Assignment Description:

Sometimes you will be given a program that someone else has written, and you will be asked to fix, update and enhance that program. In this assignment you will start with an existing implementation of the classify triangle program that will be given to you. You will also be given a starter test program that tests the classify triangle program, but those tests are not complete. In order to determine if the program is correctly implemented, you will need to update the set of test cases in the test program. You will need to update the test program until you feel that your tests adequately test all of the conditions. Then you should run the complete set of tests against the original triangle program to see how correct the triangle program is. Capture and then report on those results in a formal test report described below. For this first part you should not make any changes to the classify triangle program. You should only change the test program.

Based on the results of your initial tests, you will then update the classify triangle program to fix all defects. Continue to run the test cases as you fix defects until all of the defects have been fixed. Run one final execution of the test program and capture and then report on those results in a formal test report described below.

Note that you should NOT simply replace the logic with your logic from Assignment 1. Test teams typically don't have the luxury of rewriting code from scratch and instead must fix what's delivered to the test team.

Triangle.py contains an implementation of the classifyTriangle() function with a few bugs.

TestTriangle.py contains the initial set of test cases.

Author: Amit Bhatnagar

Summary:

Results:

Initial buggy Implementation (Original Triangle.py)				
Test ID	Input	Expected Results	Actual Result	Pass or Fail
testRightTriangleA	3,4,5	Right	InvalidInput	Fail
testRightTriangleB	5,3,4	Right	InvalidInput	Fail
testEquilateralTriangles	1,1,1	Equilateral	InvalidInput	Fail
testIsosceles	5,5,8	Isosceles	InvalidInput	Fail
testScalene	7,4,5	Scalene	InvalidInput	Fail
testValidTriangleA	4,1,8	NotATriangle	InvalidInput	Fail
testValidTriangleB	8,3,6	NotATriangle	NotATriangle	Pass
testValidInputsA	250, 10, 199	InvalidInput	InvalidInput	Pass
testValidInputsB	3, -5, 2	InvalidInput	InvalidInput	Pass
testNonInts	4.6, 3.2, 5.1	InvalidInput	InvalidInput	Pass
Improved Implementation				
Test ID	Input	Expected Results	Actual Result	Pass or Fail
testRightTriangleA	3,4,5	Right	Right	Pass
testRightTriangleB	5,3,4	Right	Right	Pass
testEquilateralTriangles	1,1,1	Equilateral	Equilateral	Pass
testIsosceles	5,5,8	Isosceles	Isosceles	Pass
testScalene	7,4,5	Scalene	Scalene	Pass
testValidTriangleA	4,1,8	NotATriangle	NotATriangle	Pass
testValidTriangleB	8,3,6	NotATriangle	NotATriangle	Pass
testValidInputsA	250, 10, 199	InvalidInput	InvalidInput	Pass
testValidInputsB	3, -5, 2	InvalidInput	InvalidInput	Pass
testNonInts	4.6, 3.2, 5.1	InvalidInput	InvalidInput	Pass

Reflection: In this assignment it was very important to consider all the edge cases before starting. Just glancing at the original implementation showed me that there would be many flaws. It was important to think of the test cases in the context of the intended functionality, not in the context of the implemented function. The majority of test cases failed for one of two reasons.

The function would return ‘InvalidInput’ for many cases since one of the conditions to return this was that b <= b. However, this would obviously always be true, so this had to be adjusted in the function. Another major problem was that the original function assumed the largest number to be entered as the last parameter, causing two identical triangles to return different results if their sides were inputted in different orders. I found it was valuable to work very slowly through the flow. For example, I would not have caught the semicolon placed after one of the return statements (which I actually mistook for my own input error).

Honor pledge: I pledge my honor that I have abided by the Stevens Honor System.

Detailed results, if any:

There are not many detailed results to discuss that are not already mentioned above, however it is important to note that I had to revise some inputs for my test cases since they would fail other checks before even reaching the check in question. For example, in my testScalene test case, I originally used the inputs (7,4,2). Although all the sides have different lengths in this case, the sides do not actually form a valid triangle. A valid triangle requires that the two smaller sides are greater than or equal to the largest side. Therefore, in this case, the function would have returned ‘NotATriangle’ before even checking whether or not it was scalene.