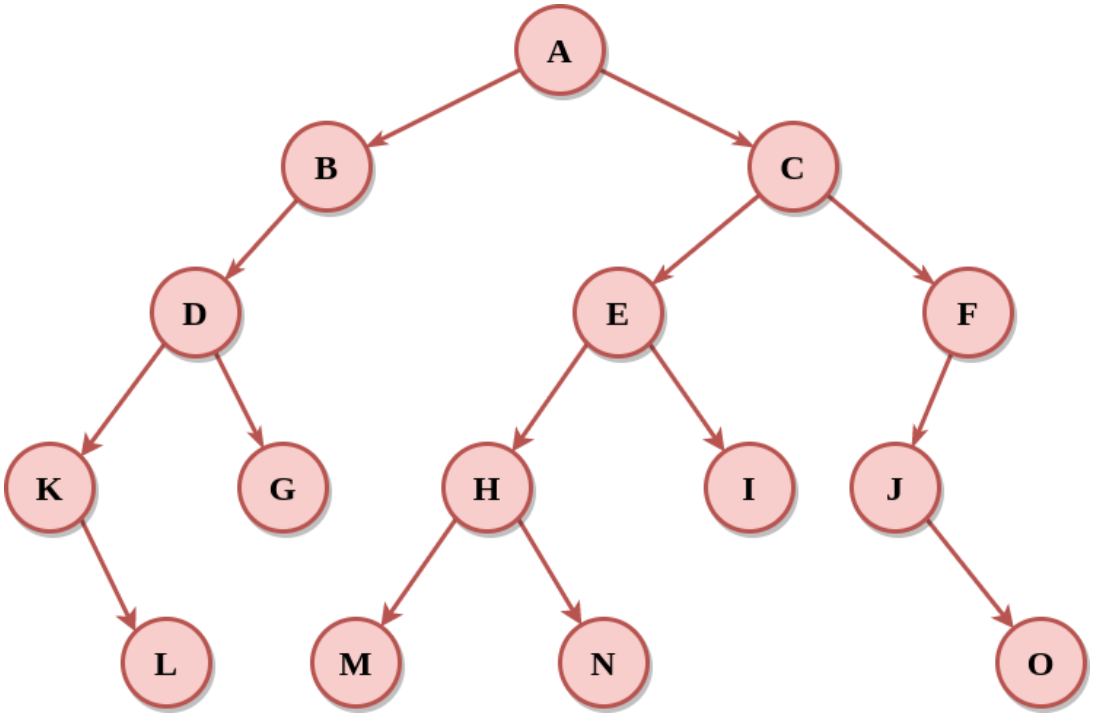
TP – CHAPITRE 6 – Arbre binaire

Sommaire

TP1 - Premier exemple d'arbre

- On considère l'arbre ci-dessous :



- La racine de cet arbre est : A .
- Les fils de la racine de cet arbre sont : B, C .
- Les feuilles de cet arbre sont : L, G, M, N, I, O .
- Les nœuds de profondeur 4 de cet arbre sont : K, G, H, I, J .
- La taille cet arbre est de : 15 .
- La hauteur de cet arbre est de : 5 .
- Le degré de cet arbre est de : 2 .

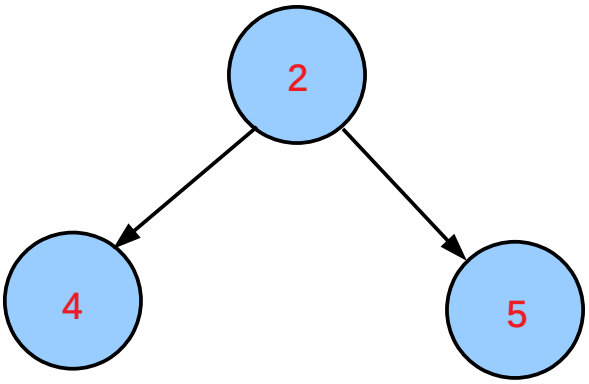
Lever la main pour valider ce TP.

TP2 - Définir une classe pour les arbres binaires

- Sur votre compte, dans le dossier **NSI** , créer un nouveau sous-dossier **TP-Chapitre6** dans lequel on rangera les TP de ce chapitre.
- Récupérer en ressource le fichier **arbre_binaire.py** dans lequel on définit une classe **AB** qui va implémenter la structure d'arbre binaire.
  Cette classe aura trois attributs privés :
  - **val** : pour la valeur de la clé ;
  - **ag** : pour le sous-arbre gauche ;
  - **ad** : pour le sous-arbre droit.
- Rédiger les **accesseurs** et les **mutateurs** de cette classe.
- Vérifier dans la console, que cette classe fonctionne correctement. L'attendu est suivant :

```
>>> arbre1 = AB(4)
>>> arbre2 = AB(5)
>>> arbre3 = AB(2,arbre1, arbre2)
>>> print(arbre1)
 (4, None, None)
>>> print(arbre2)
 (5, None, None)
>>> print(arbre3)
 (2, (4, None, None), (5, None, None))
```

- Compléter ci-dessous l'arbre binaire correspondant à l'arbre3 ci-dessus :



Lever la main pour valider ce TP.

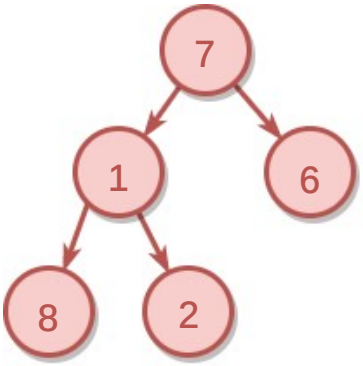
TP3 - Exemples d'arbre binaire

- On considère le code console suivant qui implémente trois arbres binaires nommés **arbre1**, **arbre2** et **arbre3** .

