OBJECT ORIENTED PROGRAMMING IN PYTHON

Included with the assignment is a script for testing your solution called `assignment2_tests.py`. This script will test the output from your code against a number of test cases and will indicate if there are errors. Once you have written your code in `assignment2.py`, you can run these tests by executing:

```
python assignment2_tests.py
```

Be sure that you can run `assignment2_tests.py` before submitting as this is how we will test your code for grading! The provided test cases are meant to help you debug your code, but you should not assume that they are exhaustive. If a problem asks you define a function or class, **you should use exactly the name specified** in the problem for this function or class. Your modified version of `assignment2.py` should be submitted to Moodle by the due date specified above.

Problem 1: Classes (50 points)

In this exercise you should write a python class called `Course`. The class member functions are described below.

Course: The `Course` class should have the following methods:

- `__init__` takes and stores a single integer course number.
- `get_course_number` takes no arguments and returns the integer course number.
- `add_student` takes a single `string` student name and adds it in a roster.
- `drop_student` takes a single `string` student name. If the student is currently in the roster, it removes that student name from the roster and returns nothing.
- `get_roster` takes no arguments and returns a list containing the names of all students enrolled in the course in **alphabetical order**.

You may user whatever instance variables and write as many other member functions as you want, as long as the listed functions work as described.

Problem 2: Finding Roots (50 points)

You should write a function called `find_roots` that solves quadratic equations of the form:

$$ax^2 + bx + c = 0 \tag{1}$$

Your function should take as arguments the three coefficients of the quadratic equation, a, b, and c in that order and return a list containing all **real** solutions to the equation **in ascending order**. Complex solutions should be ignored. For example, the call `find_roots(1, 7, 12)` should return the set `[-4.0, -3.0]`. If the equation has only a single real solution or no real solutions, `find_roots` should return a singleton list or an empty list respectively.

You may use any method for solving quadratic equations, however, you should try to make your solution is as stable as possible. `assignment2_tests.py` provides a number of test cases where root finding methods may have problems, but this list is intentionally not exhaustive and different methods will have different problem areas. Think carefully about the method you choose and where numerical errors such as underflow, overflow, and round-off may occur and code your solution to handle these scenarios. **Note: You may not use any root finding methods from python modules.**