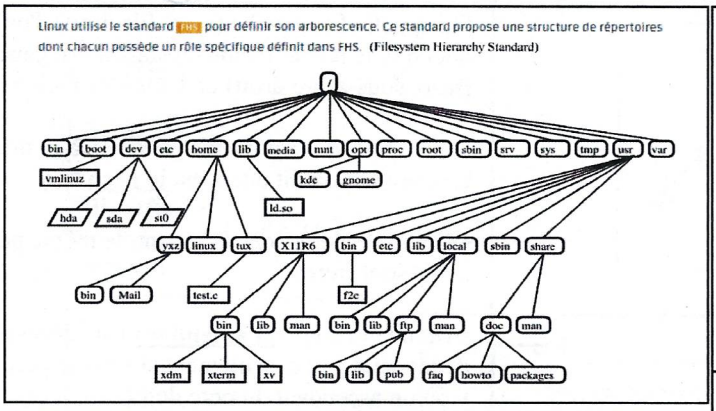
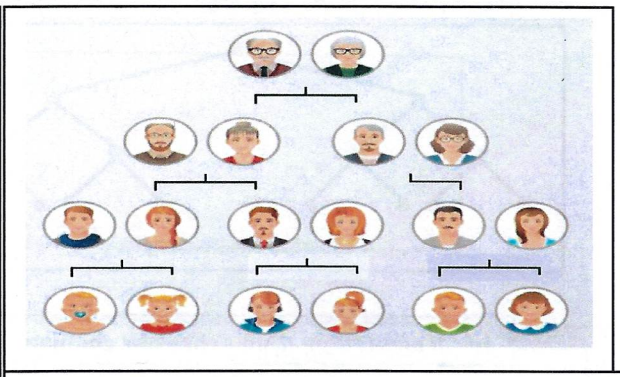
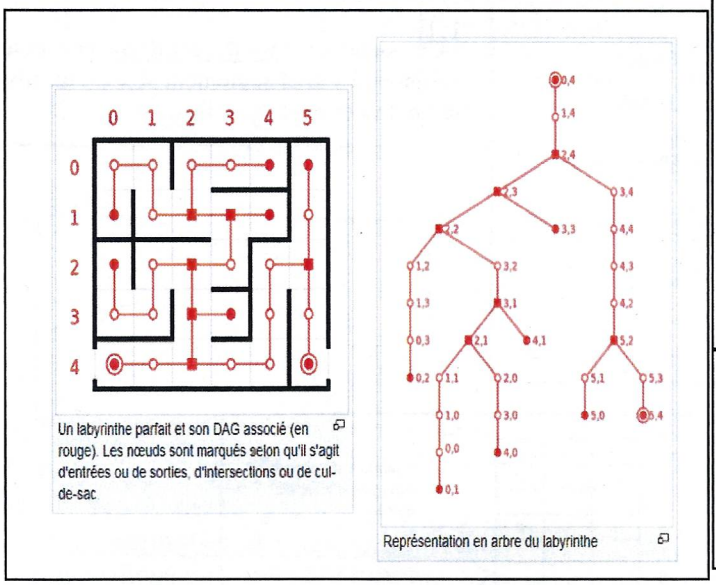
Les arbres binaires

Un arbre est une structure hiérarchique permettant de représenter de manière symbolique des informations structurées, par exemple :

- Un dossier, contenant des dossiers et des fichiers, chaque dossier pouvant contenir des dossiers et des fichiers (arbre non binaire)
- Un arbre généalogique des descendants ou des ascendants (arbre binaire).
- La représentation d'une expression arithmétique : opérateurs binaires et nombres (arbre binaire)
- Arbre syntaxique pour l'analyse d'une phrase. (arbre non binaire)
- Représentation/création d'un labyrinthe (arbre binaire ou pas)