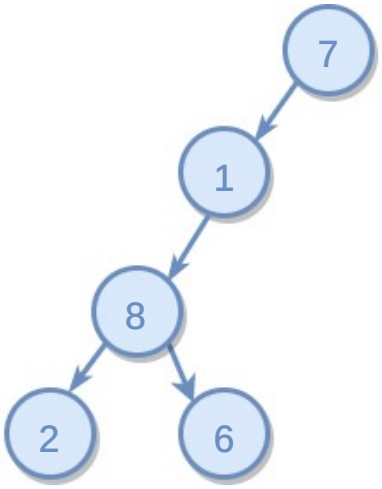
```
>>> arbre1 = AB(7, AB(1), AB(8, AB(2), AB(6)))
>>> arbre2 = AB(7, AB(1, AB(8), AB(2)), AB(6))
>>> arbre3 = AB(7, AB(1, AB(8, AB(2), AB(6))))
>>> print(arbre1)
 (7, (1, None, None), (8, (2, None, None), (6, None, None)))
>>> print(arbre2)
 (7, (1, (8, None, None), (2, None, None)), (6, None, None))
>>> print(arbre3)
 (7, (1, (8, (2, None, None), (6, None, None)), None), None)
```

- Identifier chacun des trois arbres en remplaçant le point d'interrogation par le bon numéro : **1** , **2** ou **3** **puis** compléter les nœuds des trois arbres binaires.

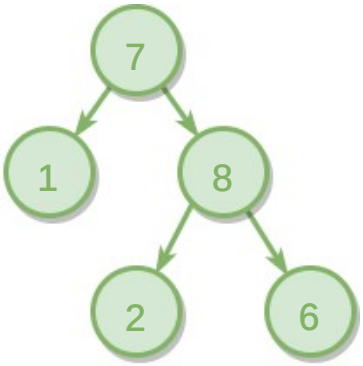
arbre2



arbre3



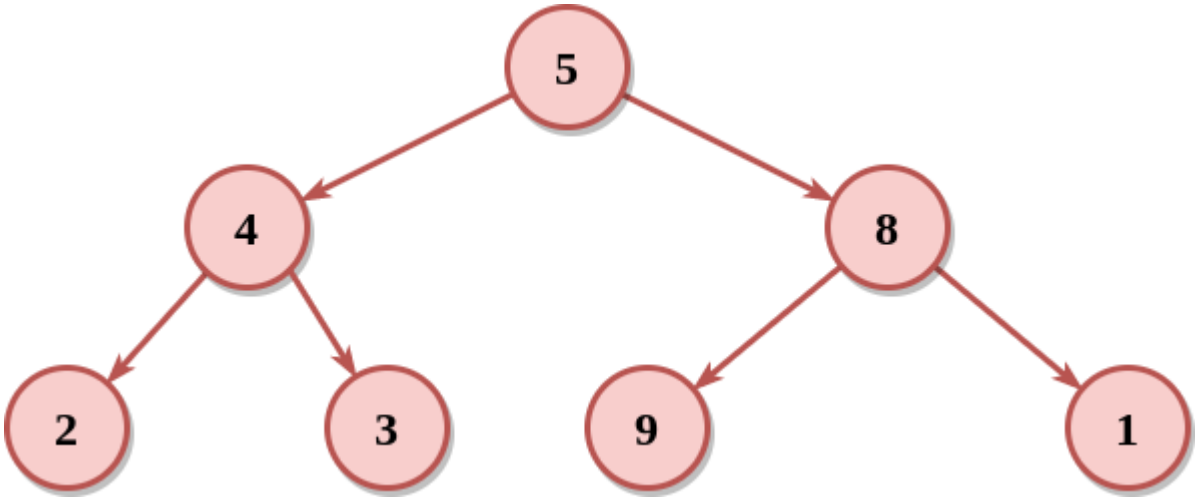
arbre1



Lever la main pour valider ce TP.

TP4 - Exemple d'arbre binaire

- Récupérer les fichiers **TP4 .py** et **dessiner_arbre.py** en ressource et les coller dans le sous-dossier **TP-Chapitre6** sur votre compte.
- Compléter le code de la ligne définissant l' **arbre1** pour qu'il corresponde à l'arbre affiché ci-dessous.



- Le retour attendu dans la console est suivant :

```
>>> print(arbre1)
  (5, (4, (2, None, None), (3, None, None)),
  (8, (9, None, None), (1, None, None)))
>>> dessiner(arbre1)


      __5__
     /     \
    4       8
   / \     / \
  2   3   9   1
```

Lever la main pour valider ce TP.

TP5 - Fonction hauteur( ) d'un arbre binaire

- Dans ce TP on va rédiger une fonction récursive **hauteur(arbre)** qui va renvoyer la hauteur de l'arbre binaire nommé **arbre** passé en argument.
- Déterminer la hauteur des arbres binaires suivants :

`arbre1 = None`	hauteur(arbre1) = 0
`arbre2 = AB(1, AB(3), AB(2))`	hauteur(arbre2) = 2
`arbre3 = AB(1, AB(2), AB(2, AB(4)))`	hauteur(arbre3) = 3
`arbre4 = AB(1)`	hauteur(arbre4) = 1
`arbre5 = AB(1, AB(2))`	hauteur(arbre5) = 2

- Expliquer dans quels cas on a :

  a. hauteur de l'arbre = 0 :

  quand l'arbre est nul

  b. hauteur de l'arbre = 1 + hauteur du sous-arbre gauche :

  quand le sous-arbre gauche est plus haut que le sous-arbre droit

  c. hauteur de l'arbre = 1 + hauteur du sous-arbre droit :

  quand le sous-arbre droit est plus haut que le sous-arbre gauche

- En utilisant le cas **a.** comme cas terminal et les cas **b.** et **c.** comme appels récursifs, compléter le pseudo-code suivant :

```
Fonction hauteur(arbre) :
    Si arbre est nul :
        retourner  0
    Sinon :
        hg = hauteur(arbre.fils_gauche)
        hd = hauteur(arbre.fils_droit)
        Si  hg > hd :
            retourner hg + 1
        Sinon :
            retourner hd + 1
```

