

```

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

class Intervalle:
    def __init__(self, binf, bsup):
        assert type(binf) in (int, float) and 0 < binf, "Bornes inf invalides"
        self.__binf = binf # __binf : attribut privé

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

    def modif_binf(self, val):
        if type(val) in (int, float) and 0 < val < self.__bsup:
            self.__binf = val

    def lire_binf(self):
        return self.__binf

    def __str__(self):
        # str(il)
        # print(il) ==> print(str(il))
        # il.__str__()
        # Intervalle.__str__(il)
        return "Intervalle : [binf: {}, bsup:{}]" \
            .format(self.__binf, self.__bsup)

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

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

    def __and__(self, other):
        # il = self & other
        assert isinstance(other, Intervalle)
        if self.__bsup > other.__binf:
            bi = max(self.__binf, other.__binf)
            bs = min(self.__bsup, other.__bsup)
            i = Intervalle(bi, bs)
        elif self.__binf < other.__binf and other.__bsup < self.__bsup:
            return other
        elif other.__binf < self.__binf and self.__bsup < other.__bsup:
            return self
        else:
            return None

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

a = Intervalle(1 , 6)
b = Intervalle(4 , 8)
print(a +b , a-b , a & b)

print(il)

'''
try:
    i = Intervalle(1 , 5)
    print("pas d'erreur")
except IntervalleError as e:
    print(e)
except AssertionError as e:
    print(e)
'''

```

```
except:
    print("Erreur")
"""
```