```
# ============================== #
# MP et PC-PC*                   #
# DS d'informatique du 2/12/2017 #
# Corrigé de l'exercice 1        #
# (Entiers de GAUSS)             #
# ============================== #


## La classe GaussInt

class GaussInt:
    """Classe permettant de manipuler des objets du type 'entiers de Gauss',
    c'est-à dire des complexes de la forme a+ib où a et b sont deux entiers
    relatifs.
    """
    def __init__(self,a,b):
        if type(a) != int or type(b) != int:
            raise TypeError("Vous devez fournir des entiers en arguments !")
        else:
            self.re = a
            self.im = b

    def __repr__(self):
        return("("+str(self.re)+","+str(self.im)+")")

    def norme(self):
        """méthode renvoyant la norme de l'entier de Gauss considéré.
        """
        return(self.re**2 + self.im**2)

    def __neg__(self):
        """surcharge de l'opérateur __neg__ pour obtenir l'opposé d'un
entier
        de Gauss.
        """
        return(GaussInt(-self.re,-self.im))

    def __add__(self,n):
        """surcharge de l'opérateur __add__ pour obtenir, à l'aide de +, la
        somme d'un entier de Gauss et d'un entier ou d'un autre entier de
Gauss.
        """
        if type(n) != GaussInt and type(n) != int:
            raise TypeError("Vous ne pouvez ajouter à un entier de Gauss
qu'un entier ou un autre entier de Gauss !")
        elif type(n) == GaussInt:
            return(GaussInt(self.re + n.re,self.im + n.im))
        else:
            return(GaussInt(self.re + n,self.im))

    def __radd__(self,n):
        """surcharge de l'opérateur __radd__ pour obtenir, à l'aide de +, la
        somme d'un entier et d'un entier de Gauss.
        """
        if type(n) != int:
            raise TypeError("Vous ne pouvez ajouter à un entier de Gauss
qu'un entier ou un autre entier de Gauss !")
        else:
            return(self + n)
```

```
    def __sub__(self,n):
        """surcharge de l'opérateur __sub__ pour obtenir, à l'aide de -, la
        différence d'un entier de Gauss et d'un entier ou d'un autre entier
de Gauss.
        """
        if type(n) != GaussInt and type(n) != int:
            raise TypeError("Vous ne pouvez soustraire à un entier de Gauss
qu'un entier ou un autre entier de Gauss !")
        else:
            return(self + (-n))

    def __rsub__(self,n):
        """surcharge de l'opérateur __radd__ pour obtenir, à l'aide de -, la
        différence d'un entier et d'un entier de Gauss.
        """
        if type(n) != int:
            raise TypeError("Vous ne pouvez soustraire un entier de Gauss
qu'à un entier ou un autre entier de Gauss !")
        else:
            return(-self + n)

    def __mul__(self,n):
        """surcharge de l'opérateur __mul__ pour obtenir, à l'aide de *, le
        produit d'un entier de Gauss par un entier ou un autre entier de
Gauss.
        """
        if type(n) != GaussInt and type(n) != int:
            raise TypeError("Vous ne pouvez multiplier un entier de Gauss
que par un entier ou un autre entier de Gauss !")
        elif type(n) == GaussInt:
            return(GaussInt(self.re * n.re - self.im * n.im,self.re * n.im +
self.im * n.re))
        else:
            return(GaussInt(self.re * n,self.im * n))

    def __rmul__(self,n):
        """surcharge de l'opérateur __rmul__ pour obtenir, à l'aide de *, le
        produit d'un entier et d'un entier de Gauss.
        """
        if type(n) != int:
            raise TypeError("Vous ne pouvez multiplier un entier de Gauss
que par un entier ou un autre entier de Gauss !")
        else:
            return(self * n)

    def __floordiv__(self,n):
        """surcharge de l'opérateur __floordiv__ pour obtenir, à l'aide de
//,
        le(s) quotient(s) de la division euclidienne d'un entier de Gauss
par un
        entier ou un autre entier de Gauss.
        """
        # 1er cas : le diviseur n'est ni un entier ni un entier de Gauss
        if type(n) != GaussInt and type(n) != int:
            raise TypeError("Vous ne pouvez diviser un entier de Gauss que
par un entier ou un autre entier de Gauss !")
        # 2ème cas : le diviseur est un entier. La division est possible si
le
        # diviseur est non nul. On le convertit alors en un entier de Gauss
        # se ramener à la division d'un entier de Gauss par un autre entier
```

```
        # de Gauss
        elif type(n) == int:
            if n == 0:
                raise ValueError("Division par 0 !")
            else:
                return(self//GaussInt(n,0))
        # 3ème cas : le diviseur est un entier de Gauss.
        else:
            if self.re == 0 and self.im == 0:
                raise ValueError("Division par 0 !")
            else:
                # importation de floor (fonction partie entière)
                from math import floor
                # norme du diviseur
                nor = n.norme()
                # alpha et beta
                a = floor((self.re * n.re + self.im * n.im)/nor)
                b = floor((self.im * n.re - self.re * n.im)/nor)
                # tuple des quotients possibles
                L = (GaussInt(a,b),
GaussInt(a+1,b),GaussInt(a,b+1),GaussInt(a+1,b+1))
                # liste des quotients
                q = []
                for e in L:
                    if (self - e*n).norme() < nor:
                        q.append((e.re,e.im))
                # renvoi du résultat
                return(q)

## Pour tester...

a = GaussInt(27,-15)
b = GaussInt(-3,-4)
print(a//b)

b = 4
print(a//b)
b = 15
print(a//b)
```