ASSIGNMENT 5 (last)

Read the instructions below carefully. The instructions must be followed. This assignment is worth 5% of your grade. The assignment is due on Monday 7th of December 8AM. No late assignments will be accepted.

The goal of this assignment is to learn and practice (via programming, quizzes and algorithm analysis) the concepts that we have studies in the last couple of weeks.

This assignment has 3 parts. Each part explains what needs to be submitted. Put all those required documents into a folder called a5_xxxxxx. Zip that folder and submit the zip file as explained in Lab 1. In particular, the folder should have the following 6 files:

a5_week3_xxxxxx.jpg
a5_week4_xxxxxx.jpg } Part 1

and

a3_part2_xxxxxx.py
a3_part2_xxxxxx.txt } Part 2

and

a3_part3_xxxxxx.py
a3_part3_testing_xxxxxx.txt } Part 3

For every method that you design in part 3 of this assignment you have to include the type contract (inside a docstring -- the rest of the usual info that goes into the docstring can be omitted for this assignment)

****
PART 1 (5 points)
****
You can think of Part 1 as a practice for your exam.

If you have difficulties with any of the two tests below, I suggest that you watch all the short coursera videos that relate to the given week. In general that is a good idea.

For Part 1 of the assignment 5 do the following.

Go to https://www.coursera.org/course/programming2

(Notice the above is the 2nd coursera course.)

Click on Exercises (on left hand side).
Complete Week 3 and Week 4 exercises (by clicking Attempt Homework, like you did in Assignment 3). Then follow the instructions from Part 1 of Assignment 3 to submit the two images that demonstrated that you completed the two quizzes.

a5_week3_xxxxxx.jpg
a5_week4_xxxxxx.jpg

Make sure that each of the following is visible in each of your images:
1. that your name is present on the top right corner
2. that the Week number is visible
3. that the score for the last attempt is visible for that week
4. NEW (top of the page saying: "Learn to Program: Crafting Quality Code

by Jennifer Campbell, Paul Gries" so that it clear the two quizzes you completed are from 2nd coursera course and not the 1st one)

Important: Each of your photos cannot be more than 200 KB (KB not MB!! we have limited space on the Blackboard Learn server). For me the screen capture produced a photo of about 100KB.

Each of the tests is worth 2.5 points. You will get full 5 points for Part 1 of the assignment, if both of your tests have 75% or more, otherwise the actual weighted average will be used.

****
PART 2 (25 points)
****
For this part, you will be solving some problems and implementing the solutions. The idea is to design and implement solutions that are as fast as possible, and to analyze these solutions as well. You will submit two files:

a3_part2_xxxxxx.py contains the functions you are asked to write for each question.
a3_part2_xxxxxx.txt contains, for each function, a rough analysis of its running time when the argument, n, or the length of the list, a, is 10,000. For example, if your function looks like this:

```
def riddle(a):
        s=0
        for x in a:
            for y in a:
                s = s+ x*y
        return s
```

Then you would write: This function has two nested for loops. The inner loop executes 10,000 times for each step of the outer loop, which also executes 10,000 times. Therefore, this function performs roughly 10,000*10,000=100,000,000 operations. If your function uses the sort() method to sort a, just say that this uses about about 140,000 operations (since sort uses roughly n log_2 n operations).

In all of the questions you can assume that a is a list containing numbers.

2a) (5 points) Write a function, largest_two(a), that returns the sum of the two largest values in the list a.

2b) (5 points) Write a function, smallest_half(a), that computes the sum of the len(a)//2 smallest values in the list a.

2c) (5 points) Write a function, median(a), that returns a value, x, such that at least half the elements in a are less than or equal to x and at least half the elements in a are greater than or equal to x.

2d) (5 points) Write a function, at_least_third(a), that returns a value in a that occurs at least len(a)//3 + 1 times. If no such element exists in a, then this function returns None.

2e) (5 points) Write a function, triple_sum(a,x), that takes a list ,a, as input and returns True if there exists i, j and k (where i and j and k are not necessarily distinct) such that a[i]+a[j]+a[k]=x. Otherwise it returns False. For example, if a=[1, 5, 8, 2, 6, 55, 90] and x=103, then triple_sum(a, x) would return True since a[1]+a[2]+a[6]=5+8+90=103. If a=[-1, 1, 5, 8, 2, 6] and x=-3, triple_sum(a, x) would

return True since a[0]+a[0]+a[0]=-1+ -1 + -1 = -3. If a=[-10,2] and x=-18, triple_sum(a, x) would return True since a[0]+a[0]+a[1]=-10+-10+2=18. If a=[1, 1, 5, 8, 2, 6] and x=1000 would return False, since there are not indices i, j and k such that a[i]+a[j]+a[k]=1000.

****
PART 3 (40 points)
****

For this part, I provided two files, called a5_part3_xxxxxx.py and a3_part3_testing_given.txt

File a5_part2_xxxxxx.py already contains a class Point that we developed in class. For this part, you will need to develop and add two more classes to a5_part2_xxxxxx.py: class Rectangle and class Canvas.

