**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.

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.

**Author:**

Samantha Inneo

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test ID | Input | Expected Results | Actual Results | Pass or Fail |
| testRightTriangleA | 3, 4, 5 | Right | InvalidInput | Fail |
| testRightTriangleB | 6, 8, 10 | Right | InvalidInput | Fail |
| testRightTriangleC | 5, 3, 4 | Right | InvalidInput | Fail |
| testRightTriangleD | 10, 8, 6 | Right | InvalidInput | Fail |
| testEquilateralTriangleA | 1, 1, 1 | Equilateral | InvalidInput | Fail |
| testEquilalateralTriangleB | 200, 200, 200 | Equilateral | InvalidInput | Fail |
| testNotATriangleA | 10, 10, 20 | NotATrianlge | InvalidInput | Fail |
| testNotATriangleB | 10, 10, 100 | NotATriangle | InvalidInput | Fail |
| testNotATriangleC | 10, 15, 30 | NotATriangle | InvalidInput | Fail |
| testIsoscelesTriangleA | 10, 10, 18 | Isosceles | InvalidInput | Fail |
| testIsoscelesTriangleB | 18, 10, 10 | Isosceles | InvalidInput | Fail |
| testScaleneTriangleA | 2, 5, 6 | Scalene | InvalidInput | Fail |
| testScaleneTriangleB | 6, 5, 2 | Scalene | InvalidInput | Fail |
| testInvalidInputA | -1, -1, -1 | InvalidInput | InvalidInput | Pass |
| testInvalidTriangleB | 201,201,201 | InvalidInput | InvalidInput | Pass |
| testInvalidTriangleC | 6.5, 5.5, 2.5 | InvalidInput | InvalidInput | Pass |
| testInvalidTriangleD | 1, 0, 5 | InvalidInput | InvalidInput | Pass |

**Summary:**

I decided I had enough test cases once I had successfully covered every flaw I could anticipate in the program. It was easy to think of some of these examples such as the various invalid inputs, because they were outlined in the comments of the code. One of the more challenging ones to think of was having the same input in reverse order, to make sure the code worked regardless of which numbers were put in first. I also made sure to check for cases where the triangle could fall under multiple categories, like a scalene right triangle.

After running the test file for the first time, it was clear there was at least one major issue with the provided code. Every single input resulted in ‘InvalidInput’ as the output, which caused nearly every test in the file to fail. I knew it was likely that all these failures would be caused by one issue, so I looked for the areas of code that return ‘InvalidInput’. I found that on line 34, the if statement checking that the number was at least 0 was checking if b >= b, which was always true, so every single input was being categorized as invalid. I changed it to instead check that b>=0, and instantly fixed that issue. The next major issue I set out to fix was that many triangles were incorrectly being labeled ‘NotATriangle’ when they were in fact valid triangles. (Ex: 1,1,1) To fix this, I went to line 46, where the logic on what defines a triangle was incorrect. It was checking that the sum of any 2 sides is shorter than the third, when it should have been checking for it to be greater. Looking at the remaining tests, I realized something might be wrong with the equilateral code, since it was incorrectly defining some isosceles to be equilateral. I found that the code only checked that the first two sides were equal, not all three. The remaining failures were all on right triangles, so that is the next place I began checking. The first reason this was wrong was that they were multiplying by 2 rather than squaring the numbers. The last fix I had to make was that they never checked which side would be a, b, and c. For Pythagorean theorem to work, c must by the largest of the 3 sides. I added ors to the clause to check if a or b are the largest. At this point, all of the tests passed.

I think this was a well designed assignment. It was smart to use the assignment we did the previous week with intentional errors because I was already familiar with what I would need to test for after I already completed the assignment. Even though I coded it on my own last week, looking at this implementation allowed me to find different solutions to some issues since it would fit better into their existing framework. The main strategy I used for debugging was to think about what failing cases had in common. For this assignment it was pretty easy since there were so few different outputs, but I think this strategy will still serve me well in the future even on more complex assignments.

**Honor Pledge:**

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

**Detailed Results:**

I explain the techniques I used to formulate test cases above in the ‘Summary’ section. The only assumptions I used for this assignment were the ones made apparent in the code, such as the upper bounding on input size of 200 and that only integers would be accepted input. While triangles can have large or decimal sides, I assumed since this was implemented in the given code that these were criteria that must be followed. For the data inputs, the tests mostly contain integer values between 0 and 200, but there are some cases that fall outside this criteria specifically to catch invalid inputs.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test ID | Input | Expected Results | Actual Results | Pass or Fail |
| testRightTriangleA | 3, 4, 5 | Right | Right | Pass |
| testRightTriangleB | 6, 8, 10 | Right | Right | Pass |
| testRightTriangleC | 5, 3, 4 | Right | Right | Pass |
| testRightTriangleD | 10, 8, 6 | Right | Right | Pass |
| testEquilateralTriangleA | 1, 1, 1 | Equilateral | Equilateral | Pass |
| testEquilalateralTriangleB | 200, 200, 200 | Equilateral | Equilateral | Pass |
| testNotATriangleA | 10, 10, 20 | NotATrianlge | NotATrianlge | Pass |
| testNotATriangleB | 10, 10, 100 | NotATriangle | NotATriangle | Pass |
| testNotATriangleC | 10, 15, 30 | NotATriangle | NotATriangle | Pass |
| testIsoscelesTriangleA | 10, 10, 18 | Isosceles | Isosceles | Pass |
| testIsoscelesTriangleB | 18, 10, 10 | Isosceles | Isosceles | Pass |
| testScaleneTriangleA | 2, 5, 6 | Scalene | Scalene | Pass |
| testScaleneTriangleB | 6, 5, 2 | Scalene | Scalene | Pass |
| testInvalidInputA | -1, -1, -1 | InvalidInput | InvalidInput | Pass |
| testInvalidTriangleB | 201,201,201 | InvalidInput | InvalidInput | Pass |
| testInvalidTriangleC | 6.5, 5.5, 2.5 | InvalidInput | InvalidInput | Pass |
| testInvalidTriangleD | 1, 0, 5 | InvalidInput | InvalidInput | Pass |

Above is the final results of my testing after making edits to the Triangle.py file. I was able to get all the tests to pass. Below you will find the test matrix. While I could have probably taken less trials to fix the code if I looked at every failing case at once, I instead decided to focus on one issue at a time. I used this strategy because I didn’t want to make multiple edits at once and then not know which fixed the failing cases, or worse broke previously passing test cases. It may not have been necessary for this type of assignment, but a more complex assignment would require this carefulness.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Test Run 1 | Test Run 2 | Test Run 3 | Test Run 4 | Test Run 5 |
| Tests Planned | 17 | 17 | 17 | 17 | 17 |
| Tests Executed | 17 | 17 | 17 | 17 | 17 |
| Tests Passed | 4 | 7 | 8 | 13 | 17 |
| Defects Found | 1 | 1 | 1 | 2 | 0 |
| Defects Fixed | 1 | 1 | 1 | 2 | 0 |

Github Link: https://github.com/samanthainneo99/SSW567HW2