- En utilisant la fonction maximum( ) simplifier ce pseudo-code ci-dessous :

```
Fonction hauteur(arbre) :
    Si arbre est nul :
        retourner  0
    Sinon :
        hg = hauteur(arbre.fils_gauche)
        hd = hauteur(arbre.fils_droit)
        retourner maximum(hg, hd) + 1
```


- Récupérer le fichier **TP5 .py** en ressource et l'enregistrer dans le dossier **TP-Chapitre6** sur votre compte.
- Compléter le code de la fonction récursive **hauteur(arbre)** en reprenant le deuxième pseudo-code précédent et en utilisant la fonction **max( )** native de Python.
- Tester ensuite votre code avec les exemples initiaux : arbre1, arbre2, arbre3, arbre4 et arbre5 de la page précédente, comme ci-dessous.

```
>>> arbre1 = None
>>> hauteur(arbre1)
0
>>> arbre2 = AB(1, AB(3), AB(2))
>>> hauteur(arbre2)
2
etc...
```

Lever la main pour valider ce TP.

TP6 - Fonction taille( ) d'un arbre binaire

- Dans ce TP on va rédiger une fonction récursive **taille(arbre)** qui va renvoyer la taille de l'arbre binaire nommé **arbre** passé en argument.
- Déterminer la taille des arbres binaires suivants :

`arbre1 = None`	taille(arbre1) = 0
`arbre2 = AB(1, AB(3), AB(2))`	taille(arbre2) = 3
`arbre3 = AB(1, None, AB(3))`	taille(arbre3) = 2
`arbre4 = AB(1)`	taille(arbre4) = 1
`arbre5 = AB(1, AB(2), AB(3,AB(1)))`	taille(arbre5) = 4

- Répondre aux deux questions ci-dessous :

  a. Dans quel cas a-t-on : `taille(arbre) = 0` :

  quand l'arbre est nul

  b. Exprimer la taille de l'arbre en fonction de la taille de ses deux sous-arbres : **arbre.fils-gauche** et **arbre.fils_droit** :

  taille(arbre) = taille(arbre.fils_gauche) + taille(arbre.fils_droit) + 1

- En utilisant le cas **a.** comme cas terminal et le cas **b.** comme appel récursif, compléter le pseudo-code suivant :

```
Fonction taille(arbre) :
    Si arbre est nul                    :
        retourner  0
    Sinon :
        retourner
                   taille(arbre.fils_gauche) +
                   taille(arbre.fils_droit) + 1
```

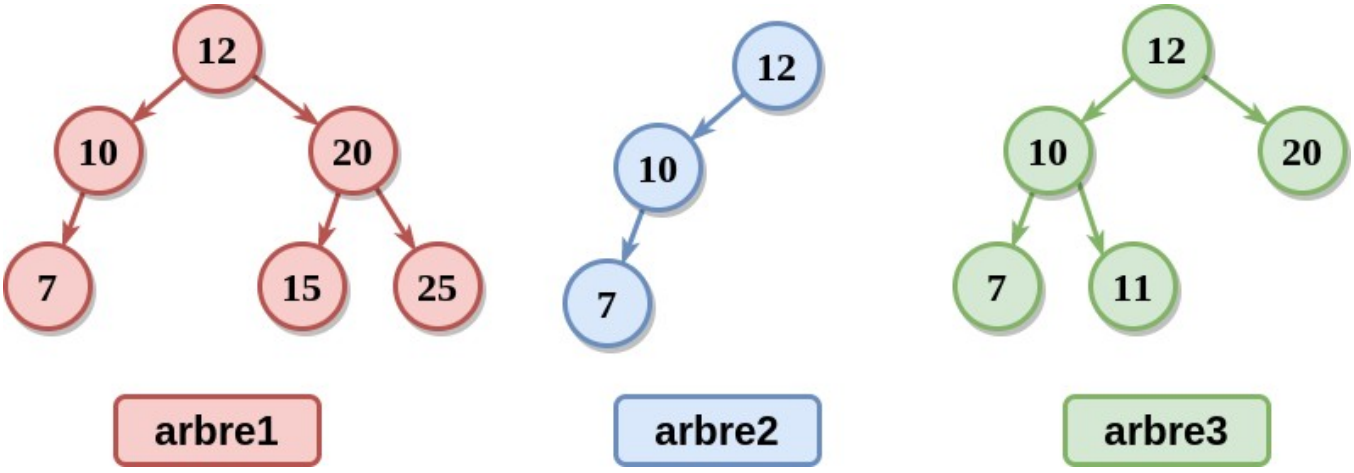
- Dupliquer le fichier **TP5 .py** en **TP6 .py** .
- Rédiger en Python, une fonction récursive **taille(arbre)** qui renvoie la taille de l'arbre binaire passé en argument.
- Tester ensuite votre code avec les exemples initiaux : arbre1, arbre2, arbre3, arbre4 et arbre5 de la page précédente, comme ci-dessous.

```
>>> arbre1 = None
>>> taille(arbre1)
0
etc...
```

Lever la main pour valider ce TP.

TP7 - Méthode .hauteur_tout_a_droite( ) d'un arbre binaire

- Dans ce TP on va rédiger une méthode **hauteur_tout_a_droite(self)** dans la classe **AB** qui va donner la hauteur entre la racine de l'arbre et la feuille de l'arbre située tout à droite.
- Pour chacun des exemples suivants, donner la hauteur tout-à-droite.



