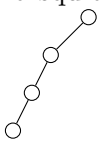
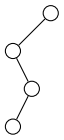
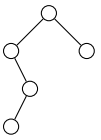
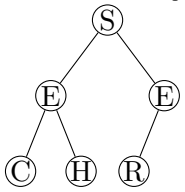
Exo 1 : Dessiner, à une symétrie près, tous les arbres possédant 3 nœuds. Combien sont binaires ?

Exo 2 : Proposer une fonction `skewness(a : 'Arbre') -> int`, récursive, qui additionne -1 lorsqu'elle part à gauche, +1 lorsqu'elle part à droite et renvoie le total.

Ainsi `skewness(`  `)` renvoit -3 ; `skewness(`  `)` renvoit -1 ; `skewness(`  `)` renvoit 0

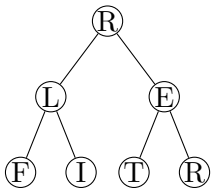
Exo 3 : On considère l'arbre ci-contre :



Quel est le mot lu de manière pré-fixé ? in-fixé ? post-fixé ?

Exo 4 : même exercice avec un parcours en largeur

Exo 5 : On considère l'arbre ci-contre :



Quel est le mot lu de manière pré-fixé ? in-fixé ? post-fixé ?

Exo 6 : même exercice avec un parcours en largeur

Exo 7 : Proposer une fonction `arbreInfix2str(a : 'Arbre') -> str`, qui traduit un arbre binaire composé de caractères en une chaîne de caractères lus de manière infixé.

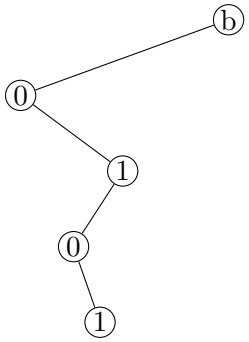
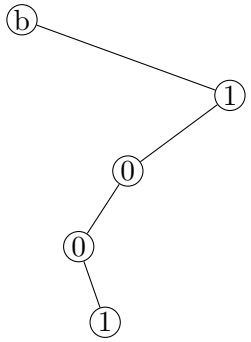
Exo 8 : même exercice de manière préfixé avec `arbrePrefix2str(a : 'Arbre') -> str`.

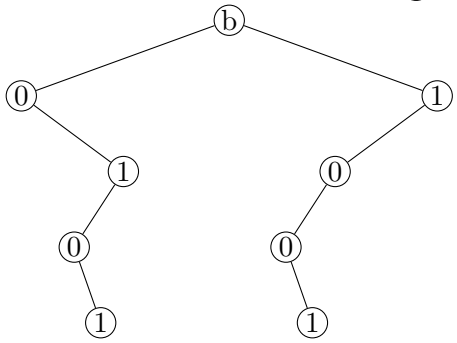
Exo 9 : même exercice en largeur avec `arbreEnLargeur2str(a : 'Arbre') -> str`.

Exo 10 : Proposer une fonction `str2arbreInfixe(mot : str) -> 'Arbre'`, qui traduit un mot en un arbre binaire comportant le mot de manière infixé.

Exo 11 : même exercice de manière postfixé avec `str2arbrePostfixe(mot : str) -> 'Arbre'`.

Exo 12 : On souhaite représenter les nombres binaires codés sur n bits à l'aide d'un arbre binaire de hauteur n. À chaque digit lu, on décide d'écrire 0 dans le fils gauche et 1 dans le droit.

Ainsi sur 4 bits, $5 = \overline{0101}^2$ s'écrira  ; tandis que $9 = \overline{1001}^2$ 

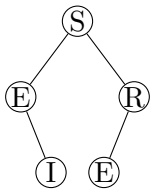
L'arbre suivant acceptera donc les nombres 5 et 9 : 

a) Écrire l'arbre qui accepte les nombres 3, 7, 10 et 13.

b) Quel arbre binaire accepte tous les nombres entiers compris entre 0 et 15 ?

c) Proposer une fonction `accepte(nb : int, ab : 'Arbre') -> bool`, qui à partir d'un nombre *nb* et d'un arbre binaire *ab* précise si ce dernier accepte ce premier.

d) Proposer une fonction `int2arbre(nb : int, hab : int) -> 'Arbre'`, qui à partir d'un nombre *nb* renvoie l'arbre de hauteur *hab* l'acceptant.