To understand how they should be designed and how they should behave, you must study in detail the test cases provided in a3_part3_testing_given.txt. These tests are your main resource in understanding what methods your two classes should have and what their input parameters are. I will explain few methods below in detail, but only those whose behaviour may not be clear from the test cases.

As in Assignment 1 and Assignment 2, for part 3 of this assignment you will need to also submit your own text file called a3_part3_testing_xxxxxx.txt demonstrating that you tested your two classes and their methods (in particular, demonstrating that you tested them by running all the calls made in a3_part3_testing_given.txt)

Details about the two classes:
=====

Class Rectangle represents a 2D (axis-parallel) rectangle that a user can draw on a computer screen. Think of a computer screen as a 2D map where each position has an x and a y coordinate.

The data that each object of type Rectangle should have (and that should be populated in the constructor, i.e., __init__ method of the class Rectangle) are:
* two Points: the first point representing the bottom left corner of the rectangle and the second representing the top right corner of the rectangle; and,
* the color of the rectangle

Note that the two points (bottom left and top right) completely determine (the axis parallel) rectangle and its position on the map. There is no default rectangle.

(see page 5 & 6 for some helpful illustrations)

The __init__ method of Rectangle (that is invoked by the constructor Rectangle) will take two objects of class Point as input and a string for the color). You may assume that the first point (sent to the constructor, i.e. __init__) will always have smaller than or equal x coordinate than the x coordinate of the second point and smaller than or equal y coordinate than the y coordinate of the second point.

*****

Class Rectangle should have 13 methods. In particular, in addition to the constructor (i.e. __init__ method) and three methods that override python's object methods (and make

your class user friendly as suggested by the test cases), your class should contain the following 9 methods:
get_bottom_left, get_top_right, get_color, reset_color, get_perimeter, get_area, move, intersects, and contains.

Here is a description of three of those methods whose job may not be obvious from the test cases.

* Method move: given numbers dx and dy this method moves the calling rectangle by dx in the x direction and by dy in the y-direction. This method should not change directly the coordinates of the two corners of the calling rectangle, but must instead call move method from the Point class.

* Method intersects: returns True if the calling rectangle intersects the given rectangle and False otherwise. Definition: two rectangles intersect if they have at least one point in common, otherwise they do not intersect.

* Method contains: given an x and a y coordinate of a point, this method tests if that point is inside of the calling rectangle. If yes it returns True and otherwise False. (A point on the boundary of the rectangle is considered to be inside).

****

Class Canvas represents a collection of Rectangles. It has 8 methods. In addition, to the constructor (i.e. __init__ method) and two methods that override python's object methods (and make your class user friendly as suggested by the test cases), your class should contain 5 more methods:
add_one_rectangle, count_same_color, total_perimeter, min_enclosing_rectangle, and common_point.

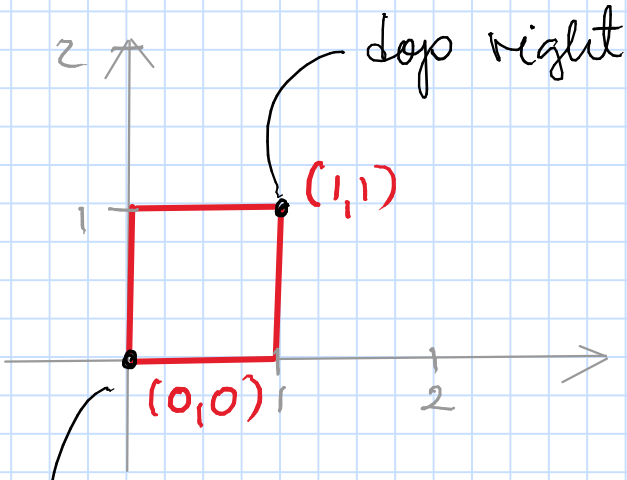
Here is a description of those methods whose job may not be obvious from the test cases.

* The method total_perimeter: returns the sum of the perimeters of all the rectangles in the calling canvas. To compute total perimeter do not compute a perimeter of an individual rectangle in the body of total_perimeter method. Instead use get_perimeter method from the Rectangle class.

* Method min_enclosing_rectangle: calculates the minimum enclosing rectangle that contains all the rectangles in the calling canvas. It returns an object of type Rectangle of any colour you prefer. To find minimum enclosing rectangle you will need to find the minimum x coordinate of all rectangles, the maximum x coordinate for all rectangles, the minimum y coordinate and the maximum y coordinate of all rectangles.