- **hauteur_tout_a_droite(arbre1) =** `3`
- **hauteur_tout_a_droite(arbre2) =** `1`
- **hauteur_tout_a_droite(arbre3) =** `2`

- Dans le fichier **arbre_binaire.py** , compléter la classe **AB** en ajoutant la méthode **hauteur_tout_a_droite( )** dont le code est suivant :

```
def hauteur_tout_a_droite(self):
    if self == None:
        return 0
    else:
        hd = self.__ad.hauteur_tout_a_droite()
        return 1 + hd
```

- Pour l'arbre binaire : **arbre4 = AB(1, AB(2), None)** , que devrait renvoyer l'instruction : **arbre4.hauteur_tout_a_droite()** : `1`

- Dans la console tester le code suivant :

```
>>> arbre4 = AB(1, AB(2), None)
>>> arbre4.hauteur_tout_a_droite()
```

- Recopier ci-dessous le message d'erreur qui s'affiche :

```
AttributeError: 'NoneType' object has no attribute
'hauteur_tout_a_droite'
```


- En effet, un objet None comme le premier sous-arbre droit n'a pas de méthode **hauteur_tout_a_droite( )** puisqu'il est vide. Il faut donc que l'algorithme se termine une étape avant le **None**, comme dans le cas déjà vu précédemment des listes chaînées. Compléter alors le code ci-dessous :

```
def hauteur_tout_a_droite(self):
    if self.__ad == None :
        return 1
    else:
        hd = self.__ad.hauteur_tout_a_droite()
        return 1 + hd
```

- Modifier le code de la méthode **hauteur_tout_a_droite( )** dans la classe **AB**, puis tester dans la console que cette méthode fonctionne avec les trois arbres binaires **arbre1**, **arbre2** et **arbre3** vu au début de ce TP7.

```
>>> arbre1 = AB(12, AB(10, AB(7)), AB(20, AB(15), AB(25)))
>>> arbre1.hauteur_tout_a_droite()
3
etc ...
```

Lever la main pour valider ce TP.

TP8 - Méthode .hauteur( ) d'un arbre binaire

- Dans ce TP on va rédiger une méthode **hauteur(self)** dans la classe **AB** . Le code est plus compliqué et plus long que la fonction **hauteur( )** du TP6 et que celui de la méthode **hauteur_tout_a_droite( )** du TP7, mais on va s'inspirer de ces deux codes.
- Compléter le code python ci-dessous de la méthode **hauteur(self)** :

```
def hauteur(self):
    # Cas d'une feuille
    if self.__ag == None and self.__ad == None:
        return 1
    # Cas où il n'y a pas de fils gauche, mais un fils droit
    elif self.__ag == None:
        hd = self.__ad.hauteur()
        return hd + 1
    # Cas où il n'y a pas de fils droit, mais un fils gauche
    elif self.__ad == None:
        hg = self.__ag.hauteur()
        return hg + 1
    # Cas où il y a un fils gauche et un fils droit
    else :
        hg = self.__ag.hauteur()
        hd = self.__ad.hauteur()
        return max(hg, hd) + 1
```

- Rédiger maintenant cette méthode dans la classe **AB** dans le fichier **arbre_binaire.py** .
- Exemple de retour attendu dans la console :

```
>>> arbre1 = AB(2)
>>> arbre1.hauteur()
1

>>> arbre2 = AB(5, AB(6), AB(2))
>>> arbre2.hauteur()
2

>>> arbre3 = AB(1, AB(2, AB(4, AB(5), AB(6))))
>>> arbre3.hauteur()
4
```

Lever la main pour valider ce TP.

TP9 - * Méthode .taille( ) d'un arbre binaire

- Reprendre le fichier **arbre_binaire.py** . Dans la classe **AB** rédiger une méthode **taille(self)** qui renvoie la taille de l'arbre binaire.
- Exemple de retour attendu dans la console :

```
>>> arbre1 = AB(2)
>>> arbre1.taille()
1

>>> arbre2 = AB(5, AB(6), AB(2))
>>> arbre2.taille()
3

>>> arbre3 = AB(1, AB(2, AB(4, AB(5), AB(6))))
>>> arbre3.taille()
5
```

Lever la main pour valider ce TP.

TP10 - Fonction contient( ) d'un arbre binaire

- Dupliquer le fichier **TP6 .py** en **TP10 .py** .
- Rédiger une fonction récursive **contient(val, arbre)** qui renvoie **True** si la valeur **val** est une clé de l'arbre et **False** sinon.
- Exemple de retour attendu dans la console :

```
>>> arbre3 = AB(1, AB(2, AB(4, AB(5), AB(6))))
>>> contient(3, arbre3)
False
>>> contient(5, arbre3)
True
>>> contient(6, arbre3)
True
>>> contient(0, arbre3)
False
```

Lever la main pour valider ce TP.

