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

**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 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:** Ashish Singh

**3) Summary table**

|  |  |  |
| --- | --- | --- |
|  | **Test Run 1** | **Test Run 2** |
| **Tests Planned** | **22** | **22** |
| **Tests Executed** | **22** | **22** |
| **Tests Passed** | **7** | **22** |
| **Defects Found** | **15** | **0** |
| **Defects Fixed** | **15** | **0** |

**Test Run - 1**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test ID** | **Input** | **Expected Results** | **Actual Result** | **Pass or Fail** |
| 1 | classifyTriangle(3,4,5) | Right | InvalidInput | Fail |
| 2 | classifyTriangle(3,5,4) | Right | InvalidInput | Fail |
| 3 | classifyTriangle(5,4,3) | Right | InvalidInput | Fail |
| 4 | classifyTriangle(201,190,200) | InvalidInput | InvalidInput | Pass |
| 5 | classifyTriangle(200,201,200) | InvalidInput | InvalidInput | Pass |
| 6 | classifyTriangle(190,200,201) | InvalidInput | InvalidInput | Pass |
| 7 | classifyTriangle(0,190,200) | InvalidInput | InvalidInput | Pass |
| 8 | classifyTriangle(200,0,200) | InvalidInput | InvalidInput | Pass |
| 9 | classifyTriangle(190,200,-1) | InvalidInput | InvalidInput | Pass |
| 10 | classifyTriangle('190',200,0) | InvalidInput | TypeError: '>' not supported between instances of 'str' and 'int' | Fail |
| 11 | classifyTriangle(3,"two",5) | InvalidInput | TypeError: '>' not supported between instances of 'str' and 'int' | Fail |
| 12 | classifyTriangle(3,2,4.1) | InvalidInput | InvalidInput | Pass |
| 13 | classifyTriangle(2,4,6) | NotATriangle | InvalidInput | Fail |
| 14 | classifyTriangle(2,4,1) | NotATriangle | InvalidInput | Fail |
| 15 | classifyTriangle(11,4,6) | NotATriangle | InvalidInput | Fail |
| 16 | classifyTriangle(1,1,1) | Equilateral | InvalidInput | Fail |
| 17 | classifyTriangle(200,200,200) | Equilateral | InvalidInput | Fail |
| 18 | classifyTriangle(100,100,4) | Equilateral | InvalidInput | Fail |
| 19 | classifyTriangle(3,6,4) | Scalene | InvalidInput | Fail |
| 20 | classifyTriangle(3,4,3) | Isosceles | InvalidInput | Fail |
| 21 | classifyTriangle(5,4,4) | Isosceles | InvalidInput | Fail |
| 22 | classifyTriangle(3,3,2) | Isosceles | InvalidInput | Fail |

**Test Run – 2**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test ID** | **Input** | **Expected Results** | **Actual Result** | **Pass or Fail** |
| 1 | classifyTriangle(3,4,5) | Right | Right | Pass |
| 2 | classifyTriangle(3,5,4) | Right | Right | Pass |
| 3 | classifyTriangle(5,4,3) | Right | Right | Pass |
| 4 | classifyTriangle(201,190,200) | InvalidInput | InvalidInput | Pass |
| 5 | classifyTriangle(200,201,200) | InvalidInput | InvalidInput | Pass |
| 6 | classifyTriangle(190,200,201) | InvalidInput | InvalidInput | Pass |
| 7 | classifyTriangle(0,190,200) | InvalidInput | InvalidInput | Pass |
| 8 | classifyTriangle(200,0,200) | InvalidInput | InvalidInput | Pass |
| 9 | classifyTriangle(190,200,-1) | InvalidInput | InvalidInput | Pass |
| 10 | classifyTriangle('190',200,0) | InvalidInput | InvalidInput | Pass |
| 11 | classifyTriangle(3,"two",5) | InvalidInput | InvalidInput | Pass |
| 12 | classifyTriangle(3,2,4.1) | InvalidInput | InvalidInput | Pass |
| 13 | classifyTriangle(2,4,6) | NotATriangle | NotATriangle | Pass |
| 14 | classifyTriangle(2,4,1) | NotATriangle | NotATriangle | Pass |
| 15 | classifyTriangle(11,4,6) | NotATriangle | NotATriangle | Pass |
| 16 | classifyTriangle(1,1,1) | Equilateral | Equilateral | Pass |
| 17 | classifyTriangle(200,200,200) | Equilateral | Equilateral | Pass |
| 18 | classifyTriangle(100,100,4) | Equilateral | Equilateral | Pass |
| 19 | classifyTriangle(3,6,4) | Scalene | Scalene | Pass |
| 20 | classifyTriangle(3,4,3) | Isosceles | Isosceles | Pass |
| 21 | classifyTriangle(5,4,4) | Isosceles | Isosceles | Pass |
| 22 | classifyTriangle(3,3,2) | Isosceles | Isosceles | Pass |

**4) Honor Pledge:** I pledge on my honor that I have not given or received any unauthorized assistance on this assignment/examination. I further pledge that I have not copied any material from a book, article, the Internet or any other source except where I have expressly cited the source.

**5) Detailed results and assumptions:**

* In **Test Run-1** some ‘InvalidInput’ tests were successful. However, not all of these returned ‘InvalidInput’ because of the right reason. Example, classifyTriangle(0,190,200) should return ‘InvalidInput’ as one of the values is zero, however, it returned this value due to a problem in the classifyTriangle code:

if a <= 0 or **b <= b** or c <= 0:

return 'InvalidInput'

* Some triangles can be Right and well as Scalene or Isosceles. My modified Triangle.py file code does not check for this as we are instructed to only modify the existing code and not replace it with our logic.
* I have included input values to check for the boundary conditions (>0 and <=200), values somewhere in the middle of this range, and by changing the order of the parameters
* For Right triangles, the original code assumed input value ‘c’ will be the hypotenuse, however, docstring at the beginning of Triangle.py file instructs that “If the sum of any two sides equals the square of the third side, then return 'Right'. Modified the classifyTriangle() code to consider the longest side as the hypotenuse.