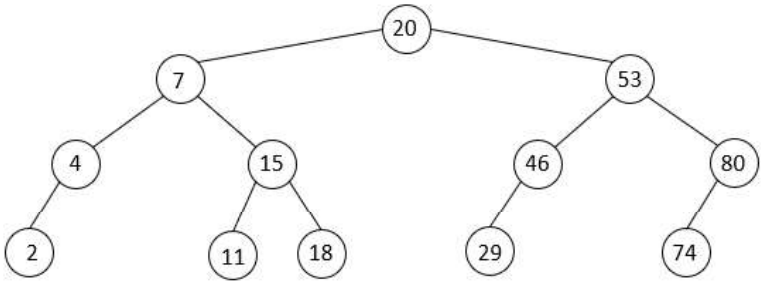
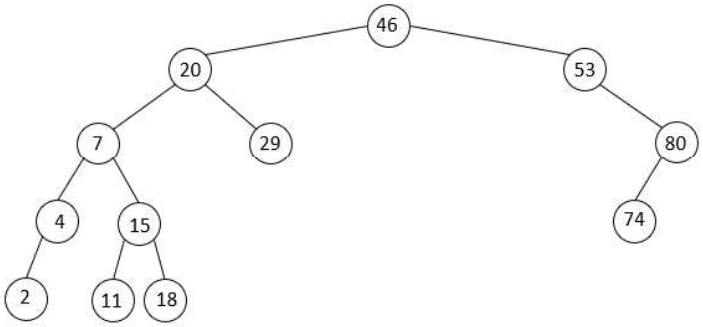
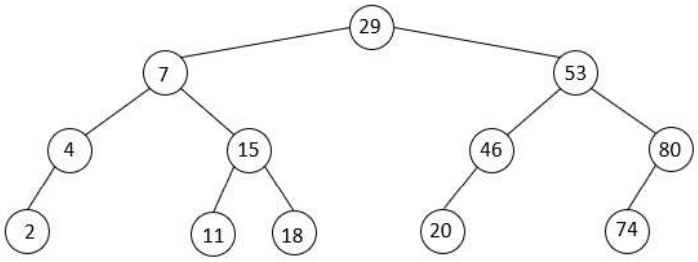
1. Parmi les trois arbres dessinés ci-dessous, recopier sur la copie le numéro correspondant à celui qui n'est pas un arbre binaire de recherche. Justifier.



Arbre 1



Arbre 2



Arbre 3

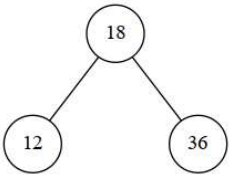
Une classe `ABR`, qui implémente une structure d'arbre binaire de recherche, possède l'interface suivante :

Numéro de lignes	Classe ABR
1	`class ABR :`
2	`    def __init__(self, valeur, sa_gauche, sa_droit):`
3	`        self.valeur = valeur #valeur de la racine`
4	`        self.sa_gauche = sa_gauche #sous-arbre gauche`
5	`        self.sa_droit = sa_droit #sous-arbre droit`
6	`    def inserer_noeud(self, valeur):`
7	`        """Renvoie un nouvel ABR avec le nœud de valeur 'valeur'`
8	`        inséré comme nouvelle feuille à sa position correcte"""`
9	`        # code non étudié dans cet exercice`
...	

On prendra la valeur `None` pour représenter un sous-arbre vide.

2. La construction d'un ABR se fait en insérant progressivement les valeurs à partir de la racine : la méthode `inserer_noeud` (dont le code n'est pas étudié dans cet exercice) place ainsi un nœud à sa "bonne place" comme feuille dans la structure, sans modifier le reste de la structure. On admet que la position de cette feuille est unique.

   a. En utilisant les méthodes de la classe ABR :
      - écrire l'instruction Python qui permet d'instancier un objet `a0`, de type ABR, ayant un seul nœud (la racine) de valeur 18.
      - écrire une séquence d'instructions qui permet ensuite d'insérer dans l'objet `a0` les deux feuilles de l'arbre de valeurs 12 et 36.

