Problem 1. (*Geo Location*) Define a data type `Location` in `location.py` that represents a location on Earth using its latitude and longitude. The data type must support the following API:

method	description
`Location(lat, lon)`	a new location l from the given latitude and longitude values
`l.distanceTo(m)`	the great-circle distance† between l and m
`str(l)`	the string representation of l as `'(lat, lon)'`

† See Problem 7 from Project 2 for the great-circle distance formula.

```
$ python location.py 48.87 -2.33 37.8 -122.4
loc1 = (48.87, -2.33)
loc2 = (37.8, -122.4)
d(loc1, loc2) = 8701.38954324
```

Problem 2. (*2D Point*) Define a data type `Point` in `point.py` that represents a point in 2D. The data type must support the following API:

method	description
`Point(x, y)`	a new point p from the given x and y values
`p.distanceTo(q)`	the Euclidean distance between p and q
`str(p)`	the string representation of p as `'(x, y)'`

```
$ python point.py 0 1 1 0
p1 = (0.0, 1.0)
p2 = (1.0, 0.0)
d(p1, p2) = 1.41421356237
```

Problem 3. (*1D Interval*) Define a data type `Interval` in `interval.py` that represents a closed 1D interval. The data type must support the following API:

method	description
`Interval(lbound, rbound)`	a new interval i from the given lower and upper bounds for the interval
`i.lbound()`	lower bound of i
`i.ubound()`	upper bound of i
`i.contains(x)`	does i contain the point x?
`i.intersects(j)`	does i intersect the interval j?
`str(i)`	the string representation of i as `'[lbound, rbound]'`

```
$ python interval.py 3.14
0 1 0.5 1.5 1 2 1.5 2.5 2.5 3.5 3 4
<ctrl-d>
[2.5, 3.5] contains 3.140000
[3.0, 4.0] contains 3.140000
[0.0, 1.0] intersects [0.5, 1.5]
[0.0, 1.0] intersects [1.0, 2.0]
[0.5, 1.5] intersects [1.0, 2.0]
[0.5, 1.5] intersects [1.5, 2.5]
[1.0, 2.0] intersects [1.5, 2.5]
[1.5, 2.5] intersects [2.5, 3.5]
[2.5, 3.5] intersects [3.0, 4.0]
```

Problem 4. (*Rectangle*) Define a data type `Rectangle` in `rectangle.py` that represents a rectangle using 1D intervals (ie, `Interval` objects) to represent its x (width) and y (height) segments. The data type must support the following API:

method	description
`Rectangle(xint, yint)`	a new rectangle r from the given x and y segments (as interval objects)
`r.area()`	the area of r
`r.perimeter()`	the perimeter of r
`r.contains(x, y)`	does r contain the point (x, y)?
`r.intersects(s)`	does r intersect the rectangle s?
`str(r)`	the string representation of r as `'[x1, x2] x [y1, y2]'`

```
$ python rectangle.py 1.01 1.34
0 1 0 1 0.7 1.2 .9 1.5
<ctrl-d>
0 1 0 1 0.7 1.2 .9 1.5
Area([0.0, 1.0] x [0.0, 1.0]) = 1.000000
Perimeter([0.0, 1.0] x [0.0, 1.0]) = 4.000000
Area([0.7, 1.2] x [0.9, 1.5]) = 0.300000
Perimeter([0.7, 1.2] x [0.9, 1.5]) = 2.200000
[0.7, 1.2] x [0.9, 1.5] contains (1.010000, 1.340000)
[0.0, 1.0] x [0.0, 1.0] intersects [0.7, 1.2] x [0.9, 1.5]
```

Problem 5. (*Rational Number*) Define a data type `rational` in `rational.py` that represents a rational number, ie, a number of the form a/b where a and $b \neq 0$ are integers. The data type must support the following API:

method	description
`Rational(x, y)`	a new rational r from the numerator x and denominator y
`r + s`	sum of r and s
`r - s`	difference of r and s
`r * s`	product of r and s
`abs(r)`	absolute value of r
`str(r)`	the string representation of r as `'x/y'`

Use the private function `_gcd()` to ensure that the numerator and denominator never have any common factors. For example, the rational number $2/4$ must be represented as $1/2$.

```
$ python rational.py 100
3.13159290356
```

Files to Submit

1. `location.py`
2. `point.py`
3. `interval.py`
4. `rectangle.py`
5. `rational.py`

Before you submit:

- Make sure your programs meet the input and output specifications by running the following command on the terminal:

```
$ python run_tests.py -v [<problems>]
```

  where the optional argument `<problems>` lists the problems (`Problem1`, `Problem2`, etc.) you want to test; all the problems are tested if no argument is given.

- Make sure your programs meet the style requirements by running the following command on the terminal:

```
$ pep8 <program>
```

  where `<program>` is the `.py` file whose style you want to check.