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

Charles Villa

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

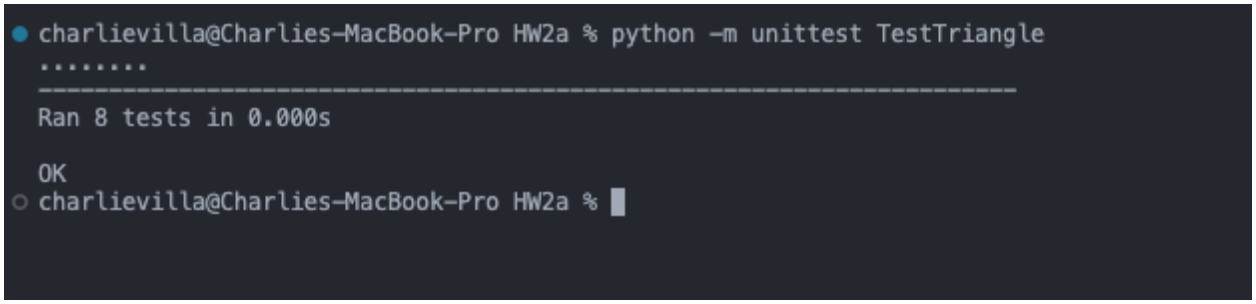
Matrix for initial function:

Test ID	**Input**	**Expected Results**	**Actual Result**	**Pass or Fail**
RightTriangleA	3,4,5	Right	InvalidInput	Fail
RightTriangleB	5,3,4	Right	InvalidInput	Fail
EquilateralTrangles	1,1,1	Equilateral	InvalidInput	Fail

IsoscelesTriangles	5,5,3 5,3,5	Isosceles Isosceles	InvalidInput InvalidInput	Fail Fail
ScaleneTriangles	5,6,7	Scalene	InvalidInput	Fail
InvalidInputs	-1,2,3 0,0,0 234,452,343	InvalidInput InvalidInput InvalidInput	InvalidInput InvalidInput InvalidInput	Pass Pass Pass
NonIntegerInputs	3.5,4,5	InvalidInput	InvalidInput	Pass
NotATriangle	1,10,12	NotATriangle	InvalidInput	Fail

Matrix after updates:

Test ID	**Input**	**Expected Results**	**Actual Result**	**Pass or Fail**
RightTriangleA	3,4,5	Right	Right	Pass
RightTriangleB	5,3,4	Right	Right	Pass
EquilateralTrangles	1,1,1	Equilateral	Equilateral	Pass
IsoscelesTriangles	5,5,3 5,3,5	Isosceles Isosceles	Isosceles Isosceles	Pass Pass
ScaleneTriangles	5,6,7	Scalene	Scalene	Pass
InvalidInputs	-1,2,3 0,0,0 234,452,343	InvalidInput InvalidInput InvalidInput	InvalidInput InvalidInput InvalidInput	Pass Pass Pass
NonIntegerInputs	3.5,4,5	InvalidInput	InvalidInput	Pass
NotATriangle	1,10,12	NotATriangle	NotATriangle	Pass



This was the result after changing the logic within the Triangle.py file. What I learned from this assignment is that even simple mistakes can cause a huge cascade of failures. There weren't many errors within the original code yet it made the code unusable. It stresses how important it is to test code and keep track of using test cases. The thing that worked for me was just drawing the code through on my iPad to see how and where the code was messing up. What didn't work was trying to locate the errors through the terminal error message.

Detailed Results:

These were the logic errors I found within the original function:

- Some of the NotATriangle conditions were subtracting the two other sides instead of adding them together.
- The equilateral triangle conditions only checked if two sides were equal instead of all 3.
- The right triangle conditions incorrectly implemented the Pythagorean theorem by doing normal multiplication instead of exponents. It also didn't check all the ways that the sides could be arranged.
- The scalene triangle condition checked the same condition twice which was redundant so it missed a condition.
- The else statement should've been the scalene triangle instead of the isosceles triangle since isosceles checks for equality.

Assumptions:

- Triangles are based on their sides, no angles are involved
- The triangle inequality theorem must be passed

Constraints:

- Input value has to be 200 or less to be considered valid(was already there)
- Inputs must be integers

Data inputs:

- Implemented data inputs for each test case situation. Those test cases were based on the logic of the code. So, for the invalid input, I made sure to get each kind of way that the invalid input should show up(ex. Input above 200 or a negative number)

Explanation:

- The copies of the code and and results are within the GitHub folder: https://github.com/CharlesVilla68/SSW-567-HW It should be public but I'll share it as well with the professor and TA.