ASSIGNMENT 5

Read the instructions below carefully. The instructions must be followed. This assignment is worth 5% of your grade. The assignment is due at the end of the last day of the semester, specifically on Wednesday 6th of December at 11:30pm. No late assignments will be accepted.

The goal of this assignment is to learn and practice (via programming) the following concepts that we have learnt thus far: efficient algorithms and their analysis, design of custom made classes and their use in making objects and finally recursion (a way to solve problems that we will learn in the next couple of classes).

This assignment has three parts (worth 100 points) plus a bonus (worth extra 20 percent).

[Brief blurb about the bonuses in general: if it was unclear before, if you get more than 100, the extra is transferred to the rest of your class grade. For example, if you were to get the (maximum possible) 120 points on Assignment 5, the extra 20% out of 5% (of the worth of the assignment) is worth 1% of your course grade. In that case, whatever course percentage you get at the end, would be increased by 1%.]

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 5 files:

a5_part1_xxxxxx.py
a5_part1_xxxxxx.txt

ands

a5_part2_xxxxxx.py
a5_part2_testing_xxxxxx.txt

and

a5_part3_xxxxxx.py

For every function and method that you design in this assignment you must include the type contract inside a docstring (the rest of the usual info that goes into the docstring, like function description and preconditions, can be omitted for this assignment)

For bonus you need to submit the file:
a5_bonus_xxxxxx.py

In each of the above files xxxxxx needs to be replaces by your student number.

As always, your programs must run without syntax errors. Review previous assignments rules for details on that.

No global variables are allowed in functions: only use variables that are in the function's namespace. That is, only use variables that are parameters or are otherwise added to the function's namespace such as through assignment or iteration

ASSIGNMENT 5

****
PART 1 (20 points)
****

For this part, you will be solving some problems and implementing the solutions. For this part, your goal is to design and implement solutions that are as fast as possible, and to analyze these solutions as well. You will submit two files:

a5_part1_xxxxxx.py needs to contain the functions you are asked to write for each question.
a5_part1_xxxxxx.txt needs to contain, for each function, a rough analysis of its running time in terms of n (where n is the length of the input list a, i.e. n=len(a)) and where n=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 Python's `sort()` method or Python's `sorted()` function to sort the list a, just say that that function call does about 140,000 operations (since Python's sort does roughly n log_2 n operations).

Alternatively, if you prefer, you can write your analysis in terms of general n. Thus for the above function riddle, one would say that its running time is O(n^2).

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

1a) (5 points) Write a function, largest_34(a), that returns the sum of the 3rd and 4th largest values in the list a. For simplicity, you may assume that the numbers in the list a are all distinct and that the list a has at least 4 elements. For example:

```
>>> largest_34([1000, 1, 100, 2, 99, 200, -100])
199
```

1b) (5 points) Write a function, largest_third(a), that computes the sum of the `len(a)//3` of the largest values in the list a. For simplicity, you may assume that the numbers in the list a are all distinct and that the list a has at least 3 elements.

1c) (5 points) Write a function, third_at_least(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`. For simplicity, you may assume that the numbers in the list a are all distinct and that the list a has at least 4 elements.

1d) (5 points) Write a function, sum_tri(a,x), that takes a list , a, as input and returns `True` if there exists indices 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 sum_tri(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, sum_tri(a, x) would return True since a[0]+a[0]+a[0]=-1+ -1 + -1 = -3. If a=[-10,2] and x=-18, sum_tri(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.

ASSIGNMENT 5

****
PART 2 (65 points)
****

For this part, you are provided with 3 files: a5_part2_xxxxxx.py, a5_part2_testing_given.txt and drawings_part2.pdf

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_part2_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 2 of this assignment you will need to also submit your own text file called a5_part2_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 a5_part2_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 plane where each position has an x and a y coordinate.

The data (i.e. attributes) that each object of type Rectangle should have (and that should be initialized 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 in the plane. There is no default rectangle.

(see drawings_part2.pdf file for some helpful illustrations)

The __init__ method of Rectangle (that is called by the constructor Rectangle) will take two objects of type Point as input and a string for the color). You may assume that the first Point (passed 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 color 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 Helly's theorem for axis-aligned rectangles: 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.

****
PART 3 (15 points)
****

Implement a recursive python function digit_sum(n) to calculate the sum of all digits of the given non-negative integer n.

Then, implement a recursive python function to compute the digital root digital_root(n) of the given integer n. Your function must use the digit_sum function.

The *digital root* of a number is calculated by taking the sum of all of the digits in a number, and repeating the process with the resulting sum until only a single digit remains. For example, if you start with 1969, you must first add 1+9+6+9 to get 25. Since the value 25 has more than a single digit, you must repeat the operation to obtain 7 as a final answer.

Your function digital_root(n) must use digit_sum function. Place both functions in the same file a5_part3_xxxxxx.py.

Note: You can implement iterative versions of the two functions for yourself, but submit the recursive versions only. Your recursive functions must not use loops.

ASSIGNMENT 5

****
BONUS (20 points)
****

To get the bonus or parts of it, you will need to redo some parts of Assignment 4 but by designing and using objects with a small change that when creating a network, you will need to read 2 files, one as in Assignment 4 (see the included files net1.txt and net3.txt), and one containing user IDs and their names (see the included files net1_with_names.txt and net3_with_names.txt). As before you have a starter code in a5_bonus_xxxxxx.py. Your code must go inside of that file in clearly indicated spaces. As always you cannot erase of modify any parts of the given code except for word "pass". IMPORTANT: all of your code must go inside of the class Person or class Network and must behave as implied by program runs and functions calls given in a5_bonus_test_runs.txt

Do not be surprised that a5_bonus_xxxxxx.py will crash once you enter the (correct) names of the 2 files. That is because only after you code correctly the whole `class Person` and `class Network` that the `main` will work.

Since one of the main goals of this assignment is to learn about object oriented design, for algorithmic part you may use some of the coding ideas you get from my solution to Assignment 4. But if you use and modify any of my code, you must indicate that in the comments of your code.

For the bonus part, you only need to write type-contract in docstrings. Function descriptions are not necessary, since they are, de facto, the same as in Assignment 4. Just as one example, recall that the users in an object Network need to be sorted by their ID, and each list of friends need to be sorted too.