Description of assignment:

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.

- These are the two files: Triangle.py and TestTriangle.py
    - ***Triangle.py*** is a starter implementation of the triangle classification program.
    - ***TestTriangle.py*** **c**ontains a starter set of unittest test cases to test the classifyTriangle() function in the file Triangle.py file.

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

Avery Cunningham

Summary:

In this assignment I was able to design a set of test cases to ensure a function was working as intended. In this case I first worked to design sufficient test cases to cover all input and output possibilities for the classifyTriangle function. I made sure to handle all edge cases including input order, size, and all triangle types. Once the required test cases were implemented, I

completed part 2 of the assignment by improving the logic and syntax in the classifyTriangle function. Since the test cases were already implemented, it was easy to tell when the errors in the function had been fixed. It took me a total of 3 rounds of testing and 11 test cases to fully improve the function in this assignment. Overall, I found this to be a very educational assignment. It was useful to track how the code improved through successive rounds of testing. I also am a fan of test driven development and enjoyed using a form of it in this application.

https://github.com/A-very-Cunning-ham/SSW-567-Triangles

Part 1:

Test ID	**Input**	**Expected Result**	**Actual Result**	**Pass or Fail**
test_invalidInput_wrongType	("2", "3", "5")	InvalidInput	Error	Fail/Error
test_RightTriangleA	(3,4,5)	Right	InvalidInput	Fail
test_RightTriangleB	(5,3,4)	Right	InvalidInput	Fail
test_equilateral	(3,3,3)	Equilateral	InvalidInput	Fail
test_nonright_isosceles	(1, 1, 1.5)	Isosceles	InvalidInput	Fail
test_nonright_scalene	(1, 2.5, 3)	Scalene	InvalidInput	Fail
test_notATriangle	(1, 1, 3)	NotATriangle	InvalidInput	Fail
test_right_isosceles	(1, sqrt(2), 1)	Right	InvalidInput	Fail
test_right_scalene	(3, 4, 5)	Right	InvalidInput	Fail
test_invalidInput_notPositive	(-1, -3, -4)	InvalidInput	InvalidInput	Pass
test_invalidInput_outOfRange	(300, 300, 300)	InvalidInput	InvalidInput	Pass

Part 2:

Test ID	Input	Expected Result	Actual Result	Pass or Fail
test_invalidInput_wrongType	("2", "3", "5")	InvalidInput	InvalidInput	Pass
test_RightTriangleA	(3,4,5)	Right	Right	Pass
test_RightTriangleB	(5,3,4)	Right	Right	Pass
test_equilateral	(3,3,3)	Equilateral	Equilateral	Pass
test_nonright_isosceles	(1, 1, 1.5)	Isosceles	Isosceles	Pass
test_nonright_scalene	(1, 2.5, 3)	Scalene	Scalene	Pass
test_notATriangle	(1, 1, 3)	NotATriangle	NotATriangle	Pass
test_right_isosceles	(1, sqrt(2), 1)	Right	Right	Pass
test_right_scalene	(3, 4, 5)	Right	Right	Pass
test_invalidInput_notPositive	(-1, -3, -4)	InvalidInput	InvalidInput	Pass
test_invalidInput_outOfRange	(300, 300, 300)	InvalidInput	InvalidInput	Pass

Testing Matrix:

	Test Run 1	Test Run 2	Test Run 3
Tests Planned	All 11 initial tests	No new tests	No new tests

Tests Executed	11	11	11
Tests Passed	2	5	11
Defects Found	Many logic and syntax errors	Logic errors in triangle classification	None
Defects Fixed	Syntax errors and logic errors in input sanitization	All logic errors found	None

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

Detailed results:

One assumption made in this assignment is that the classifyTriangle function prioritizes classifying right triangles over other types of triangles. This is needed in order to keep a single output from the function, but does not account for the cases where there are right scalene and isosceles triangles. A second return value and slightly modified logic would be needed to completely describe all variations of triangles.