1. **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. Triangle.py is a starter implementation of the triangle classification program. TestTriangle.py contains 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.

2. **Author:** Jason McCauley 20006660

3. **Summary:** Results of test set against the initial buggy implementation of classifyTriangle in the original Triangle.py:

Test ID	Input	Expected Results	Actual Result	Pass or Fail
testInputValues1	201, 4, 5	InvalidInput	InvalidInput	Pass
testInputValues2	3, 201, 5	InvalidInput	InvalidInput	Pass
testInputValues3	3, 4, 201	InvalidInput	InvalidInput	Pass
testInputValues4	-1, 4, 5	InvalidInput	InvalidInput	Pass
testInputValues5	3, -1, 5	InvalidInput	InvalidInput	Pass
testInputValues6	3, 4, -1	InvalidInput	InvalidInput	Pass
testInputTypes1	“string”, 4, 5	InvalidInput	TypeError: ‘>’ not supported between instances of ‘str’ and ‘int’	Error

testInputTypes2	3, [], 5	InvalidInput	TypeError: '>' not supported between instances of 'list' and 'int'	Error
testInputTypes3	3, 4, {}	InvalidInput	TypeError: '>' not supported between instances of 'dict' and 'int'	Error
testValidTriangle1	10, 4, 5	NotATriangle	'InvalidInput' != 'NotATriangle'	Fail
testValidTriangle2	3, 10, 5	NotATriangle	'InvalidInput' != 'NotATriangle'	Fail
testValidTriangle3	3, 4, 10	NotATriangle	'InvalidInput' != 'NotATriangle'	Fail
testEquilateralTriangle	5, 5, 5	Equilateral	'InvalidInput' != 'Equilateral'	Fail
testIsoscelesTriangle	5, 5, 3	Isosceles	InvalidInput' != 'Isosceles'	Fail
testIsoscelesRightTriangle	3, 3, 3*math.sqrt(2)	IsoscelesRight	InvalidInput' != 'IsoscelesRight'	Fail
testScaleneTriangle	4, 5, 6	Scalene	InvalidInput' != 'Scalene'	Fail
testScaleneRightTriangle	5, 12, 13	ScaleneRight	InvalidInput' != 'ScaleneRight'	Fail

Results of running test set against the improved implementation of classifyTriangle:

Test ID	Input	Expected Results	Actual Result	Pass or Fail
testInputValues1	201, 4, 5	InvalidInput	InvalidInput	Pass
testInputValues2	3, 201, 5	InvalidInput	InvalidInput	Pass
testInputValues3	3, 4, 201	InvalidInput	InvalidInput	Pass
testInputValues4	-1, 4, 5	InvalidInput	InvalidInput	Pass

testInputValues5	3, -1, 5	InvalidInput	InvalidInput	Pass
testInputValues6	3, 4, -1	InvalidInput	InvalidInput	Pass
testInputTypes1	“string”, 4, 5	InvalidInput	InvalidInput	Pass
testInputTypes2	3, [], 5	InvalidInput	InvalidInput	Pass
testInputTypes3	3, 4, {}	InvalidInput	InvalidInput	Pass
testValidTriangle1	10, 4, 5	NotATriangle	NotATriangle	Pass
testValidTriangle2	3, 10, 5	NotATriangle	NotATriangle	Pass
testValidTriangle3	3, 4, 10	NotATriangle	NotATriangle	Pass
testEquilateralTriangle	5, 5, 5	Equilateral	Equilateral	Pass
testIsoscelesTriangle	5, 5, 3	Isosceles	Isosceles	Pass
testIsoscelesRightTriangle	3, 3, 3*math.sqrt(2)	IsoscelesRight	IsoscelesRight	Pass
testScaleneTriangle	4, 5, 6	Scalene	Scalene	Pass
testScaleneRightTriangle	5, 12, 13	ScaleneRight	ScaleneRight’	Pass

A matrix summarizing results:

	Test Run 1	Test Run 2
Tests Planned	17	17
Tests Executed	17	17
Tests Passed	6	17
Defects Found	11	0
Defects Fixed	0	11

Description of strategy to decide whether test cases were sufficient: To decide whether my test cases were sufficient, I had to read through my program and consider where unexpected could arise. The first being the type of inputs — not only did I have to ensure that only ints and floats could be entered, I also had to ensure that this logic held up for all three inputs. The same logic applies to the bounds of the inputs. I had to check that all three inputs would properly throw

errors if just one of them was negative or greater than 200. After verifying the inputs, I now had to check that the logic for determining the triangle type was sound. To accomplish this, I created test cases for each possible triangle type, certifying that for the given side lengths, the expected blocks of code would be accessed and executed. With this plan in mind, I was able to develop a thorough and diverse set of test cases, ensuring that my code both functioned as intended and handled inputs robustly.

Reflection: My biggest takeaway from this assignment is that when working in groups or in the industry, not all code is going to be written by you. Often times, you are going to be working with code that was written by someone else, therefore, it is important to be able to read through existing code, understand the flow and thought process behind it, as well as the results it intends to produce. Additionally, it is important to be able to read through that code and identify any shortcomings or bugs that it may produce, from which you can make modifications. Additionally, this assignment reinforced the importance of unit testing, and making sure that your code can properly handle a variety of input values and types without producing unexpected results.

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

5. **Detailed Results:** For this assignment, I assumed that the user would be entering exactly three inputs, given that a triangle has three side lengths. Furthermore, I assumed that the input would be either type int or float. Originally, I intended for the user to only be able to enter inputs of type int, however, it would be impossible for the user to enter an isosceles right triangle, as the hypotenuse is always the leg length * sqrt(2), which of course, cannot be entered as an int. For this reason, I had to expand the program to allow inputs of type float. Consequently, for determining if a triangle is right, I had to refine my comparison statements to account for floating point precision.