TP11 - * Surcharge de l'opérateur d'appartenance in

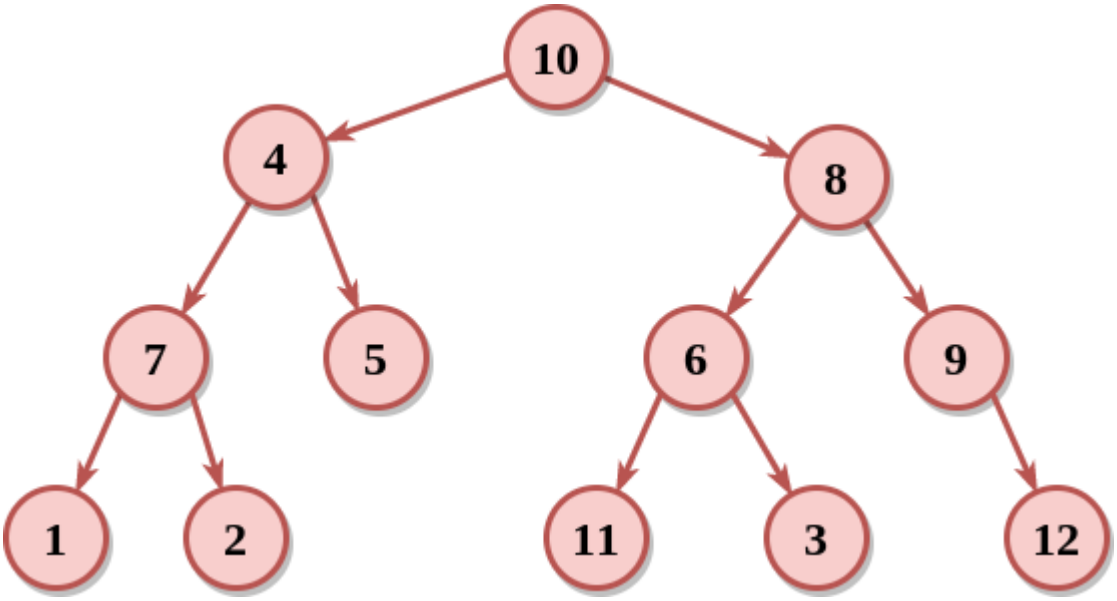
- Reprendre le fichier **arbre_binaire.py** . Dans la classe **AB** surcharger l'opérateur d'appartenance in en rédigeant la méthode **__contains__(self, val)** qui renvoie **True** si la valeur val en argument est une clé de l'arbre et **False** sinon.
- Exemple de retour attendu dans la console :

```
>>> arbre3 = AB(1, AB(2, AB(4, AB(5), AB(6))))
>>> 1 in arbre4
True
>>> 2 in arbre4
True
>>> 8 in arbre3
False
```

Lever la main pour valider ce TP.

TP12 - Parcours d'un arbre binaire

- On considère l'arbre binaire ci-dessous.



- Déterminer ci-dessous le parcours en profondeur prefixe de cet arbre binaire :

[10, 4, 7, 1, 2, 5, 8, 6, 11, 3, 9, 12]

- Déterminer ci-dessous le parcours en profondeur infixe de cet arbre binaire :

[1, 7, 2, 4, 5, 10, 11, 6, 3, 8, 9, 12]

- Déterminer ci-dessous le parcours en profondeur postfixe de cet arbre binaire :

[1, 2, 7, 5, 4, 11, 3, 6, 12, 9, 8, 10]

- Déterminer ci-dessous le parcours en largeur de cet arbre binaire :

[10, 4, 8, 7, 5, 6, 9, 1, 2, 11, 3, 12]

Lever la main pour valider ce TP.

TP13 - Parcours préfixe d'un arbre binaire

- Dans ce TP on va rédiger une fonction récursive **prefixe(arbre)** qui va retourner le parcours en profondeur préfixe (**NGD**) de l'arbre passé en argument, sous la forme d'une liste.
- Récupérer le fichier **parcours.py** en ressource et l'enregistrer dans le dossier **TP-Chapitre6** sur votre compte.
- Compléter le code de la fonction récursive **prefixe(arbre)** .
- Exemple de retour attendu dans la console :

```
>>> dessiner(arbre1)

   5
  / \
 2   1

>>> prefixe(arbre1)
[5, 2, 1]
```

```
>>> dessiner(arbre2)

     __8
    /   \
   5     7
  / \
 1   2

>>> prefixe(arbre2)
[8, 5, 1, 2, 7]
```

```
>>> dessiner(arbre3)

     __5__
    /     \
   4       8
  / \     / \
 2   3   9   1

>>> prefixe(arbre3)
[5, 4, 2, 3, 8, 9, 1]
```

Lever la main pour valider ce TP.

TP14 - Parcours infixe d'un arbre binaire

- Dans ce TP on va rédiger une fonction récursive **infixe(arbre)** qui va retourner le parcours en profondeur préfixe (**GND**) de l'arbre passé en argument, sous la forme d'une liste.
- Toujours dans le fichier **parcours.py** rédiger le code de la fonction récursive **infixe(arbre)** .
- Exemple de retour attendu dans la console :

```
>>> dessiner(arbre1)

   5
  / \
 2   1

>>> infixe(arbre1)
[2, 5, 1]
>>> dessiner(arbre2)

     __8
    /   \
   5     7
  / \
 1   2

>>> infixe(arbre2)
[1, 5, 2, 8, 7]
```

```
>>> dessiner(arbre3)

     __5__
    /     \
   4       8
  / \     / \
 2   3   9   1

>>> infixe(arbre3)
[2, 4, 3, 5, 9, 8, 1]
```

Lever la main pour valider ce TP.

TP15 - Parcours postfixe d'un arbre binaire

- Dans ce TP on va rédiger une fonction récursive **postfixe(arbre)** qui va retourner le parcours en profondeur préfixe (**GDN**) de l'arbre passé en argument, sous la forme d'une liste.
- Toujours dans le fichier **parcours.py** rédiger la fonction récursive **postfixe(arbre)** .
- Exemple de retour attendu dans la console :

