SSW 567 HW02a - Testing a legacy program and reporting on testing results

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: Jolene Ciccarone

Summary: In this assignment, I made test cases for the function classifyTriangle() and then improved the classifyTriangle() function to successfully pass the test cases. I learned that there are many different factors to consider when creating test cases and that even seemingly simple code can be very detailed. In addition, it is very important to think about the data inputs the code is asking for and how to deal with it through data validation. I originally did not think about decimal lengths, out of order lengths or large and zero lengths because I assumed all the inputs would be in-order integers. After taking into consideration these extra specifications, it required the need for more test cases as well as more conditions in the source code. I also learned more about how to edit one's code without taking away its integrity. Oftentimes in real-world situations, it's necessary to fix bugs in code without completely re-writing it in your own style. This was definitely a challenge because of how many bugs there were in the code, but I tried my best to keep all

the working parts and not rewrite them out of need for more efficiency. The assignment also helped us explore the importance of writing smart and efficient test cases before even writing code. Writing test cases first helps one organize their thoughts and also better understand what is the purpose of the program. Understanding first how the program is supposed to react to different data inputs better clears a path for understanding what functionality is needed in the program. All in all, this was an enriching and challenging assignment that tackled many different aspects of testing.

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

Detailed Results:

Part 1:

The task was to enhance the set of test cases for the Triangle problem that adequately tests the classifyTriangle() function to find problems.

Here is the test report when the improved test cases were run against the original Triangle function. There were 17 failed test cases out of 23 test cases run.

Test ID	Input	Expected Results	Actual Result	Pass or Fail
testScaRightTriangleA	3,4,5	"Scalene and Right"	"InvalidInput"	Fail
testScaRightTriangleB	5,12,13	"Scalene and Right"	"InvalidInput"	Fail
testEqualTriangleA	6,6,6	"Equilateral"	"InvalidInput"	Fail
testEqualTriangleB	56,56,56	"Equilateral"	"InvalidIInput"	Fail
testEqualTriangleC	5.3,5.3,5.3	"Equilateral"	"InvalidIInput"	Fail
testIsoTriangleA	4,4,6	"Isosceles"	"InvalidInput"	Fail
testIsoTriangleB	46,46,25	"Isosceles"	"InvalidInput"	Fail
testIsoTriangleC	1.23,1.23,2	"Isosceles"	"InvalidInput"	Fail
testIsoRightA	2,2,sqrt(8)	"Isosceles and	"InvalidInput"	Fail

		Right”		
testIsoRightB	3,3,sqrt(18)	“Isosceles and Right”	“InvalidInput”	Fail
testScaTriangleA	7,10,14	“Scalene”	“InvalidInput”	Fail
testScaTriangleB	15,13,20	“Scalene”	“InvalidInput”	Fail
testScaTriangleC	3.56,7.2,4	“Scalene”	“InvalidInput”	Fail
testZeroInputA	0,1,2	“InvalidInput”	“InvalidInput”	Pass
testZeroInputB	1,0,2	“InvalidInput”	“InvalidInput”	Pass
testZeroInputC	2,1,0	“InvalidInput”	“InvalidInput”	Pass
testLargeInputA	201,100,150	“InvalidInput”	“InvalidInput”	Pass
testLargeInputB	100,201,150	“InvalidInput”	“InvalidInput”	Pass
testLargeInputC	100,150,201	“InvalidInput”	“InvalidInput”	Pass
testNotTriangleA	1,1,2	“NotATriangle”	“InvalidInput”	Fail
testNotTriangleB	5,6,13	“NotATriangle”	“InvalidInput”	Fail
testOrderA	5,4,3	“Scalene and Right”	“InvalidInput”	Fail
testOrderB	12,13,5	“Scalene and Right”	“InvalidInput”	Fail

Original Triangle Source Code:

