```python
'''
IPEIN
CHP 2 : POO
Exercice : 7
groupe : sp2
'''
class IntervalleError(Exception):
   pass

class Intervalle:
   def __init__(self, binf, bsup):
       # vérifier que cette cond est valide,
       # sinon générer une exception de type AssertionError
       """assert type(binf) in (int, float) and 0 < binf \
               ,"Borne inf incorrecte"

       self.binf = binf"""

       if type(binf) in (int, float) and \
          type(bsup) in [int, float] and \
           0 < binf < bsup :
           self.__bsup = bsup
           self.__binf = binf
       else:
           # générer (raise) une exception de type IntervalleError
           # python Exception Tree
           raise IntervalleError("Bornes invalide")

   def modif_binf(self, val):
       if type(val) in (int, float) and 0 < val < self.__bsup:
           self.__binf  = val
       else:
           raise IntervalleError("Borne inf invalide")

   def lire_binf(self):
       return self.__binf

   def __str__(self):
       # i = Intervamlle(1,5)
       # str(i)
       # print(i) => print(str(i))
       # i.__str__()
       # Intervalle.__str__(i)
       return "Intervalle : binf = {}, bsup = {}"\
               .format(self.__binf, self.__bsup)

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

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

# prog principal
try:  #Essayer
   # de créer une instance
   i = Intervalle(1,5)
   j = Intervalle(1,5)
   k = i + j
   print(k) # print(str(k)) <=> print(k.__str__())
   print(Intervalle(1,5)+Intervalle(1,5))
   print(i.__add__(j))

   # +  ---> __add__
   # -  --> __sub__
   # &  --> __and__  => k  = i & j

   #i.modif_binf(-5)
   i.modif_binf(3)
   bi = i.lire_binf()
   print(3 in i)
   #print(i.binf, i.bsup)
   #i.binf = -2 à interdire => binf doit être privé

except IntervalleError as e :
   print(e)
```

```
except : # pour toutes les autres exceptions
   print("Erreur inatendue")

print("passer à la suite de prog")
```