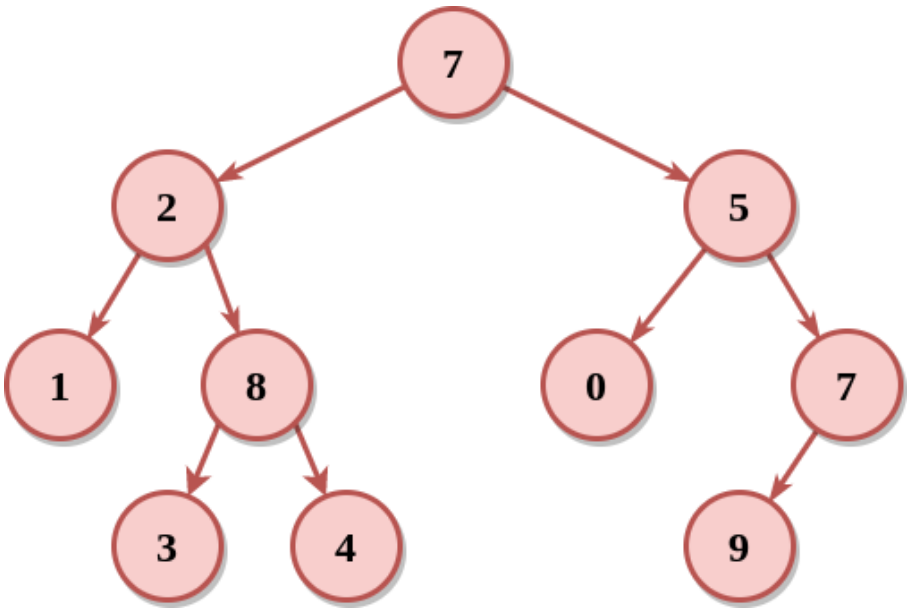
```
>>> dessiner(arbre1)

   5
  / \
 2   1

>>> postfixe(arbre1)
[2, 1, 5]
>>> dessiner(arbre2)

     __8
    /   \
   5     7
  / \
 1   2

>>> postfixe(arbre2)
[1, 2, 5, 7, 8]
```

```
>>> dessiner(arbre3)

     __5__
    /     \
   4       8
  / \     / \
 2   3   9   1

>>> postfixe(arbre3)
[2, 3, 4, 9, 1, 8, 5]
```

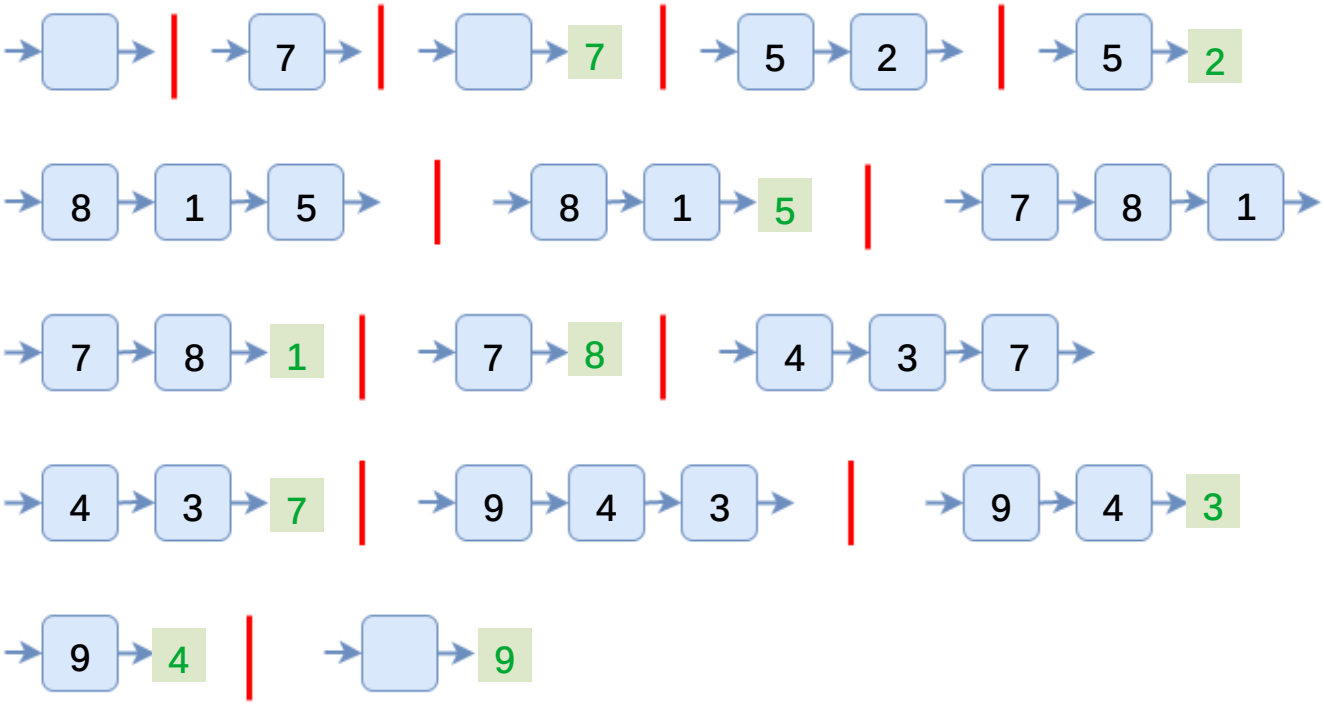
Lever la main pour valider ce TP.

TP16 - Parcours en largeur d'un arbre binaire

- On considère l'arbre binaire suivant.



- Compléter la succession d'évolutions de la file ci-dessous qui permet d'afficher le parcours en largeur de l'arbre ci-dessus, comme dans l'exemple vu en cours.



Lever la main pour valider ce TP.

TP17 - Parcours en largeur d'un arbre binaire

- On va maintenant rédiger une fonction **largeur(arbre)** qui va retourner le parcours en largeur de l'arbre passé en argument, sous la forme d'une liste. Pour cela, on va utiliser une file intermédiaire comme expliqué dans le cours.
- Récupérer les fichiers **TP17 .py** et **file.py** donnés en ressource.
- Compléter le code de la fonction **largeur(arbre)** .
- Exemple de retour attendu dans la console :

```
>>> dessiner(arbre1)

   2
  / \
 3   8

>>> largeur(arbre1)
[2, 3, 8]
>>> dessiner(arbre2)

     __8
    /   \
   5     7
  / \
 1   2

>>> largeur(arbre2)
[8, 5, 7, 1, 2]
```

```
>>> dessiner(arbre3)

     __5__
    /     \
   4       8
  / \     / \
 2   3   9   1

>>> largeur(arbre3)
[5, 4, 8, 2, 3, 9, 1]
```

Lever la main pour valider ce TP.