Exo 13 : **a)** Proposer une fonction `affichage() -> None`, qui affiche un arbre de manière in-fixé de sorte que `Noeud(filsG, valeur, filsD)` s'affiche `(filsG, valeur, filsD)`.
e.g. L'arbre correspond à `((None,E,(None,I,None)),S,((None,E,None),R,None))`
b) Proposer une amélioration pour ne pas afficher les `None`, ainsi cela donnera `((,E,(,I,)),S,((,E,),R,))`

Exo 14 : Écrire les mots "informatique", "ecologie", "energie", "pratique", "algorithme" et "humain" dans un arbre binaire de recherche commençant par "humain".

Exo 15 : Écrire tous les arbres binaires de recherche contenant "a", "b" et "c".

Exo 16 : Écrire une méthode `minimum() -> int`, qui renvoie la valeur minimale contenue dans un arbre binaire d'entiers. Quel est son coût temporel ?

Exo 17 : Même exo avec `maximum() -> int`.

Exo 18 : Écrire une méthode `minimumABR() -> int`, qui renvoie la valeur minimale contenue dans un arbre binaire de recherche d'entiers. Quel est son coût temporel ?

Exo 19 : Même exo avec `maximumABR() -> int`.

Exo 20 : Proposer une modification de la méthode `ajout(val : object) -> None`, afin que la valeur ne soit ajoutée que si elle n'existe pas déjà dans l'arbre binaire de recherche : `ajoutSansRepet(val : object) -> None`.

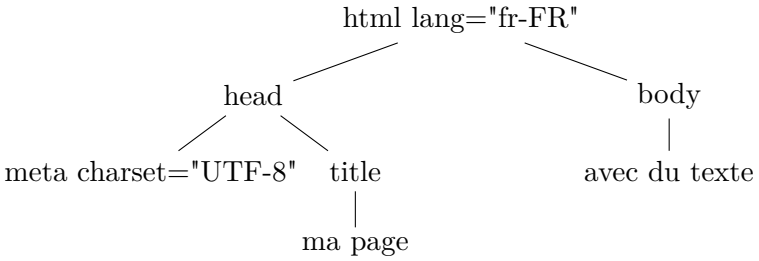
Exo 21 : Proposer une méthode `occurence(val : object) -> int`, qui renvoie le nombre d'occurence de `val` dans l'arbre binaire.

Exo 22 : Même exo avec `occurenceABR(val : object) -> int`, que l'on applique sur un arbre binaire de recherche.

Exo 23 : Proposer une méthode `occMini() -> tuple`, qui renvoie la valeur minimale d'un arbre ainsi que le nombre de fois qu'elle apparaît

Exo 24 : Un langage à balisage (tel que HTML) est facilement représentable à l'aide d'un arbre que l'on lirait de manière pré-fixé. En effet, un élément serait dans une balise s'il est le fils de la balise. Par exemple :

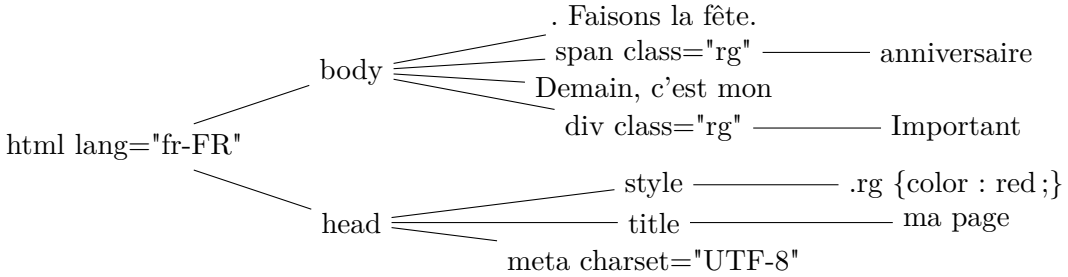
```
<html lang="fr-FR">
<head>
   <meta charset="UTF-8"/>
   <title>ma page</title>
</head> <body>
   avec du texte
</body> </html>
```



a) Écrire l'arbre correspondant au code suivant :

```
<html lang="fr-FR">  <head>
   <meta charset="UTF-8"/>
   <title>NSI</title>
</head>  <body>
   <h1> Présentation </h1>
   <p> La spécialité <b>NSI</b> traite d'informatique. </p>
   Voilà.
</body>  </html>
```

b) Réécrire cet arbre verticalement (le fils le plus en haut correspond au fils le plus à droite)



c) Écrire la page html correspondant à cet arbre.