```
#################
## EXAMPLE: simple Coordinate class
#################
class
Coordinate(object):
    """ A coordinate made up of an x and y value
"""
    def __init__(self, x, y):
        """ Sets the x and y
values """
        self.x = x
        self.y = y
    def __str__(self):

""" Returns a string representation of self """
        return
"<" + str(self.x) + "," + str(self.y) + ">"
    def
distance(self, other):
        """ Returns the euclidean distance between two
points """
        x_diff_sq = (self.x-other.x)**2
        y_diff_sq =
(self.y-other.y)**2
        return (x_diff_sq + y_diff_sq)**0.5


c = Coordinate(3,4)
origin =
Coordinate(0,0)
print(c.x, origin.x)
print(c.distance(origin))
print(Coordinate.distance(c,
origin))
print(origin.distance(c))
print(c)


#################
## EXAMPLE: simple class to
represent fractions
## Try adding more built-in operations like multiply, divide
### Try adding
a reduce method to reduce the fraction (use gcd)
#################
class Fraction(object):

"""
    A number represented as a fraction
    """
    def
__init__(self, num, denom):
        """ num and denom are integers
"""
        assert type(num) == int and type(denom) == int, "ints not
used"
        self.num = num
        self.denom = denom
    def __str__(self):

""" Retunrs a string representation of self """
        return
str(self.num) + "/" + str(self.denom)
    def __add__(self, other):

""" Returns a new fraction representing the addition """

top = self.num*other.denom + self.denom*other.num
        bott = self.denom*other.denom

return Fraction(top, bott)
    def __sub__(self, other):
        """ Returns a
new fraction representing the subtraction """
```

```
        top = self.num*other.denom
- self.denom*other.num
        bott = self.denom*other.denom
        return Fraction(top,
bott)
    def __float__(self):
        """ Returns a float value of the fraction
"""
        return self.num/self.denom
    def inverse(self):

""" Returns a new fraction representing 1/self """
        return
Fraction(self.denom, self.num)

a = Fraction(1,4)
b = Fraction(3,4)
c = a + b # c is a Fraction
object
print(c)
print(float(c))
print(Fraction.__float__(c))
print(float(b.inverse()))
##c =
Fraction(3.14, 2.7) # assertion error
##print a*b # error, did not define how to multiply two
Fraction objects


###############
## EXAMPLE: a set of integers as class
###############
class
intSet(object):
    """
    An intSet is a set of integers
    The value is
represented by a list of ints, self.vals
    Each int in the set occurs in self.vals exactly
once
    """
    def __init__(self):
        """ Create an empty
set of integers """
        self.vals = []

    def insert(self, e):

""" Assumes e is an integer and inserts e into self """

if not e in self.vals:
            self.vals.append(e)

    def member(self, e):

""" Assumes e is an integer
        Returns True if e is in self, and False
otherwise """
        return e in self.vals

    def remove(self, e):

""" Assumes e is an integer and removes e from self
        Raises ValueError if
e is not in self """
        try:
            self.vals.remove(e)

except:
            raise ValueError(str(e) + ' not found')

    def __str__(self):

""" Returns a string representation of self """
```

```
self.vals.sort()
        return '{' + ','.join([str(e) for e in self.vals]) + '}'


s =
intSet()
print(s)
s.insert(3)
s.insert(4)
s.insert(3)
print(s)
s.member(3)
s.member(5)
s.insert
(6)
print(s)
#s.remove(3)  # leads to an error
print(s)
s.remove(3)
```