C: > Users > 15165 > OneDrive - stevens.edu > SSW 567 > HW 02a > Triangle.py > classifyTriangle

```python
# -*- coding: utf-8 -*-
"""
Created on Thu Jan 14 13:44:00 2016
Updated Jan 21, 2018

The primary goal of this file is to demonstrate a simple python program to classify triangles

@author: jrr
@author: rk
"""

def classifyTriangle(a,b,c):
    """
    Your correct code goes here...  Fix the faulty logic below until the code passes all of
    you test cases.

    This function returns a string with the type of triangle from three integer values
    corresponding to the lengths of the three sides of the Triangle.

    return:
        If all three sides are equal, return 'Equilateral'
        If exactly one pair of sides are equal, return 'Isoceles'
        If no pair of  sides are equal, return 'Scalene'
        If not a valid triangle, then return 'NotATriangle'
        If the sum of any two sides equals the squate of the third side, then return 'Right'

      BEWARE: there may be a bug or two in this code
    """

    # require that the input values be >= 0 and <= 200
    if a > 200 or b > 200 or c > 200:
        return 'InvalidInput'

    if a <= 0 or b <= b or c <= 0:
        return 'InvalidInput'
```

```python

    # verify that all 3 inputs are integers
    # Python's "isinstance(object,type) returns True if the object is of the specified type
    if not(isinstance(a,int) and isinstance(b,int) and isinstance(c,int)):
        return 'InvalidInput';

    # Triangle Inequality Theorem
    # This information was not in the requirements spec but
    # is important for correctness
    # the sum of any two sides must be strictly greater than the third side
    if (a >= (b - c)) or (b >= (a - c)) or (c >= (a + b)):
        return 'NotATriangle'

    # now we know that we have a valid triangle
    if a == b and b == a:
        return 'Equilateral'
    elif ((a * 2) + (b * 2)) == (c * 2):
        return 'Right'
    elif (a != b) and  (b != c) and (a != b):
        return 'Scalene'
    else:
        return 'Isoceles'
```

Improved Test cases:

```python
import unittest
import math

from Triangle import classifyTriangle

# This code implements the unit test functionality
# https://docs.python.org/3/library/unittest.html has a nice description of the framework
from mybrand import my_brand
my_brand("SSW 567 HW 02a-Testing a legacy program and reporting on testing results")

class TestTriangles(unittest.TestCase):
    # define multiple sets of tests as functions with names that begin

    def testScaRightTriangleA(self):
        self.assertEqual(classifyTriangle(3,4,5),'Scalene and Right')

    def testScaRightTriangleB(self):
        self.assertEqual(classifyTriangle(5,12,13),'Scalene and Right')

    def testEqualTriangleA(self):
        self.assertEqual(classifyTriangle(6,6,6),'Equilateral')

    def testEqualTriangleB(self):
        self.assertEqual(classifyTriangle(56,56,56),'Equilateral')

    def testEqualTriangleC(self):
        self.assertEqual(classifyTriangle(5.3,5.3,5.3),'Equilateral')

    def testIsoTriangleA(self):
        self.assertEqual(classifyTriangle(4,4,6),'Isosceles')

    def testIsoTriangleB(self):
        self.assertEqual(classifyTriangle(46,46,25),'Isosceles')

    def testIsoTriangleC(self):
        self.assertEqual(classifyTriangle(1.23,1.23,2),'Isosceles')
```

```
    def testIsoRightA(self):
        self.assertEqual(classifyTriangle(2,2,math.sqrt(8)),'Isosceles and Right')

    def testIsoRightB(self):
        self.assertEqual(classifyTriangle(3,3,math.sqrt(18)),'Isosceles and Right')

    def testScaTriangleA(self):
        self.assertEqual(classifyTriangle(7,10,14),'Scalene')

    def testScaTriangleB(self):
        self.assertEqual(classifyTriangle(15,13,20),'Scalene')

    def testScaTriangleC(self):
        self.assertEqual(classifyTriangle(3.56, 7.2, 4),'Scalene')

    def testZeroInputA(self):
        self.assertEqual(classifyTriangle(0,1,2),'InvalidInput')

    def testZeroInputB(self):
        self.assertEqual(classifyTriangle(1,0,2),'InvalidInput')

    def testZeroInputC(self):
        self.assertEqual(classifyTriangle(2,1,0),'InvalidInput')

    def testLargeInputA(self):
        self.assertEqual(classifyTriangle(201,100,150),'InvalidInput')

    def testLargeInputB(self):
        self.assertEqual(classifyTriangle(100,201,150),'InvalidInput')

    def testLargeInputC(self):
        self.assertEqual(classifyTriangle(100,150,201),'InvalidInput')

    def testNotTriangleA(self):
        self.assertEqual(classifyTriangle(1,1,2),'NotATriangle')
```

```
        def testNotTriangleB(self):
            self.assertEqual(classifyTriangle(5,6,13),'NotATriangle')

        def testOrderA(self):
            self.assertEqual(classifyTriangle(5,4,3),'Scalene and Right')

        def testOrderB(self):
            self.assertEqual(classifyTriangle(12,13,5),'Scalene and Right')



if __name__ == '__main__':
    print('Running unit tests')
    unittest.main()
```

Part 2:

After enhancing the test code and testing it against the buggy classifyTriangle() function, update the logic in classifyTriangle() to fix all the logic bugs found in the code. Run the same test set on the improved classifyTriangle() function and create a test report on the improved logic.