TP18 - * Vérifier qu'un arbre binaire est parfait

- On va maintenant rédiger une fonction **est_parfait(arbre)** qui va retourner **True** si l'arbre binaire passé en paramètre est parfait et **False** sinon.
- Dupliquer le fichier **TP17 .py** en **TP18 .py** , puis en utilisant la relation entre hauteur et taille qui caractérise un arbre parfait, rédiger la fonction **est_parfait(arbre)**.
- Exemple de retour attendu dans la console :

```
>>> dessiner(arbre1)

   2
  / \
 3   8

>>> est_parfait(arbre1)
True
>>> dessiner(arbre2)

     __8
    /   \
   5     7
  / \
 1   2

>>> est_parfait(arbre2)
False
```

```
>>> dessiner(arbre3)

     __5__
    /     \
   4       8
  / \     / \
 2   3   9   1

>>> est_parfait(arbre3)
True
```

Lever la main pour valider ce TP.

TP19 - Insérer en feuille dans un arbre binaire de recherche

- On va maintenant rédiger une nouvelle classe **ABR** pour implémenter la structure d'arbre binaire de recherche. Cette classe va hériter de la classe **AB** . Dans ce TP on suppose qu'on a déjà implémenté en Python le code de la classe **ABR** .
- Dessiner sur une feuille de brouillon l'arbre binaire de recherche **abr1** que l'on obtient après les instructions suivantes :

```
>>> abr1 = ABR(7)
>>> abr1.inserer(8)
>>> abr1.inserer(5)
>>> abr1.inserer(9)
>>> abr1.inserer(1)
>>> abr1.inserer(4)
>>> abr1.inserer(6)
```

- Dessiner sur une feuille de brouillon l'arbre binaire de recherche **abr2** que l'on obtient après les instructions suivantes :

```
>>> abr2 = ABR(7)
>>> abr2.inserer(5)
>>> abr2.inserer(3)
>>> abr2.inserer(6)
>>> abr2.inserer(10)
>>> abr2.inserer(2)
>>> abr2.inserer(8)
>>> abr2.inserer(12)
```

Lever la main pour valider ce TP.

TP20 - Insérer en feuille dans un arbre binaire de recherche

- On va maintenant rédiger la classe **ABR** pour implémenter la structure d'arbre binaire de recherche. Cette classe va hériter de la classe **AB** .
- Récupérer le fichier **abr.py** donné en ressource.
- Compléter le code de la méthode **inserer()** qui permet d'insérer **en feuille** une nouvelle clé dans l'arbre binaire de recherche.
- Exemple de retour attendu dans la console :

```
>>> abr1 = ABR(6)
>>> abr1.inserer(4)
>>> abr1.inserer(8)
>>> abr1.inserer(2)
>>> abr1.inserer(5)
>>> abr1.inserer(9)
>>> abr1.inserer(7)
>>> dessiner(abr1)

    __6__
   /     \
  4       8
 / \     / \
2   5   7   9
```

Lever la main pour valider ce TP.

TP21 - * Vérifier qu'un arbre binaire est un arbre binaire de recherche

- On va maintenant rédiger une fonction **est_un_ABR(arbre)** qui va retourner **True** si l'arbre binaire passé en paramètre est un arbre binaire de recherche et **False** sinon.
- Dupliquer le fichier **TP18 .py** en **TP21 .py** , puis rédiger la fonction **est_un_ABR(arbre)**.
- Exemple de retour attendu dans la console :

```
>>> arbre1 = AB(5, AB(4, AB(2), AB(3)), AB(8, AB(9), AB(1)))
>>> est_un_ABR(arbre1)
False
>>> arbre2 = AB(8, AB(5, AB(2), AB(6)), AB(12, AB(10), AB(14)))
>>> est_un_ABR(arbre2)
True
```

Lever la main pour valider ce TP.

TP22 - Rechercher la clé maximale dans un arbre binaire de recherche

- On va maintenant rédiger une fonction itérative **cle_max(abr)** qui va rechercher la clé maximale contenu dans un arbre de recherche.
- Récupérer le fichier **TP22 .py** donné en ressource.
- Compléter le code de la fonction **cle_max(abr)** .
- Exemple de retour attendu dans la console :

```
>>> dessiner(abr1)

      __4__
     /     \
    2       6
   / \     / \
  1   3   5   7

>>> cle_max(abr1)
7
>>> dessiner(abr2)

      4
     /
    2
   /
  1

>>> cle_max(abr2)
4
```

```
>>> dessiner(abr3)

    ____8
   /     \
  2__     10
     \      \
      7      15
     /
    6

>>> cle_max(abr3)
15
```

Lever la main pour valider ce TP.

TP23 - Vérifier qu'un arbre binaire de recherche est bien construit

- On va maintenant rédiger une fonction **bien_construit(arbre)** qui va retourner **True** si l'arbre binaire de recherche passé en paramètre est un arbre binaire de recherche bien construit et **False** sinon.
- Récupérer le fichier **TP23 .py** donnés en ressource.
- Rédiger le code de la fonction **bien_construit(arbre)** .
- Exemple de retour attendu dans la console :

```
>>> dessiner(abr1)

      __4__
     /     \
    2       6
   / \     / \
  1   3   5   7

>>> bien_construit(abr1)
True
>>> dessiner(abr2)

      4
     /
    2
   /
  1

>>> bien_construit(abr2)
False
```

Lever la main pour valider ce TP.

