**SSW 567 – Homework 2a**

**Author – Suchita Dmello**

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

**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/30792/files/4506927/download?verifier=jLYmOGOtUMBSHsZaB8s0Uv6K2OEd4E8Kot4R5PC4&wrap=1)is a starter implementation of the triangle classification program.
  + [***TestTriangle.py***](https://sit.instructure.com/courses/30792/files/4506926/download?verifier=I3T0FJXx8f5B5iNZZTOHN9KSJHYfJ4wPAQTG7IEP&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 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*](https://sit.instructure.com/courses/30792/files/4506927/download?verifier=jLYmOGOtUMBSHsZaB8s0Uv6K2OEd4E8Kot4R5PC4&wrap=1)*contains an implementation of the classifyTriangle() function with a few bugs.*  
  
[*TestTriangle.py*](https://sit.instructure.com/courses/30792/files/4506926/download?verifier=I3T0FJXx8f5B5iNZZTOHN9KSJHYfJ4wPAQTG7IEP&wrap=1)*contains the initial set of test cases*

Repository: <https://github.com/sdmello14/ssw567-Hw02>

**Summary**:

**Part 0:** Created the following git hub repository for week 2 homework assignment:

**Repository:** https ://github.com/sdmello14/ssw567-Hw02

**Part 1:**

Total 20 test cases were run from the TestTriangle.py file to test Triangle.py code. All the test cases initially failed.