First set of Improvements made to Triangle code:

- In the second if statement, the code returns "InvalidInput" if b <= b. This needed to be corrected to b <=0 because b<= b would always be true, regardless if there is an invalid input or not.
- The Triangle Inequality Theorem part needed editing to showcase that a triangle is defined by how the sum of any two sides must be strictly **greater** than the third side.
- The conditions for the equilateral triangle needed to be edited to account for equality of all three sides, not just two sides.
- The equation for the right triangle needed to be edited to show that (a^2 + b^2 = c^2), not (2a + 2b = 2c).

The first set of Improvements led to 9 failed tests out of 23 tests. This is the Test Report for first test run with the improved Triangle code:

Test ID	Input	Expected Results	Actual Result	Pass or Fail
testScaRightTriangleA	3,4,5	"Scalene and Right"	"Right"	Fail
testScaRightTriangleB	5,12,13	"Scalene and Right"	"Right"	Fail
testEqualTriangleA	6,6,6	"Equilateral"	"Equilateral"	Pass
testEqualTriangleB	56,56,56	"Equilateral"	"Equilateral"	Pass
testEqualTriangleC	5.3,5.3,5.3	"Equilateral"	"InvalidIInput"	Fail
testIsoTriangleA	4,4,6	"Isosceles"	"Isosceles"	Pass
testIsoTriangleB	46,46,25	"Isosceles"	"Isosceles"	Pass
testIsoTriangleC	1.23,1.23,2	"Isosceles"	"InvalidInput"	Fail
testIsoRightA	2,2,sqrt(8)	"Isosceles and	"InvalidInput"	Fail

		Right”		
testIsoRightB	3,3,sqrt(18)	“Isosceles and Right”	“InvalidInput”	Fail
testScaTriangleA	7,10,14	“Scalene”	“Scalene”	Pass
testScaTriangleB	15,13,20	“Scalene”	“Scalene”	Pass
testScaTriangleC	3.56,7.2,4	“Scalene”	“InvalidInput”	Fail
testZeroInputA	0,1,2	“InvalidInput”	“InvalidInput”	Pass
testZeroInputB	1,0,2	“InvalidInput”	“InvalidInput”	Pass
testZeroInputC	2,1,0	“InvalidInput”	“InvalidInput”	Pass
testLargeInputA	201,100,150	“InvalidInput”	“InvalidInput”	Pass
testLargeInputB	100,201,150	“InvalidInput”	“InvalidInput”	Pass
testLargeInputC	100,150,201	“InvalidInput”	“InvalidInput”	Pass
testNotTriangleA	1,1,2	“NotATriangle”	“NotATriangle”	Pass
testNotTriangleB	5,6,13	“NotATriangle”	“NotATriangle”	Pass
testOrderA	5,4,3	“Scalene and Right”	“Scalene”	Fail
testOrderB	12,13,5	“Scalene and Right”	“Scalene”	Fail

Analysis:

These results show that the Triangle code has yet to be improved enough to account for these properties:

- Decimal values
- Multi-faceted triangles (Scalene and Right or Isosceles and Right)
- Out of order lengths (where c isn’t always the largest value)

First draft of the Triangle Source Code:

```
@author: Jolene Ciccarone
"""

def classifyTriangle(a,b,c):

    # require that the input values be >= 0 and <= 200
    if a > 200 or b > 200 or c > 200:
        return 'InvalidInput'

    if a <= 0 or b <= 0 or c <= 0:
        return 'InvalidInput'

    # verify that all 3 inputs are integers
    # Python's "isinstance(object,type) returns True if the object is of the specified type
    if not(isinstance(a,int) and isinstance(b,int) and isinstance(c,int)):
        return 'InvalidInput';

    # Triangle Inequality Theorem
    # This information was not in the requirements spec but
    # is important for correctness
    # the sum of any two sides must be strictly greater than the third side
    if not(((b + c) > a) and ((a + c) > b) and ((a + b) > c)):
        return 'NotATriangle'

    # now we know that we have a valid triangle
    if a == b and b == c and a == c:
        return 'Equilateral'
    elif ((a ** 2) + (b ** 2)) == (c ** 2):
        return 'Right'
    elif (a != b) and  (b != c) and (a != b):
        return 'Scalene'
    else:
        return 'Isosceles'
```