TP24 - Trier une liste à l'aide d'un arbre binaire de recherche

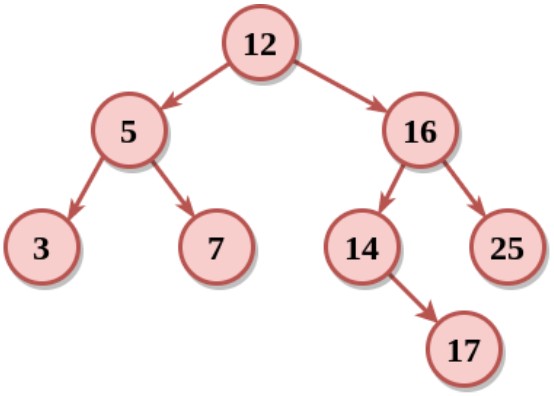
- On va maintenant rédiger une fonction **tri_par_ABR(liste)** qui prend une liste en paramètre et qui retourne une autre liste qui correspond à la liste triée.
- Dupliquer le fichier **TP23 .py** en **TP24 .py** .
- Importer la fonction **infixe** depuis le module **parcours** .
- Rédiger ensuite le code de la fonction **tri_par_ABR(liste)** .
- Exemple de retour attendu dans la console :

```
>>> tri_par_ABR([1, 7, 9, 3])
[1, 3, 7, 9]
>>> tri_par_ABR([9, 5, 1, 7, 3, 4, 6, 8, 2])
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

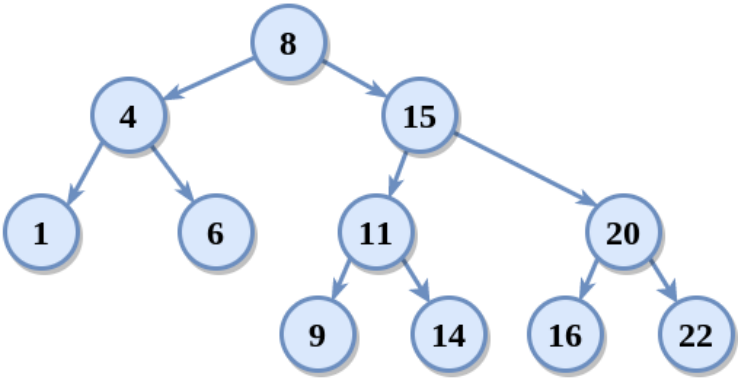
Lever la main pour valider ce TP.

TP25 - Successeur de la racine dans un arbre binaire

- On va maintenant rédiger une fonction **successeur(abr)** qui prend un arbre binaire de recherche en paramètre et qui retourne la valeur de la feuille qui correspond au successeur de la racine de l'arbre en paramètre, c'est à dire la plus petite valeur qui se trouve dans le fils droit.
- Pour chacun des arbres suivants, donner le successeur de la racine.



arbre1



arbre2

Le successeur de l'arbre1 est : 14

Le successeur de l'arbre2 est : 9

- Récupérer le fichier **TP25 .py** en ressource, puis rédiger le code de la fonction **successeur(abr)** .

- Exemple de retour attendu dans la console :

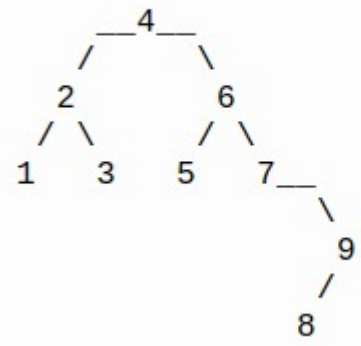
```
>>> dessiner(abr1)

      __4__
     /     \
    2       6
   / \     / \
  1   3   5   7__
                 \
                  9
                 /
                8

>>> successeur(abr1)
5
>>> dessiner(abr2)

      7__
     /   \
    5     9
   /     /
  1     8

>>> successeur(abr2)
8
```

Lever la main pour valider ce TP.

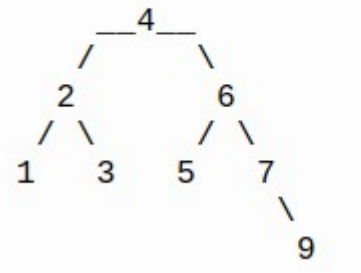
TP26 - * Supprimer un nœud dans un arbre binaire de recherche

- On va maintenant rédiger une nouvelle fonction **supprimer(abr, val)** qui permet de supprimer le nœuds dont la clé est **val** dans l'arbre binaire de recherche **abr**.
- Dupliquer le fichier **TP25 .py** en **TP26 .py** .
- Rédiger le code de la fonction **supprimer(abr, val)** .
- On distingue plusieurs situations :
  - la valeur **val** n'est pas dans l'arbre : on ne fait rien
  - la valeur **val** est une feuille de l'arbre : on supprime le lien avec le père.
  - la valeur **val** a un seul fils, on redirige le lien de son père vers ce fils.
  - la valeur **val** a deux fils : on cherche son successeur et on le substitue à la valeur **val** puis on supprime ce successeur dans le fils droit.
- Exemple de retours attendus dans la console :

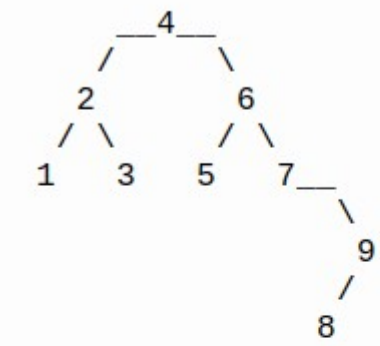
```
>>> dessiner(abr1)
```



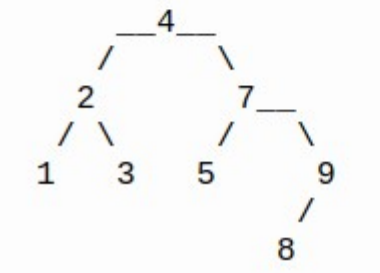

```
>>> abr1 = supprimer(abr1, 8)
>>> dessiner(abr1)
```




```
>>> dessiner(abr1)
```




```
>>> abr1 = supprimer(abr1, 6)
>>> dessiner(abr1)
```



Lever la main pour valider ce TP.

