CSC148, Winter 2017
build Point class

Somewhere in the real world there is a description of points in two-dimensional space:

> In two dimensions, a point is two numbers (coordinates) that are treated collectively as a single object. Points are often written in parentheses with a comma separating the coordinates. For example, (0, 0) represents the origin, and (x, y) represents the point x units to the right and y units up from the origin. Some of the typical operations that one associates with points might be calculating the distance of a point from the origin, or from another point, or finding a midpoint of two points, or asking if a point falls within a given rectangle or circle.

Find the most important noun (good candidate for a class. . .), its most important attributes, and operations that sort of noun should support.

On the back of this sheet are steps taken from the recipe for designing Python classes. You'll be using this in more depth next week during lab, but for now we'll use it as a guide to creating the Point class. Here are exercises to carry out on paper, perhaps you'll need some scrap paper as well as this hand-out. You are, of course, welcome, to continue the exercise on a computer in Python.

1. Carry out steps 1 and 2 of Part 1 for the description of points above.
2. Carry out steps 3 and 4 of Part 1 for the description of points above.
3. Carry out step 3 of Part 2 for the description of points above.

Part 1: Define the API for the class

1. **Class name and description.** Pick a noun to be the name of the class, write a one-line summary of what that class represents, and (optionally) write a longer, more detailed description.
2. **Example.** Write some simple examples of client code that uses your class.
3. **Public methods.** Decide what services your class should provide. For each, define the API for a method that will provide the action. Use the first four steps of the Function Design Recipe to do so:
   (1) Example
   (2) Type Contract
   (3) Header
   (4) Description
4. **Public attributes.** Decide what data you would like client code to be able to access without calling a method. Add a section to your class docstring after the longer description, specifying the **type**, **name**, and **description** of each of these attributes.

Part 2: Implement the class

1. **Internal attributes.** Define the **type**, **name**, and **description** of each of the internal attributes (if there are any). Put this in a class comment (using the hash symbol) below the class docstring.
2. **Representation invariants.** Add a section to your internal class comment containing any representation invariants.
3. **Public methods.** Use the last two steps of the function design recipe to implement all of the methods:
   (5) Code the Body
   (6) Test your Method