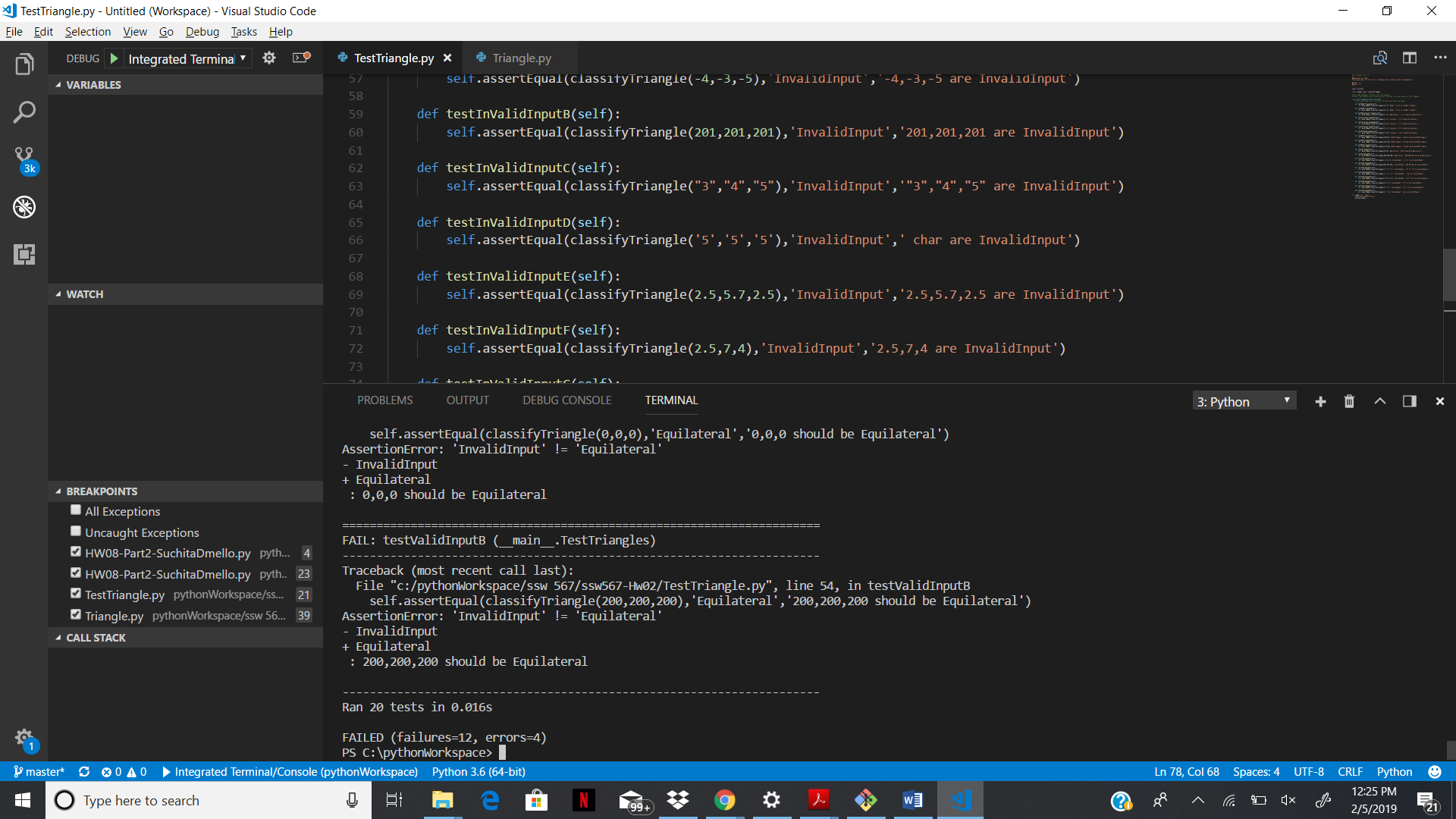


Table below gives the details of the test case before defect fixing:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test ID** | **Input** | **Expected Results** | **Actual Result** | **Pass or Fail** |
| TC01 | 3,4,5 | Right | InvalidInput | Fail |
| TC02 | 5,3,4 | Right | InvalidInput | Fail |
| TC03 | 1,1,1 | Equilateral | InvalidInput | Fail |
| TC04 | 1,1,2 | Isoceles | InvalidInput | Fail |
| TC05 | 2,1,2 | Isoceles | InvalidInput | Fail |
| TC06 | 5,1,1 | Isoceles | InvalidInput | Fail |
| TC07 | 7,5,15 | Scalene | InvalidInput | Fail |
| TC08 | 10,20,2 | NotATriangle | InvalidInput | Fail |
| TC09 | 10,2,20 | NotATriangle | InvalidInput | Fail |
| TC10 | 2,10,20 | NotATriangle | InvalidInput | Fail |
| TC11 | 0,0,0 | NotATriangle | InvalidInput | Fail |
| TC12 | 200,200,200 | Equilateral | InvalidInput | Fail |
| TC13 | -4,-3,-5 | InvalidInput | InvalidInput | Pass |
| TC14 | 201,201,201 | InvalidInput | InvalidInput | Pass |
| TC15 | "3","4","5" | InvalidInput | TypeError | Pass |
| TC16 | '5','5','5' | InvalidInput | TypeError | Pass |
| TC17 | 2.5,5.7,2.5 | InvalidInput | InvalidInput | Pass |
| TC18 | 2.5,7,4 | InvalidInput | InvalidInput | Pass |
| TC19 | 2,7,”4” | InvalidInput | TypeError | Pass |
| TC20 | 2,’7’,4 | InvalidInput | TypeError | Pass |

**Part 02:** To fix the test cases, I analyzed the Triangle.py code to identify the defects. In total 8 defects were found in the code which lead to the failure of the test cases. Table below shows the details of the defect found.