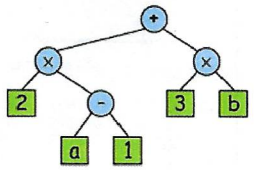


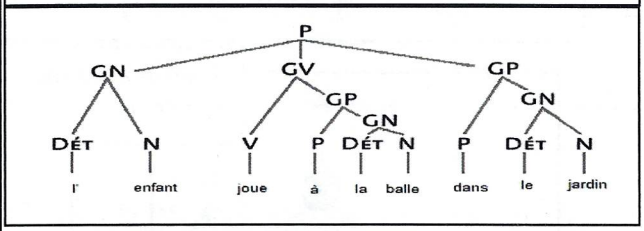


Expression arithmétique

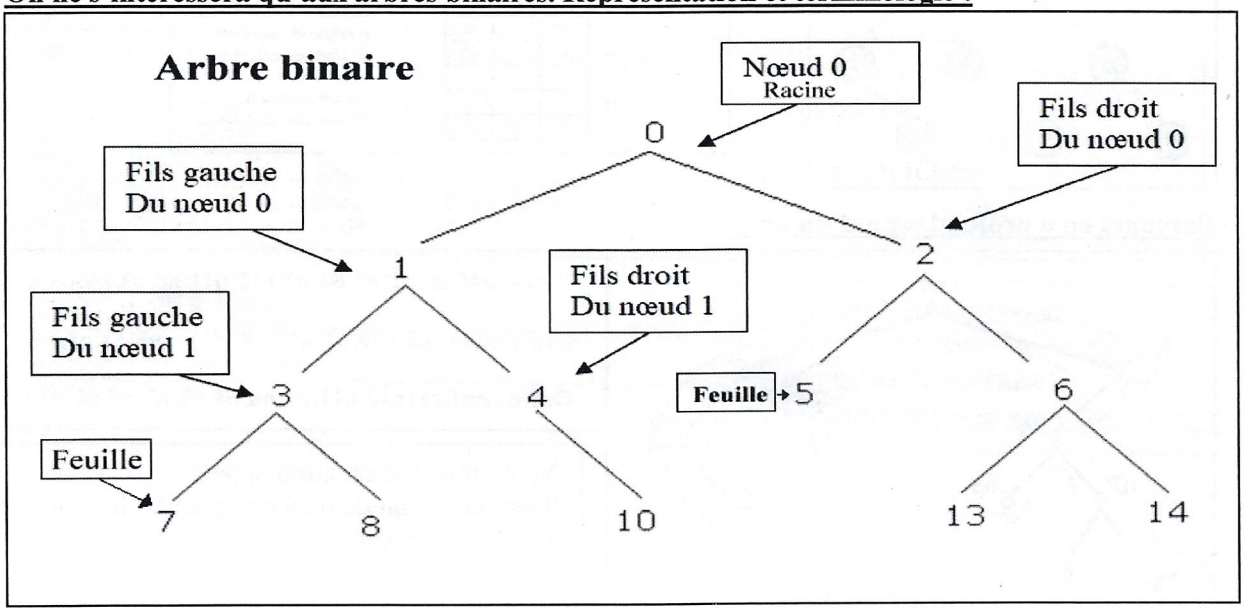
- Arbre binaire associé avec une expression arithmétique
  - Nœuds intérieurs : operateurs
  - Nœuds extérieurs : opérandes

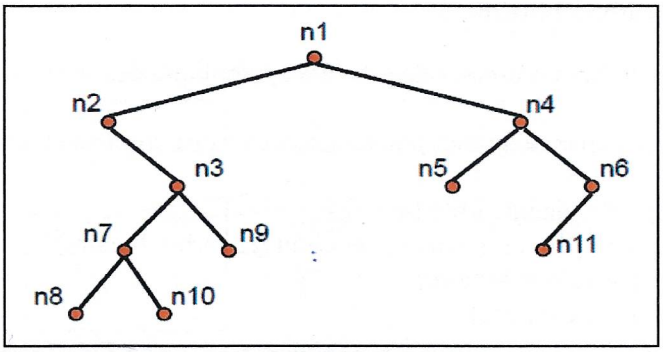
Exemple: arbre d'expression arithmétique pour l'expression
((2 × (a - 1)) + (3 × b))



