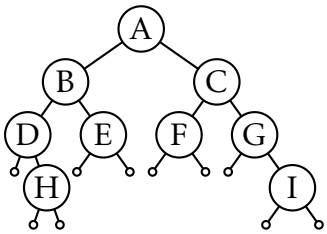
Exercices sur les arbres binaires

Arbres binaires

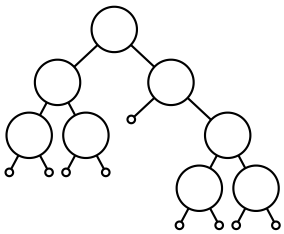
Dans cette partie, on utilisera les fonctions usuelles `gauche(arbre)`, `racine(arbre)`, `droite(arbre)` et `est_vide(arbre)`.

Exercice 1 : On considère la fonction ci-dessous :

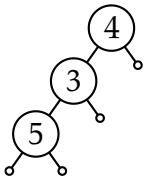
```
def parcours(arbre):
    if not est_vide(arbre):
        parcours(gauche(arbre))
        parcours(droite(arbre))
        print(racine(arbre))
```



1) Comment s'appelle un tel parcours ?
2) Dans quel ordre sont affichés les nœuds de l'arbre ci-contre ?
3) Compléter les nœuds de l'arbre ci-contre pour que les racines soient affichées dans l'ordre suivant : C O M P U T E R.
4) Reprendre l'arbre de la question 2 et donner l'affichage avec un parcours infixe.



Exercice 2 : On appelle **peigne gauche** un arbre binaire où tous les descendants droits sont vides. L'arbre vide est un peigne gauche.

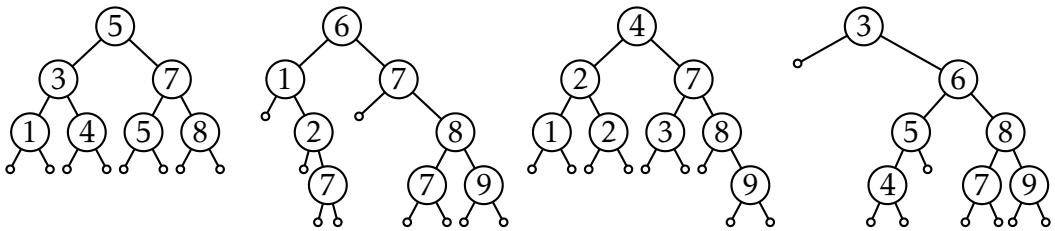


1) Écrire une fonction `est_peigne_gauche(arbre)` qui renvoie un booléen indiquant si `arbre` est un peigne gauche ou non.
2) On utilise des triplets `(gauche, racine, droite)` pour représenter les arbres binaires. L'arbre vide sera noté `N`. Ainsi, l'arbre ci-dessus est représenté par : `(((N, 5, N), 3, N), 4, N)`.
   a) Écrire la représentation d'un peigne gauche de 4 nœuds contenant chacun la valeur 0.
   b) Écrire une fonction `peigne_gauche(n)` qui renvoie un peigne gauche de `n` nœuds contenant tous 0.

Arbres binaires de recherche

Pour cette partie, on accepte que les arbres binaires de recherche contiennent plusieurs fois la même valeur. Une valeur `v` peut se retrouver aussi bien dans le descendant gauche que dans le descendant droit d'un nœud dont la racine est `v`.

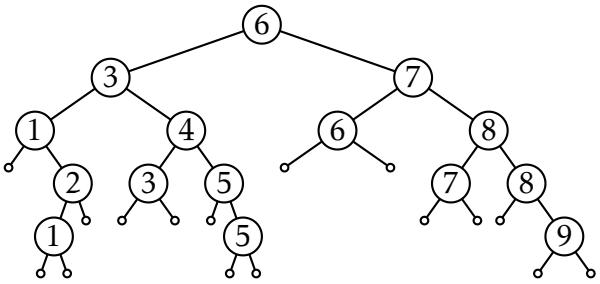
Exercice 3 : Parmi les 4 arbres ci-dessous, expliquer lesquels ne sont pas des arbres binaires de recherche.



Exercice 4 : On utilise l'algorithme ci-dessous pour ajouter une valeur `val` à un arbre binaire de recherche :

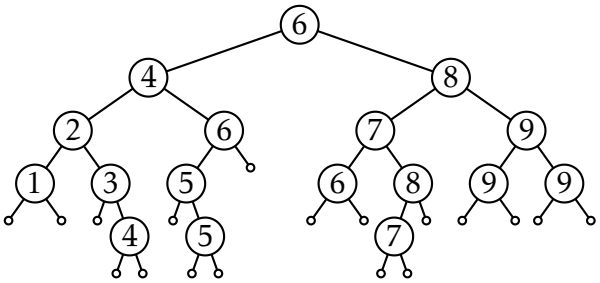
- Si l'arbre est vide, on crée une feuille contenant la valeur.
- Sinon, si la racine est strictement supérieure à `val`, on l'ajoute au descendant gauche.
- Sinon, on ajoute `val` au descendant droit.

1) Pour chacune des listes de nombres suivants, dessiner l'arbre obtenu en ajoutant les nombres dans l'ordre, en partant de l'arbre vide.
   a) `[4, 1, 3, 5, 8, 7, 9]`
   b) `[5, 1, 3, 7, 5, 1, 6]`
   c) `[8, 9, 2, 7, 1, 3, 4]`
2) Déterminer un ordre dans lequel donner les nombres suivants pour obtenir l'arbre ci-contre.



Exercice 5 : On souhaite faire une fonction récursive `compter(arbre, val)` qui renvoie le nombre d'occurences de `val` dans `arbre`.

1) On considère l'arbre `a` ci-dessous :



   Déterminer le résultat des appels ci-dessous :
   a) `compter(a, 6)`
   b) `compter(a, 2)`
   c) `compter(a, 10)`
   d) `compter(a, 0)`
2) Que doit renvoyer `compter(arbre, val)` si `arbre` est vide ?
3) En utilisant `compter(gauche(arbre), val)` et `compter(droite(arbre), val)`, indiquer ce que doit renvoyer `compter(arbre, val)` dans les cas suivants :
   a) `val` < `racine(arbre)`
   b) `val` > `racine(arbre)`
   c) `val` = `racine(arbre)`
4) Écrire le code Python de la fonction `compter(arbre, val)`.