|  |  |  |  |
| --- | --- | --- | --- |
| **Defect ID** | **Line Number** | **Actual code** | **Updated Code** |
| Defect01 | Line 34 | if a <= 0 or b <= b or c <= 0: | if a < 0 or b < 0 or c < 0: |
| Defect02 | Line 39 | Integer check is at the last | Integer check should be done first |
| Defect03 | Line 46 | if (a >= (b - c)) or (b >= (a - c)) or (c >= (a + b)): | if (a >= (b + c)) or (b >= (a + c)) or (c >= (a + b)): |
| Defect04 | Line 50 | if a == b and b == a: | if a == b and b == c: |
| Defect05 | Line 52 | elif ((a \* 2) + (b \* 2)) == (c \* 2): | elif ((a \* a) + (b \* b)) == (c \* c): |
| Defect06 | Line 54 | elif (a != b) and (b != c) and (a != b): | elif (a != b) and (b != c) and (a != c): |
| Defect07 | Line 46 | if (a >= (b - c)) or (b >= (a - c)) or (c >= (a + b)): | if (a < (b + c)) or (b < (a + c)) or (c < (a + b)): |
| Defect08 | Line 52 | elif (((a \* a) + (b \* b)) == (c \* c)): | elif (((a \* a) + (b \* b)) == (c \* c)) or (((b \* b) + (c \* c)) == (a \* a)) or (((a \* a) + (c \* c)) == (b \* b)): |

All the defects were fixed due to which all the test cases were passed successfully.

