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.

- 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.

2. Author: Bruno Salgado

3. Summary:

	Test Run 1	Test Run 2
Tests Planned	9	9
Tests Executed	9	9
Tests Passed	3	9
Defects Found	6	0
Defects Fixed	3	0

Through this assignment, I was able to experience a scenario where I updated test case code for an existing file. When I first read through the test cases, I felt they were incomplete and did not cover all possible flows in the tested function. By adding more test cases, I was able to cover more cases in the legacy code and also uncover issues in the code that helped me to eventually correct the pre-existing logic. After rewriting some of the logic, I was able to retest the code using my updated test cases and used this process of coding and testing to eventually achieve complete success across all of my test cases.

The strategy I used to determine when I had achieved a sufficient number of test cases involved writing two different tests per functionality. For instance, the triangle classification code should be able to discern equilateral, isosceles, and scalene triangles, therefore, my test cases include two test cases for each of these scenarios. I also included a couple of edge case tests which tested to see if the legacy code caught invalid triangles or invalid inputs. Altogether, the 9 test cases I have in my test file sufficiently covered the legacy code and helped me to ensure the legacy code was mostly bug free.

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

Test Run 1 (buggy code):

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
test_isosceles_A	(40, 40, 60)	“Isosceles”	“InvalidInput”	Fail
test_isosceles_B	(2, 7, 7)	“Isosceles”	“InvalidInput”	Fail
test_scalene_A	(3, 4, 6)	“Scalene”	“Scalene”	Pass
test_scalene_B	(100, 99, 102)	“Scalene”	“InvalidInput”	Fail
test_equilateral_C	(0.1, 0.1, 0.1)	“InvalidInput”	“InvalidInput”	Pass
test_not_a_triangle	(1, 2, 3)	“NotATriangle”	“NotATriangle”	Pass

Test Run 2 (fixed code):

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
test_isosceles_A	(40, 40, 60)	“Isosceles”	“Isosceles”	Pass
test_isosceles_B	(2, 7, 7)	“Isosceles”	“Isosceles”	Pass
test_scalene_A	(3, 4, 6)	“Scalene”	“Scalene”	Pass
test_scalene_B	(100, 99, 102)	“Scalene”	“Scalene”	Pass
test_equilateral_C	(0.1, 0.1, 0.1)	“InvalidInput”	“InvalidInput”	Pass
test_not_a_triangle	(1, 2, 3)	“NotATriangle”	“NotATriangle”	Pass