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

class Intervalle:
   # i1.__init__(1,2)
   #Intervalle.__init__(i1, 1, 2)
   #Intervalle(1, 2)
   def __init__(self, binf, bsup):
       try:
           binf = float(binf)
           bsup = float(bsup)
           if 0 < binf < bsup:
               self.__binf = binf
               self.__bsup = bsup
       except:
           raise IntervalleError("Bornes invalides")

       """
       assert type(binf) in [int, float] and binf > 0, \
                       "Borne inf invalide"
       self.__binf = binf #attribut privé : __binf

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

   def modif_binf(self, val):
       if type(val) in [int, float] and 0<val<self.__bsup:
           self.__binf = val
       else:
           raise IntervalleError("Nouvelle Borne inf invalide")
   def lire_binf(self):
       return self.__binf
   def __str__(self):
       # print(i1) => print(str(i1))
       # str(i1)
       # i1.__str__()
       # Intervalle.__str__(i1)
       return "Inetrvalle [binf : {}, bsup : {}]"\
               .format(self.__binf, self.__bsup)

   def __contains__(self, val):
       # val in self
       return self.__binf <=  val <= self.__bsup

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


   def __sub__(self, other):
       # i = self - other
       assert isinstance(other, Intervalle)
       bi = self.__binf - other.__bsup
       bs = self.__bsup - other.__binf
       return Intervalle(bi, bs)

   def __and__(self, other):
       # i = self & other
       assert isinstance(other, Intervalle)
       if self.__bsup < other.__binf or self.__binf > other.__bsup:
           return None
       else:
           pass
a = Intervalle(1, 5)
b = Intervalle(1, 5)
print(a+b)
"""
try: # essayer
   # de créer une instance
   i1 = Intervalle(1, 5)
   while 1:
       try:
            i1.modif_binf(int(input("val")))
        except:
            continue
        else:
            break

   print(i1.binf, i1.bsup)

#s'il y a une exception
except AssertionError as ex:
   print(ex)
except IntervalleError as e:
   print("Erreur: ", e)
"""
```