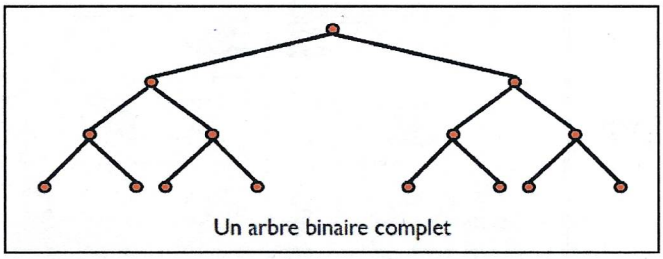


<u>On ne s'intéressera qu'aux arbres binaires. Représentation et terminologie :</u>





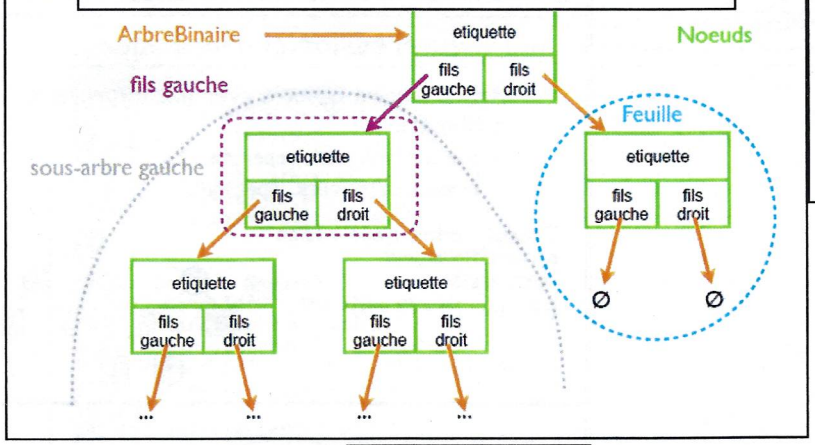
Le noeud d'étiquette n1 est la racine de l'arbre
- n2 est le fils gauche de n1, n4 son fils droit
- n2 et n4 sont frères
- n5, n8, n9, n10 et n11 sont les feuilles
- Il y a 5 branches
- la hauteur de l'arbre est 4
- la profondeur de n9 est 3


Un arbre binaire complet

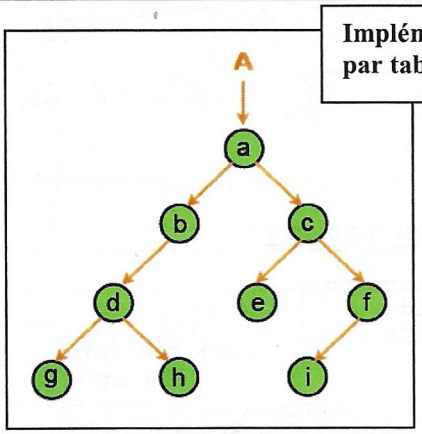
On appelle fils gauche (resp. fils droit) d'un noeud n, le noeud racine du sous-arbre gauche (resp. sous-arbre droit) de l'arbre de racine n.

- Si un noeud x a pour fils gauche (resp. droit) un noeud y, on dit que x est **le père** de y.
- On dit de deux noeuds qui ont le même père qu'ils sont frères.
- Un noeud x est **un ascendant** (**prédécesseur**) d'un noeud y si et seulement si x est le père de y ou un ascendant du père de y.
- Un noeud x est **un descendant** (**successeur**) d'un noeud y si et seulement si x est un fils de y ou un descendant d'un fils de y.

