Cody Fernandez, Gleb Myshkin, Jonathan Memoli, Matthew Feroz
SSW 567-WS
I pledge my honor that I have abided by the Stevens Honor System.

HW 02a - Testing a legacy program and reporting on testing results

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 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

Repo Name & Link:

- TriangleTesting
- https://github.com/CodyBF/TriangleTesting

Part 1:

Test ID	Input	Expected Results	Actual Result	Pass or Fail
testInvalidInputA	100,150,201	InvalidInput	InvalidInput	Pass
testInvalidInputB	-2,-1,-5	InvalidInput	InvalidInput	Pass
testInvalidInputC	'2',10,12	InvalidInput	InvalidInput	Pass
testNotATriangleA	1,2,3	NotATriangle	InvalidInput	Fail
testNotATriangleB	3,4,8	NotATriangle	InvalidInput	Fail
testRightTriangleA	3,4,5	Right	InvalidInput	Fail
testRightTriangleB	5,3,4	Right	InvalidInput	Fail
testScaleneTriangleA	3,5,7	Scalene	InvalidInput	Fail
testScaleneTriangleB	4,11,8	Scalene	InvalidInput	Fail
testEquilateralTriangleA	1,1,1	Equilateral	InvalidInput	Fail
testEquilateralTriangleB	2,2,2	Equilateral	InvalidInput	Fail
testIsoscelesTriangleA	2,2,3	Isosceles	InvalidInput	Fail
testIsoscelesTriangleB	5,5,7	Isosceles	InvalidInput	Fail

Part 2:

Test ID	Input	Expected Results	Actual Result	Pass or Fail
testInvalidInputA	100,150,201	InvalidInput	InvalidInput	Pass
testInvalidInputB	-2,-1,-5	InvalidInput	InvalidInput	Pass
testInvalidInputC	'2',10,12	InvalidInput	InvalidInput	Pass
testNotATriangleA	1,2,3	NotATriangle	NotATriangle	Pass
testNotATriangleB	3,4,8	NotATriangle	NotATriangle	Pass
testRightTriangleA	3,4,5	Right	Right	Pass
testRightTriangleB	5,3,4	Right	Right	Pass
testScaleneTriangleA	3,5,7	Scalene	Scalene	Pass
testScaleneTriangleB	4,11,8	Scalene	Scalene	Pass
testEquilateralTriangleA	1,1,1	Equilateral	Equilateral	Pass
testEquilateralTriangleB	2,2,2	Equilateral	Equilateral	Pass
testIsoscelesTriangleA	2,2,3	Isosceles	Isosceles	Pass
testIsoscelesTriangleB	5,5,7	Isosceles	Isosceles	Pass

Summary:

- All Issues found in code
    - Should move the checker for our values being integers above the checks of greater than 0 less
    - # than 200.
    - line 34, the '=' symbols are unnecessary.
    - line 34, the if statement for 'b<=b' will always result in a return of 'InvalidInput'
    - line 40, the semicolon after 'InvalidInput' is unnecessary
    - line 46, the logic for deciding if the inputs are a triangle is wrong,
    - change the '-' symbols to '+' symbols.
    - line 50, the logic for equilateral triangles is wrong, its missing a check for the value of c
    - line 52, the logic for right triangle is wrong, instead of multiplying our values by 2 with a single '*' we should write it as 'a ** 2)
    - line 54, the logic for scalene triangles is wrong, last check should be 'a!=c'
    - line 57, Isosceles is spelled incorrectly

Listed above are all the issues we found in the code, it's worth noting that the second issue found is what caused all initial tests to fail. We believe the assignment did a good job of showing that it's good to be prepared for as many bugs as we can think of, and write the proper test cases to ensure everything works out correctly. Even one small bug in the code can cause most test cases to fail, regardless of input, and it's important to pinpoint what caused the problem and fix it. After making changes to all bugs we found, all of our test cases passed successfully as shown in the matrix below. We felt that two tests for each triangle, checking whether or not the inputs made a valid triangle, and 3 tests for invalid inputs would be sufficient.

	Test Run 1	**Test Run 2**
Tests Planned	13	13
Tests Executed	13	13
Tests Passed	3	13
Defects Found	10	0
Defects Fixed	0	10

- **Screen dump of us running test set on improved classifyTriangle() function**