Assignment 3

This assignment is similar to the last one, but here, the language of choice is Python. Specifically, there is some code being given to you for the classes being used. These are:

1. `two_d_point.py`
2. `quadrilateral.py`
3. `rectangle.py`
4. `square.py`

Additionally, there are also some test class skeletons also provided (`test_two_d_point.py` and `test_quadrilateral.py`).

Your tasks are as follows:

1. Complete the codebase for `TwoDPoint`, `Quarilateral`, `Rectangle`, and `Square`. The incomplete portions of the code are marked with *TODO* and the methods are provided with documentation explaining what each incomplete method should do. There may also be a few minor bugs in the provided code. It is a part of this assignment to fix these (you are encouraged to use the debugger and discover these bugs by use of unit tests, if necessary).
   - TwoDPoint (10 points)
   - Quadrilateral (10 points)
   - Rectangle (10 points)
   - Square (10 points)

2. Write a class called `ShapeSorter` (the file name must be `sorter.py`) with a method called `sort`. This method should take as its argument a variable number of quadrilaterals (i.e., `*args`) and return a list of these shapes sorted in increasing order of their smallest x-coordinate. You must also decide if this method should be an instance, static, or class method. (15 points)
3. Every shape (i.e., quadrilateral, rectangle, and square) must also use the appropriate magic methods to check for equality and to have a human-readable string representation of an instance of the class. These *must* be the standard usage of Python's magic methods, and NOT something else. (3 shapes, 5 points each = 15 points)
4. Complete the unit tests provided in the classes `TestTwoDPoint` and `TestQuadrilateral`. (10 points)
5. Write unit test classes `TestRectangle` and `TestSquare`. In these classes, you must write unit test methods for Rectangle's `center()` and `area()` methods, and Square's `snap()` method. (20 points)

The code base is available here.

For this assignment, you may assume that all shapes have sides that are axis-aligned. For example, rectangles and squares will have sides that are parallel to the x and y axes. This does not, of course, make sense for quadrilaterals in general.

A note on Python syntax of importing modules

Import statements in Python can be absolute or relative (recall that the 'dot'-based syntax in import statements are translated to paths. So, for two files in the same package or folder, you may use either option. For example, the import statement in `quadrilateral.py` is

```
from .two_d_point import TwoDPoint
```

but it could have also been written explicitly as

```
from just_a_folder.two_d_point import TwoDPoint
```

Both are semantically equivalent.