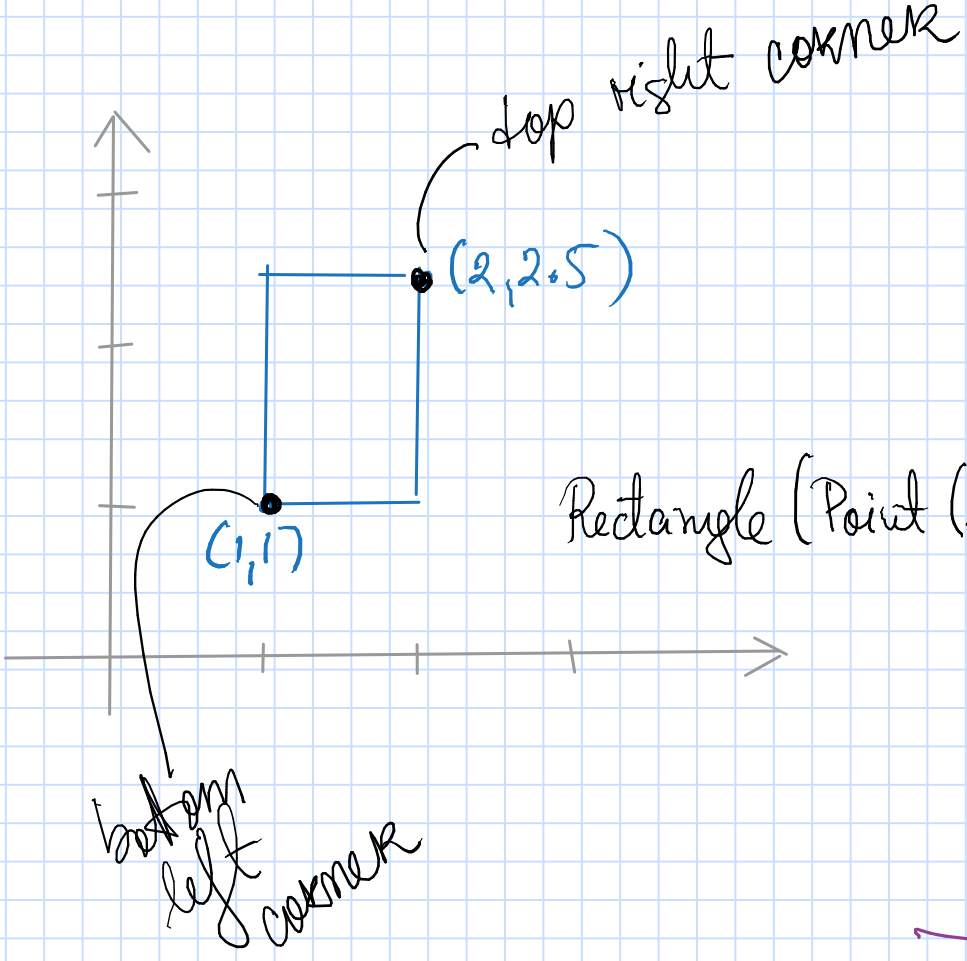
* Method common_point: returns True if there exists a point that intersects all rectangles in the calling canvas. To test this (for axis parallel rectangles like ours), it is enough to test if every pair of rectangles intersects (according to a Hellyâ€™s theorem http://en.wikipedia.org/wiki/Helly's_theorem).

***
Finally recall, from the beginning of the description of this assignment that each of your methods should have a type contract.



Rectangle(Point(), Point(1,1), "red")

↓ bottom left corner

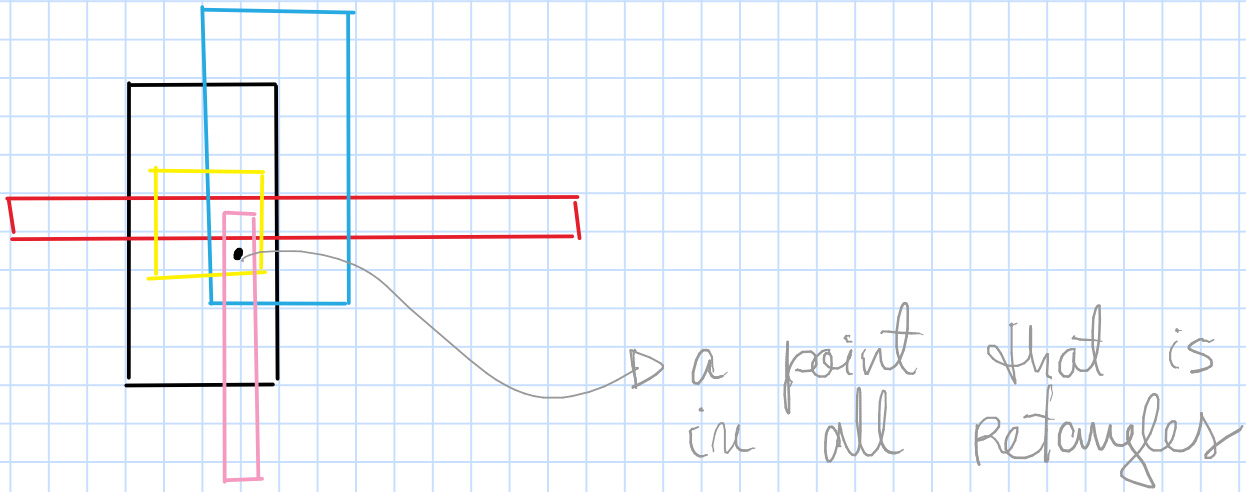


Rectangle(Point(1,1), Point(2,2.5), "blue")

turn → Purge

examples for common-point method

an example of a collection of rectangles that does have a point in common



an example of a collection of rectangles that does not have a point in common