CSC148 Summer 2016

Rectangle class

(do it before looking at any sample solution)

Rectangle can be defined in many different ways. Somewhere in the real world, there is a description of rectangle as follows:

> *A rectangle is defined by its top-left coordinates as well as its width and height. A rectangle is usually represented by a quadruple (x, y, w, h) where x and y represent the top-left coordinate, w represents the width, and h represents the height. For example, (10, 20, 300, 400) represents a rectangle that its top-left coordinate is located at point (10,20), its width is 300 and its height is 400. Some of the typical operations that one associates with rectangles might be translating the rectangle to the right, left, up, and down, or if two rectangles are conceptually equal, which means have same coordinate and size, or asking if a rectangle falls within another rectangle, or asking if a rectangle has overlap with another rectangle, etc.*

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 Rectangle class. Here are exercises to carry out on paper, perhaps you'll need some scrap paper as well as this hand-out. After you are done on the paper, of course, welcome to continue the exercise in PyCharm.

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

Class Design Recipe

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) Header
   (2) Type Contract
   (3) Example
   (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