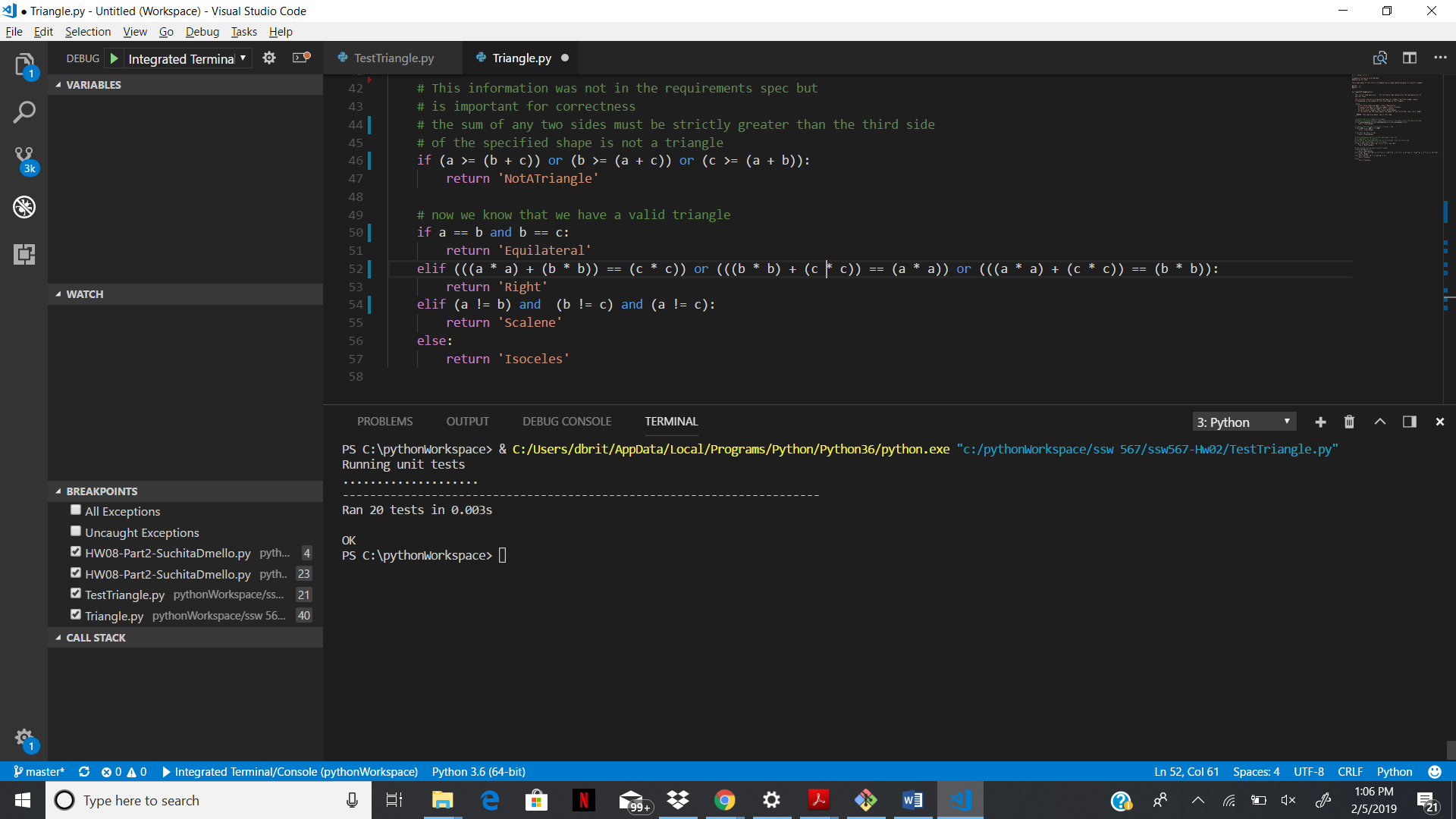


Table below shows the details of test cases after the defect fixing:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test ID** | **Input** | **Expected Results** | **Actual Result** | **Pass or Fail** |
| TC01 | 3,4,5 | Right | Right | Pass |
| TC02 | 5,3,4 | Right | Right | Pass |
| TC03 | 1,1,1 | Equilateral | Equilateral | Pass |
| TC04 | 1,1,2 | Isoceles | Isoceles | Pass |
| TC05 | 2,1,2 | Isoceles | Isoceles | Pass |
| TC06 | 5,1,1 | Isoceles | Isoceles | Pass |
| TC07 | 7,5,15 | Scalene | Scalene | Pass |
| TC08 | 10,20,2 | NotATriangle | NotATriangle | Pass |
| TC09 | 10,2,20 | NotATriangle | NotATriangle | Pass |
| TC10 | 2,10,20 | NotATriangle | NotATriangle | Pass |
| TC11 | 0,0,0 | NotATriangle | NotATriangle | Pass |
| TC12 | 200,200,200 | Equilateral | Equilateral | Pass |
| TC13 | -4,-3,-5 | InvalidInput | InvalidInput | Pass |
| TC14 | 201,201,201 | InvalidInput | InvalidInput | Pass |
| TC15 | "3","4","5" | InvalidInput | InvalidInput | Pass |
| TC16 | '5','5','5' | InvalidInput | InvalidInput | Pass |
| TC17 | 2.5,5.7,2.5 | InvalidInput | InvalidInput | Pass |
| TC18 | 2.5,7,4 | InvalidInput | InvalidInput | Pass |
| TC19 | 2,7,”4” | InvalidInput | InvalidInput | Pass |
| TC20 | 2,’7’,4 | InvalidInput | InvalidInput | Pass |

**Result:**

For part01, I added more 17 test cases which includes test cases with following scenarios:

1. Inputs to test different types of triangle
2. Same input in different combination
3. Boundary condition to test the input
4. Different data types of input

Since the test cases cover all the scenarios, I concluded that the 20 test cases will be sufficient to test the performance of the Triangle.py file.

For part02, I analyzed the code line by line to identify the defects and was able to find and fix 8 defects in the code. After the defect fixing, all the test cases passed successfully.

|  |  |  |
| --- | --- | --- |
|  | **Test Run 1** | **Test Run 2** |
| **Tests Planned** | 20 | 20 |
| **Tests Executed** | 20 | 20 |
| **Tests Passed** | 0 | 20 |
| **Defects Found** | 8 | 0 |
| **Defects Fixed** | 0 | 8 |

**Conclusion:** This assignment helps me to understand the importance of test cases. I have always learned that writing test cases before coding is the best practice, but never implemented the same. This exercise showed the advantage of writing the test cases before and I will practice the same in future.