**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***](https://sit.instructure.com/courses/40463/files/6396462/download?wrap=1)is a starter implementation of the triangle classification program.
  + [***TestTriangle.py***](https://sit.instructure.com/courses/40463/files/6396461/download?wrap=1)**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 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.

**Author**: Khusboo Patel

**Summary**: Written total 12 test cases to verify the initial *“classifyTriangle”* program. Using test framework was able to identify 9 failures. This helped to pinpoint the root cause of the source code error in *“classifyTriangle”* program. There were multiple issues in the program. This approach is straight forward to identify the issues right away in the initial stages.

**Reflection**: Gave a good understanding of the how unit testing works. Had to take into account both negative and positive scenarios.

**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.”*

**Detailed Results:**

Below is the initial summary of test cases results:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test ID** | **Input** | **Expected Results** | **Actual Result** | **Pass or Fail** |
| TC\_1 | 3,4,5 | Right | InvalidInput | Fail |
| TC\_2 | 5,3,4 | Right | InvalidInput | Fail |
| TC\_3 | 1,1,1 | Equilateral | InvalidInput | Fail |
| TC\_4 | 3,3,3 | Equilateral | InvalidInput | Fail |
| TC\_5 | 2,1,1 | Isoceles | InvalidInput | Fail |
| TC\_6 | 1,2,1 | Isoceles | InvalidInput | Fail |
| TC\_7 | 2,3,1 | Scalene | InvalidInput | Fail |
| TC\_8 | 2,5,6 | Scalene | InvalidInput | Fail |
| TC\_9 | -1,2,1 | InvalidInput | InvalidInput | Pass |
| TC\_10 | 1.5,3,2 | InvalidInput | InvalidInput | Pass |
| TC\_11 | 202,1,1 | InvalidInput | InvalidInput | Pass |
| TC\_12 | 0,0,0 | InvalidInput | InvalidInput | Pass |
| TC\_13 | 1,2,15 | NotATriangle | InvalidInput | Fail |

A screenshot of a computer

Description automatically generated

Below is the summary of test cases results for updated *“classifyTriangle()”* program :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test ID** | **Input** | **Expected Results** | **Actual Result** | **Pass or Fail** |
| TC\_1 | 3,4,5 | Right | Right | Pass |
| TC\_2 | 5,3,4 | Right | Right | Pass |
| TC\_3 | 1,1,1 | Equilateral | Equilateral | Pass |
| TC\_4 | 3,3,3 | Equilateral | Equilateral | Pass |
| TC\_5 | 2,1,1 | Isoceles | Isoceles | Pass |
| TC\_6 | 1,2,1 | Isoceles | Isoceles | Pass |
| TC\_7 | 2,3,1 | Scalene | Scalene | Pass |
| TC\_8 | 2,5,6 | Scalene | Scalene | Pass |
| TC\_9 | -1,2,1 | InvalidInput | InvalidInput | Pass |
| TC\_10 | 1.5,3,2 | InvalidInput | InvalidInput | Pass |
| TC\_11 | 202,1,1 | InvalidInput | InvalidInput | Pass |
| TC\_12 | 0,0,0 | InvalidInput | InvalidInput | Pass |
| TC\_13 | 1,2,15 | NotATriangle | NotATriangle | Pass |

Text

Description automatically generated

**Summary Matrix:**

Provided different inputs for each side of the triangle to verify if any side is triggering an invalid scenario. This technique is used to test the robustness of each input.

Below is the test execution summary matrix of triangle program:

|  |  |  |
| --- | --- | --- |
| **Test details** | **Test Run 1** | **Test Run 2** |
| Tests Planned | 13 | 13 |
| Tests Executed | 13 | 13 |
| Tests Passed | 4 | 13 |
| Defects Found | 9 | 0 |
| Defects Fixed | 0 | 9 |

Github Repository: <https://github.com/khusboo98/567_HW_2a_Triangle>