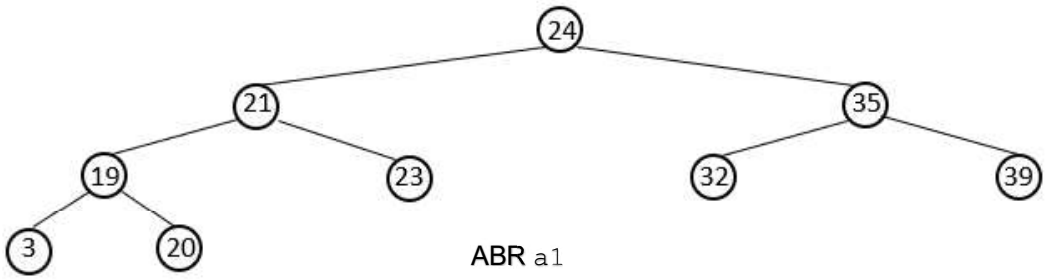


ABR `a0`

   Selon l'ordre dans lequel les valeurs sont insérées, on construit des ABR ayant des structures différentes.

   Voilà par exemple ci-dessous un ABR (nommé `a1`) obtenu en créant une instance de type ABR ayant un seul nœud (la racine) de valeur 24 puis en insérant successivement les valeurs dans l'ordre suivant :

   21 ; 35 ; 19 ; 23 ; 32 ; 39 ; 3 ; 20



ABR `a1`

   b. Dessiner sur la copie l'ABR (nommé `a2`) que l'on obtiendrait en créant une instance de type ABR ayant un seul nœud (la racine) de valeur 3 puis en insérant successivement les valeurs dans l'ordre suivant :

      20 ; 19 ; 21 ; 23 ; 32 ; 24 ; 35 ; 39

   c. Donner la hauteur des ABR `a1` et `a2`.

   d. On complète la classe ABR avec une méthode `calculer_hauteur` qui renvoie la hauteur de l'arbre.
      Recopier sur la copie les lignes 10 et 13 en les complétant par des commentaires et la ligne 14 en la complétant par une instruction dans le code ci-après de cette méthode.
      On pourra utiliser la fonction Python `max` qui prend en paramètres deux nombres et renvoie le maximun de ces deux nombres.

Numéro de lignes	Méthode `calculer_hauteur`
1	`def calculer_hauteur(self):`
2	`    """ Renvoie la hauteur de l'arbre"""`
3	`    if self.sa_droit is None and self.sa_gauche is None:`
4	`    #l'arbre est réduit à une feuille`
5	`        return 1`
6	`    elif self.sa_droit is None`
7	`    #arbre avec une racine et seulement un sous-arbre gauche`
8	`        return 1 + self.sa_gauche.calculer_hauteur()`
9	`    elif self.sa_gauche is None:`
10	`    # à compléter`
11	`         return 1 + self.sa_droit.calculer_hauteur()`
12	`    else:`
13	`    # à compléter`
14	`         return à compléter`

3. La différence de hauteur entre l'ABR `a1` et l'ABR `a2` aura des conséquences lors de la recherche d'une valeur dans l'ABR.

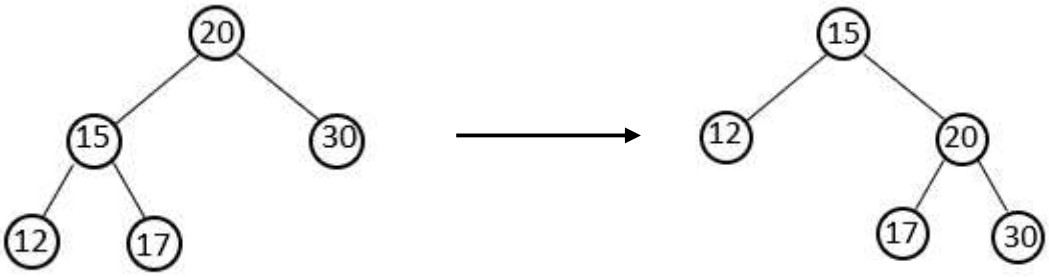
   a. Recopier et compléter sur la copie les lignes 6, 8, 11 et 13 du code ci-dessous de la méthode `rechercher_valeur`, qui permet de tester la présence ou l'absence d'une valeur donnée dans l'ABR :

