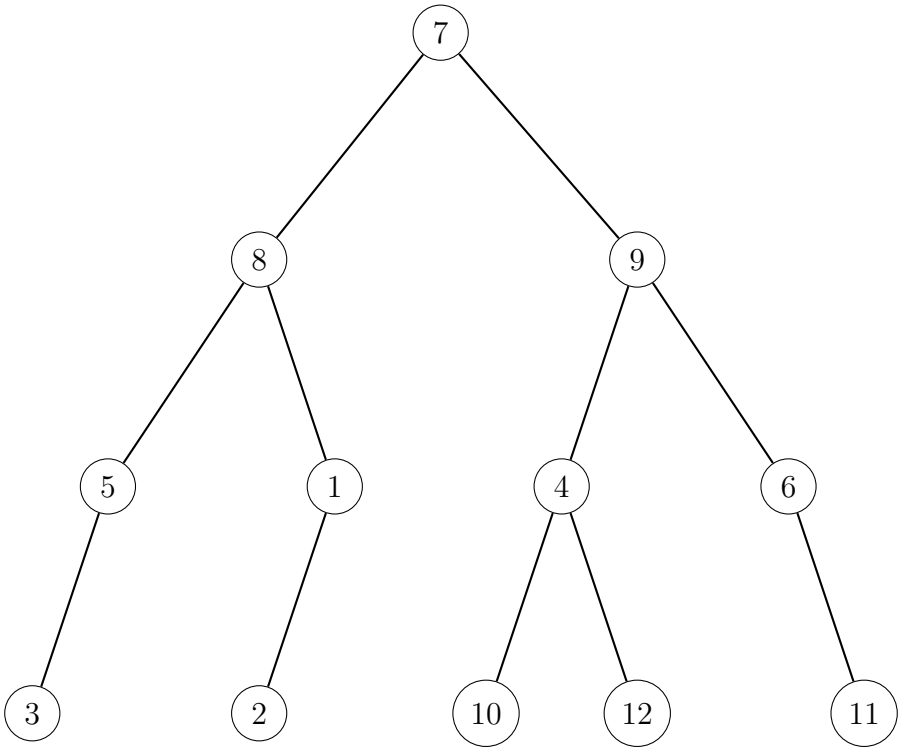
fiche 4

Arbres binaires

Version avec preuves.

Nous reprendrons la construction de l'arbre binaire vu en cours.
Nous pourrons prendre en exemple l'arbre binaire suivant :



Exercice 1 : Définir l'arbre **exemple_1**.

Correction

```
from arbre_bin import *

vide = Arb_bin()

#Exercice 1

exemple_1 = \
Arb_bin(7 , Arb_bin( 8, Arb_bin(5 , Arb_bin(3,vide,vide),vide ), \
                    Arb_bin(1,Arb_bin(2,vide,vide),vide)) , Arb_bin(9 ,\
                Arb_bin(4,Arb_bin(10,vide,vide),Arb_bin(12,vide,vide)),\
               Arb_bin(6,vide, Arb_bin(11,vide,vide))))
```

Exercice 2 :

1. Écrire la fonction `recherche` qui prend en argument un arbre et un élément. La fonction renvoie le booléen correspondant à l'appartenance de l'élément dans l'arbre.

2. Écrire la fonction `mini_maxi` qui prend en argument un arbre et renvoie le minimum et le maximum élément de l'arbre.

Correction

```
from arbre_bin import *

vide = Arb_bin()


exemple_1 = \
Arb_bin(7 , Arb_bin( 8, Arb_bin(5 , Arb_bin(3,vide,vide),vide ), \
                    Arb_bin(1,Arb_bin(2,vide,vide),vide)) , Arb_bin(9 ,\
                 Arb_bin(4,Arb_bin(10,vide,vide),Arb_bin(12,vide,vide)),\
                Arb_bin(6,vide, Arb_bin(11,vide,vide))))



#exercice 2
def recherche(arbre , element ) :
    if arbre.test_vide() :
        return False
    else :
        rac = arbre.get_racine() # getter de la racine
        a_g = arbre.a_g() # getter de l'arbre gauche
        a_d = arbre.a_d() # getter de l'arbre droit
        if rac == element :
            return True
        else :
            return recherche(a_g , element ) \
                   or recherche(a_d , element )

infini = float('inf')

def mini_maxi_i ( arbre ) : # version avec l'infini
    if arbre.test_vide() :
        return (infini , -infini )  # +infini pour le mini et -infini pour le
maxi
    else :
        rac = arbre.get_racine() # getter de la racine
        a_g = arbre.a_g() # getter de l'arbre gauche
        a_d = arbre.a_d() # getter de l'arbre droit
        (mini_g , maxi_g ) = mini_maxi_i(a_g)
        (mini_d,maxi_d ) = mini_maxi_i(a_d)
        return (min(rac , mini_g , mini_d ) ,max(rac,maxi_g , maxi_d))




def mini_maxi (arbre) : # sans l'utilisation de l'infini
    if arbre.test_vide() :
        return "erreur␣d'arbre"
    else :
        rac = arbre.get_racine() # getter de la racine
```

```
        a_g = arbre.a_g() # getter de l'arbre gauche
        a_d = arbre.a_d() # getter de l'arbre droit
        if a_g.test_vide() and a_d.test_vide() :
            return (rac,rac)
        elif a_g.test_vide() :
            (mini_d,maxi_d ) = mini_maxi(a_d)
            return (min(rac ,  mini_d ) ,max(rac, maxi_d))
        elif a_d.test_vide() :
            (mini_g , maxi_g ) = mini_maxi(a_g)
            return (min(rac ,  mini_g ) ,max(rac, maxi_g))
        else :
            (mini_g , maxi_g ) = mini_maxi(a_g)
            (mini_d,maxi_d ) = mini_maxi(a_d)
            return (min(rac , mini_g , mini_d ) ,max(rac,maxi_g , maxi_d))
```