Implémentation d'un arbre par chaînage :



Implémentation par tableau :



A=4

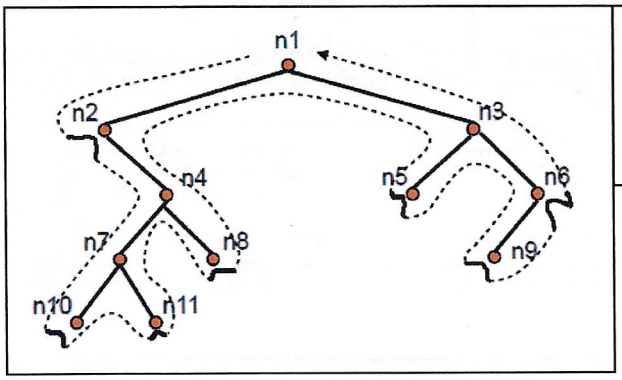
	val	G	D
1	d	7	10
2			
3	i	0	0
4	a	6	9
5			
6	b	1	0
7	g	0	0
8	e	0	0
9	c	8	12
10	h	0	0
11			
12	f	3	0

libre

Le nombre de noeuds est limité par la taille du tableau.

L'adresse associée à chaque noeud de l'arbre est arbitraire : on peut ajouter ou enlever des noeuds de l'arbre à condition de maintenir un chaînage entre les cases libres de façon à pouvoir accéder à l'espace libre du tableau.

Parcours en « profondeur » d'un arbre



Ordre préfixe : n1, n2, n4, n7, n10, n11, n8, n3, n5, n6, n9

Ordre infixe : n2, n10, n7, n11, n4, n8, n1, n5, n3, n9, n6

Ordre postfixe : n10, n11, n7, n8, n4, n2, n5, n9, n6, n3, n1

Note : postfixé est aussi appelé « suffixe »
Il est important de bien comprendre les notions d'ordre ci-dessus.

Dans le parcours préfixé, l'affichage des noeuds se fait au moment ou l'on **descend** et que l'on passe à **gauche du noeud**.
Dans le parcours infixé, l'affichage se fait au moment où l'on remonte entre les noeuds gauche et droits (entre deux frères)
Dans le parcours postfixé, l'affichage du noeud se fait en remontant à droite du noeud.

Note : pour les feuilles, elles sont systématiquement affichées immédiatement une fois visitée

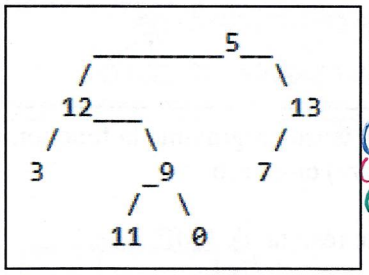
Exercice : Donner l'affichage des noeuds en préfixé, infixé et postfixé de l'arbre binaire en page 1.

Préfixé : 0, 1, 2, 7, 8, 4, 10, 2, 5, 6, 13, 14

Infixé : 8, 7, 3, 10, 4, 1, 0, 14, 13, 6, 5, 2

Postfixé : 7, 8, 3, 10, 4, 1, 13, 14, 5, 13, 14, 6, 2, 0

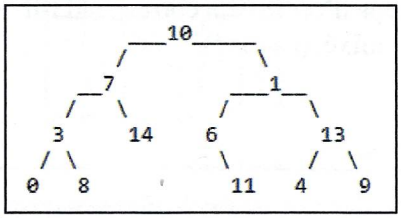
Exercice 1 :



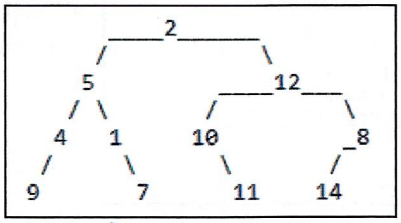
Indiquez à quels ordres correspondent les affichages ci-dessous :

```
[Node(5), Node(12), Node(3), Node(9), Node(11), Node(0), Node(13), Node(7)]
[Node(3), Node(12), Node(11), Node(9), Node(0), Node(5), Node(7), Node(13)]
[Node(3), Node(11), Node(0), Node(9), Node(12), Node(7), Node(13), Node(5)]
```

Exercice 2 :



exercice 3 :



Pour chacun des arbres, donner l'affichage des noeuds en fonction des 3 ordres (préfixe,infixe,et postfixé)

P : 10, 7, 3, 0, 9, 14, 1, 6, 11, 13, 4, 9
I : 8, 0, 3, 14, 7, 10, 11, 6, 9, 4, 13, 1
P : 0, 8, 3, 14, 7, 10, 11, 6, 4, 9, 13, 1, 10

