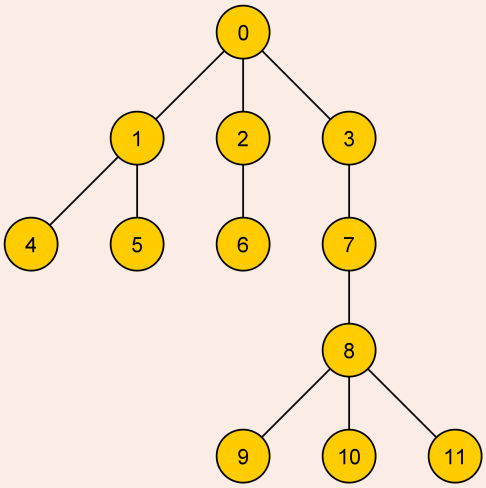
Arborescences - Exercices

Exercice 1 : Parcours d'une arborescence

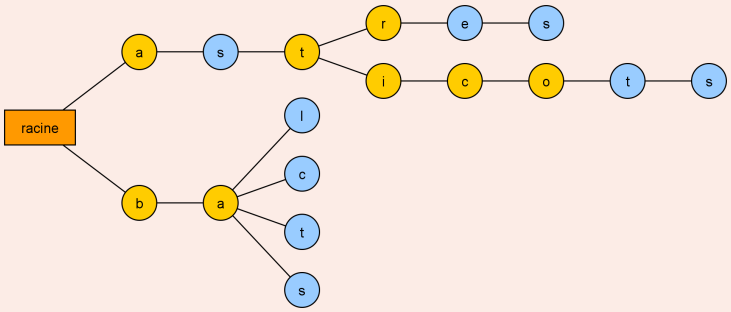
Voici une arborescence



1. Donner les valeurs des nœuds obtenues lors d'un parcours préfixe.
2. Faire de même pour un parcours postfixe.
3. Dans le fichier `nodeTS.py` se trouve une implémentation de cette arborescence.
   a. Implémenter la méthode `prefixe` pour qu'elle affiche le parcours préfixe de l'arborescence (mettre un `print` dans cette méthode)
   b. Faire de même avec la méthode `postfixe`.
   c. Modifier ces deux méthodes pour qu'elles n'affichent plus simplement le parcours, mais renvoient la liste des éléments dans l'ordre du parcours. Pour ce faire vous pouvez utiliser une variable de classe `liste_parcours` dans un premier temps.
   d. Faire la même chose, mais sans variable de classe. Penser qu'une méthode peut prendre des paramètres par défaut, et que `prefixe` peut nécessiter en paramètre la liste des nœuds déjà parcourus).

Exercice 2 : Trie

Le mot trie vient de l'Anglais et est tiré du verbe *to retrieve* (extraire). Il peut se prononcer comme le verbe *to try*.
Nous allons nous en servir pour stocker tout les mots d'un (gros) dictionnaire.



Le plus simple pour implémenter cette structure de données est de procéder ainsi : chaque nœud contient

- la lettre
- un booléen qui indique si oui ou non cette lettre termine un mot.
- un dictionnaire avec pour clés les lettres des fils du nœud et pour valeur ces nœuds.

```
class NodeTrie:
    def __init__(self, lettre=None, est_fin=False):
        self.lettre = lettre
        self.est_fin = est_fin
        self.fils = dict()
```

Ainsi, pour parcourir les lettres des nœuds qui sont les fils d'un nœud donné (noté `self`), il suffit d'exécuter
`for lettre in self.fils`.

Pour parcourir effectivement les ,nœuds on écrira
`for noeud in self.fils.values()`.

Pour créer la racine du trie, il suffit d'exécuter `racine = NodeTrie()`.

1. Écrire la méthode `ajoute_ mot` qui
   - prend en entrée un mot qui est un `str`;
   - ajoute ce mot à partir du nœud courant.

   Dans les faits, l'utilisateur ne se servira de cette méthode qu'à partir de la racine (mais pas le programmeur de cette méthode !).
2. Pour peupler le trie avec les mots qui sont dans le fichier `dico.txt`, voici ce que vous devez écrire.

```
with open('dico.txt', 'rt', encoding='utf8') as fichier:  # on ouvre le fichier
    ligne = fichier.readline()  # on lit une ligne
    while ligne: # tant que la ligne n'est pas vide
        trie.ajoute_mot(ligne[:-1])  # On enlève le dernier caractere, c'est un \n
        ligne = fichier.readline()  # On passe au mot suivant
```

Ce fichier contient presque tous les mots de la langue française, en majuscules et sans accents.

3. Écrire la méthode `contient` qui
   - prend en entrée un mot qui est un `str` ;
   - renvoie `True` si ce mot appartient à l'arborescence qui part du nœud et `False` sinon.

   Là encore l'utilisateur n'utilisera cette méthode qu'à partir de la racine (mais pas le programmeur de cette méthode !).
4. tester la méthode sur le trie.
5. (facultatif) écrire une méthode `commence_par` qui :
   - en entrée prend un début de mot qui est un `str` ;
   - renvoie la liste de tous les mots qui commencent par ce début s'il y en a.