Exercice 3 : On considère un liste. Le but est de construire simplement un arbre dont les étiquettes sont les éléments de la liste.
L'algorithme récursif est le suivant :

- Si la liste est vide, la fonction renvoie l'arbre vide.
- Sinon, la fonction divise la liste en deux.
  - La racine de l'arbre devient le centre de la liste.
  - Le sous arbre gauche est construit par la première partie de la liste.
  - Le sous arbre droit est construit par la deuxième partie de la liste.

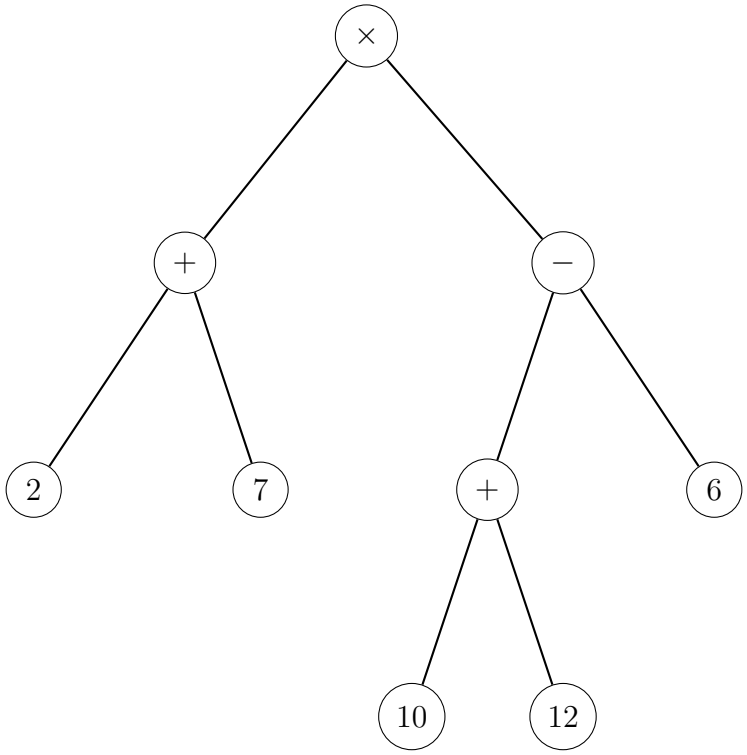
Écrire la fonction `arbre_of_liste` qui convertit une liste en un arbre binaire.

Correction

```
from arbre_bin import *

vide = Arb_bin()


exemple_1 = \
Arb_bin(7 , Arb_bin( 8, Arb_bin(5 , Arb_bin(3,vide,vide),vide ), \
                    Arb_bin(1,Arb_bin(2,vide,vide),vide)) , Arb_bin(9 ,\
                Arb_bin(4,Arb_bin(10,vide,vide),Arb_bin(12,vide,vide)),\
               Arb_bin(6,vide, Arb_bin(11,vide,vide))))



#exercice 3
def arbre_of_liste (liste ):
    if liste == [] :  # le cas ou la liste est vide
        return vide
    else :
        position_centre = len(liste)//2
        centre = liste[position_centre]
        liste_gauche = liste[:position_centre]
        liste_droite = liste[position_centre +1 :]
```

```
        a_g =arbre_of_liste( liste_gauche  )
        a_d = arbre_of_liste(liste_droite)
        return Arb_bin(centre , a_g, a_d)
```

Exercice 4 : On considère un arbre arithmétique suivant :



1. Calculer la valeur du résultat de l'exemple au dessus.
2. Déterminer les fonctions `fois`, `plus`, `moins` et `divise` qui prennent en arguments deux entiers, et renvoie le résultat du calcul.
3. Implémenter l'arbre exemple, à l'aide des fonctions précédentes.
4. Écrire la fonction `est_une_feuille`, qui prend en argument un arbre, et renvoie le booléen correspondant à l'état de feuille de l'arbre.
5. Écrire la fonction `calcul`, qui prend en argument un arbre arithmétique et renvoie le résultat du calcul de l'arbre.

Correction

```
from arbre_bin import *

vide = Arb_bin()


def plus(a,b) :
    return a+b

def fois(a,b) :
    return a*b

def moins(a,b) :
    return a-b

def divise(a,b) :
```

```
    return a/b

def est_une_feuille(arbre ) :
    return arbre.arb_gauche == vide and arbre.arb_droit == vide

exemple =Arb_bin(fois ,\
        Arb_bin(plus , Arb_bin(2,vide,vide) , Arb_bin(7,vide,vide)) ,\
        Arb_bin(moins,\
            Arb_bin(plus , Arb_bin(10,vide,vide) , Arb_bin(12,vide,vide)),\
               Arb_bin(6,vide,vide)))

def calcul(arbre) :
    if arbre == vide :
        return 0
    elif est_une_feuille(arbre) :
        return arbre.get_racine()
    else :
        return arbre.get_racine()(calcul(arbre.arb_gauche) , calcul(arbre.
arb_droit))

liste_exemple = fois(plus(5,fois(3,5)) , 7)
```