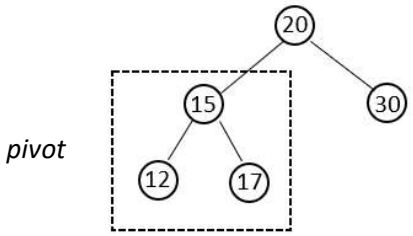
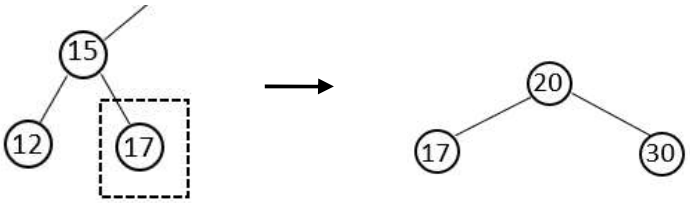
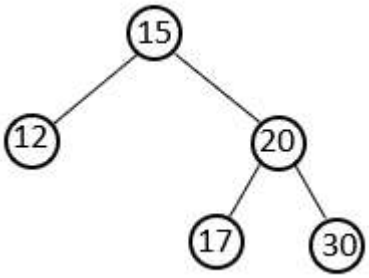
Numéro de lignes	Méthode `rechercher_valeur`
1	`def rechercher_valeur(self, v):`
2	`    """`
3	`    Renvoie True si la valeur v est trouvée dans l'ABR,`
4	`    False sinon`
5	`    """`
6	`    if à compléter`
7	`        return True`
8	`    elif à compléter and self.sa_gauche is not None:`
9	`        return self.sa_gauche.rechercher_valeur(v)`
10	`    elif v > self.valeur and self.sa_droit is not None:`
11	`        return à compléter`
12	`    else:`
13	`        return à compléter`

   b. On admet que le nombre de fois où la méthode `rechercher_valeur` est appelée pour rechercher la valeur 39 dans l'ABR `a2` est 7.
   Donner le nombre de fois où la méthode `rechercher_valeur` est appelée pour rechercher la valeur 20 dans l'ABR `a1`.

4. Il existe des algorithmes pour modifier la structure d'un ABR, afin par exemple de diminuer la hauteur d'un ABR ; on s'intéresse aux algorithmes appelés *rotation*, consistant à faire "pivoter" une partie de l'arbre autour d'un de ses nœuds.

   L'exemple ci-dessous permet d'expliquer l'algorithme pour réaliser une rotation droite d'un ABR autour de sa racine :



On appelle *pivot* le sous-arbre gauche de la racine de l'arbre	
Le sous-arbre droit du pivot devient le sous-arbre gauche de la racine	
La racine ainsi modifiée devient le sous-arbre droit du pivot et la racine du pivot devient la nouvelle racine de l'ABR	

On admet que ces transformations conservent la propriété d'ABR de l'arbre.

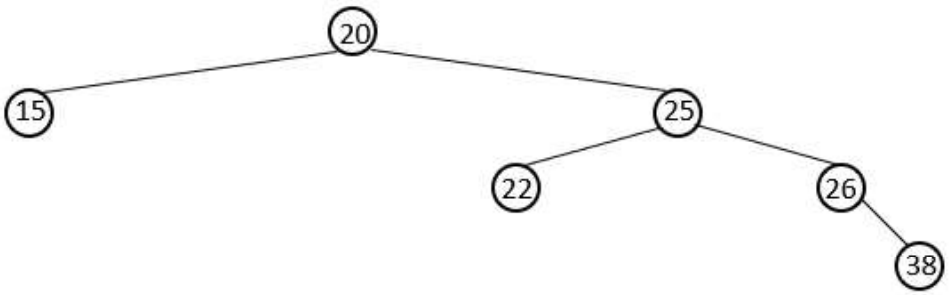
La méthode `rotation_droite` ci-après renvoie une nouvelle instance de type ABR, correspondant à une rotation droite de l'objet de type ABR à partir duquel elle est appelée :

Numéro de lignes	Méthode `rotation_droite`
1	`def rotation_droite(self):`
2	`    """ Renvoie une instance d'un ABR apres une rotation droite`
3	`        On suppose qu'il existe un sous-arbre gauche"""`
4	`    pivot = self.sa_gauche`
5	`    self.sa_gauche = pivot.sa_droit`
6	`    pivot.sa_droit = self`
7	`    return ABR(pivot.valeur,pivot.sa_gauche,pivot.sa_droit)`

Pour réaliser une rotation gauche, on suivra alors l'algorithme suivant :

- on appelle *pivot* le sous-arbre droit de la racine de l'arbre,
- le sous-arbre gauche du pivot devient le sous-arbre droit de la racine,
- la racine ainsi modifiée devient le sous-arbre gauche du pivot et la racine du pivot devient la nouvelle racine de l'ABR

a. En suivant les différentes étapes de cet algorithme, dessiner l'arbre obtenu après une rotation gauche de l'ABR suivant :



b. Écrire le code d'une méthode Python `rotation_gauche` qui réalise la rotation gauche d'un ABR autour de sa racine.