P : 2, 5, 4, 9, 1, 7, 12, 10, 11, 8, 14
I : 9, 4, 5, 7, 1, 2, 11, 10, 12, 14, 8
P : 9, 4, 5, 7, 1, 2, 11, 10, 12, 14, 8

Programmation des arbres en python

Nous allons utiliser la bibliothèque binarytree qui nécessite une installation préalable (pip install binarytree , infos sur : https://pypi.org/project/binarytree/)

```
1  from binarytree import tree,bst,heap,build,Node
2
3  arbre1 = tree()   # créer un arbre aléatoire
4  print (arbre1)
5
6  print (arbre1.preorder) # affichage préfixé
7  print (arbre1.inorder)  # affichage infixé
8  print (arbre1.postorder) # affichage postfixé
```

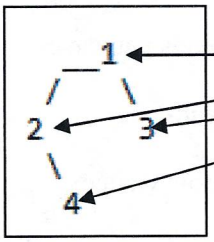
Tester le code pour vous approprier les instructions.
Nous allons ensuite construire nos propres arbres

```
10 # Arbre complet :
11 arbre2 = heap(height=4, is_perfect=True)
12 print (arbre2)
```

Rajouter le code ci-contre.
Modifier la valeur de height et de is_perfect (True/False)
Sur quoi agissent ces paramètres ?

Rep :

On veut réaliser l'arbre suivant, taper et tester le code :



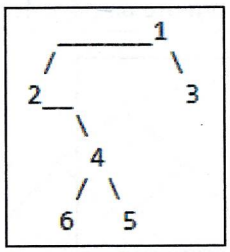

```
14  #construction d'un arbre
15  racine = Node(1)
16  racine.left = Node(2)
17  racine.right = Node(3)
18  racine.left.right = Node(4)
19
20  print (racine)
```

Taper le code ci-contre et compléter le afin de générer l'arbre avec les noeuds 1 à 6.
Comment appelle t-on les noeud 3, 5 et 6 de cet arbre ?

Le noeud 4 a t-il un frère ?

Donner le fils gauche de 4.

Qui est le père du noeud 2 ?



Code à compléter :

→ Tester aussi l'instruction **print (racine.levelorder)** (pour afficher la liste des noeuds)

Fonction visite récursive de notre arbre :

```
1   from binarytree import tree,bst,heap,build,Node
2
3   def visite(arbre):
4       if arbre == None:
5           return
6
7       print (arbre.value,end =" ")
8       visite (arbre.left)
9       visite (arbre.right)
10
11
12  racine = Node(1)
13  racine.left = Node(2)
14  racine.right = Node(3)
15  racine.left.right = Node(4)
16  racine.left.right.right = Node(5)
17  racine.left.right.left = Node(6)
18
19  print (racine)
20  visite(racine)
```

Ajouter à votre programme la fonction visite(arbre) ci-contre.

Donner le résultat de l'affichage :

A quel type d'ordre cela correspond t-il (préfixé,infixé, postfixé) ?

```
3   def visite(arbre):
4       if arbre == None:
5           return
6
7
8       visite (arbre.left)
9       print (arbre.value,end =" ")
10      visite (arbre.right)
```

Modifier maintenant juste la position du print et tester.
A quel ordre d'affichage des noeuds cela correspond t-il ?

```
3   def visite(arbre):
4       if arbre == None:
5           return
6
7       visite (arbre.left)
8       visite (arbre.right)
9       print (arbre.value,end =" ")
10
```

Modifier à nouveau la position du print et tester.
A quel ordre d'affichage des noeuds cela correspond t-il ?

Travail à faire : Effectuer la décomposition du parcours de la fonction récursive tel que vue en cours.
Attention : à la fin de la fonction visite, il y a un return par défaut

```
3   def hauteur(arbre):
4       if arbre == None:
5           return 0
6
7       return max(1+ hauteur(arbre.left) ,1 + hauteur(arbre.right))
```

Que fait ce programme ?

Ecriture simple mais le déroulement est beaucoup plus complexe qu'il n'y parait ! à méditer ...