```
'''
IPEIN
CHP 2 : POO
Exercice : 7
Groupe : sp3
'''
class IntervalleError(Exception):
   pass

class Intervalle:
   def __init__(self,binf, bsup):
       # appel au constructeur avec
       # i = Intervalle(1,2)
       # i.modif_binf(1.5)
       # i.__init__(1,2)
       # Intervalle.__init__(i,2,5)

       # vérifier que la cond est correcte,
       # sinon , elle génère une exception (arrêt du prog)
       """assert type(binf) in (int, float) \
               and 0 < binf, "Borne invalide"
       self.binf = binf

       if type(bsup) in (int, float) and binf < bsup :
           self.bsup = bsup
       else:
           # générer une exception : raise
           raise IntervalleError("Borne sup invalide")"""

       try:
           binf = float(binf)
           bsup = float(bsup)
           if not(0 < binf < bsup) :
               raise IntervalleError("bornes invalides: ne sont pas dans le bon ordre")
       except ValueError:
           raise IntervalleError("bornes invalides: non numériques")
       else:
           # Encapsulation : attr privé
           #               + méthode de modification
           #               + méthode de lecture
           # attribut privé : accesible seulement dans
           # la classe
           self.__binf = binf
           self.__bsup = bsup

   def modif_binf(self,valeur):
       if type(valeur) in [float,int] and \
           0 < valeur < self.__bsup :
           self.__binf = valeur
       else:
           raise IntervalleError("borne inf invalide")

   def lire_binf(self):
       return self.__binf

   def __str__(self):
       # appel avec
       # str(i) avec i un Intervalle
       # print(i) => print(str(i))
       # i.__str__()
       # Intervalle.__str__(i)
       return "Intervalle : binf = {}, bsup= {}"\
               .format(self.__binf, self.__bsup)

   def __contains__(self, valeur):
       # valeur in self
       assert type(valeur) in (int, float)
       return self.__binf <= valeur <= self.__bsup

   def __add__(self,other):
       # i3 = self + other
       assert isinstance(other,Intervalle)
       bi = self.__binf + other.__binf
       bs = self.__bsup + other.__bsup
       i = Intervalle(bi, bs)
       return i

   # __add__ => a + b
   # __and__ => a & b
   # __sub__ => a - b

# essayer de créer une instance de type Intervalle
try:
```

```
   i = Intervalle("0.8","5")
   i.modif_binf(4)
   print(4 in i)
   i3 = Intervalle(1,5) + Intervalle(1,5)
   print(str(i3))
except IntervalleError as e:
   print(e)

print("suite de prorgamme")
# i.binf = -2


# Python Exception Tree

#
class A(list):
   pass

A()
[]
L = A(range(5))
L
[0, 1, 2, 3, 4]
1 in L
True
1.5 in L
False
class A(list):
   def __contains__(self, val):
       return self[0] <= val <= self[-1]


L = A(range(5))
1.5 in L
True
L
[0, 1, 2, 3, 4]
class A(list):
   def __contains__(self, val):
       return self[0] <= val <= self[-1]
   def ajouter(self, val):
       self.append(val)


L = A(range(5))
L.ajouter(6)
L
[0, 1, 2, 3, 4, 6]
class A:
   def __init__(self):
       self.x=0


class B(A):
   def __init__(self):
       A.__init__(self)
       super().__init__()
       self.y = self.x+1
```