Second Set of Improvements made to Triangle code:

- Eliminated the condition that only accepted integers
- Used the math.isclose() function from the math library to compare floating point values
- Added boolean variables “iso” and “right” to keep track of the isosceles and right triangles
- Added a condition that calculated the right triangle formula using all of the different combinations of the lengths
- Made different conditions to account for multi-faceted triangles (Scalene and Right vs. only Scalene Triangles

The second set of Improvements led to 0 failed tests out of 23 tests
Test Report for second test run with improved Triangle code:

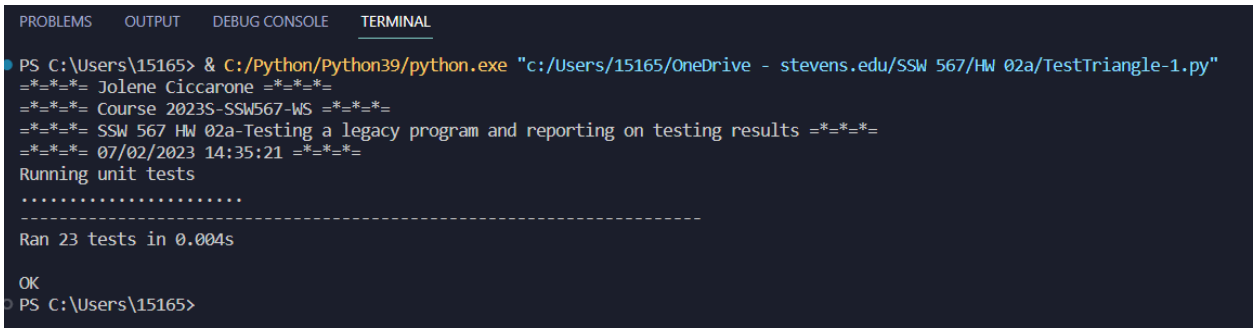
Test ID	Input	Expected Results	Actual Result	Pass or Fail
testScaRightTriangleA	3,4,5	“Scalene and Right”	“Scalene and Right”	Pass
testScaRightTriangleB	5,12,13	“Scalene and Right”	“Scalene and Right”	Pass
testEqualTriangleA	6,6,6	“Equilateral”	“Equilateral”	Pass
testEqualTriangleB	56,56,56	“Equilateral”	“Equilateral”	Pass
testEqualTriangleC	5.3,5.3,5.3	“Equilateral”	“Equilateral”	Pass
testIsoTriangleA	4,4,6	“Isosceles”	“Isosceles”	Pass
testIsoTriangleB	46,46,25	“Isosceles”	“Isosceles”	Pass
testIsoTriangleC	1.23,1.23,2	“Isosceles”	“Isosceles”	Pass
testIsoRightA	2,2,sqrt(8)	“Isosceles and Right”	“Isosceles and Right”	Pass
testIsoRightB	3,3,sqrt(18)	“Isosceles and Right”	“Isosceles and Right”	Pass
testScaTriangleA	7,10,14	“Scalene”	“Scalene”	Pass
testScaTriangleB	15,13,20	“Scalene”	“Scalene”	Pass
testScaTriangleC	3.56,7.2,4	“Scalene”	“Scalene”	Pass
testZeroInputA	0,1,2	“InvalidInput”	“InvalidInput”	Pass
testZeroInputB	1,0,2	“InvalidInput”	“InvalidInput”	Pass
testZeroInputC	2,1,0	“InvalidInput”	“InvalidInput”	Pass
testLargeInputA	201,100,150	“InvalidInput”	“InvalidInput”	Pass
testLargeInputB	100,201,150	“InvalidInput”	“InvalidInput”	Pass
testLargeInputC	100,150,201	“InvalidInput”	“InvalidInput”	Pass
testNotTriangleA	1,1,2	“NotATriangle”	“NotATriangle”	Pass

testNotTriangleB	5,6,13	“NotATriangle”	“NotATriangle”	Pass
testOrderA	5,4,3	“Scalene and Right”	“Scalene and Right”	Pass
testOrderB	12,13,5	“Scalene and Right”	“Scalene and Right”	Pass

Final improved Triangle source code:

```
import math

def classifyTriangle(a,b,c):

    # require that the input values be >= 0 and <= 200
    if a > 200 or b > 200 or c > 200:
        return 'InvalidInput'

    if a <= 0 or b <= 0 or c <= 0:
        return 'InvalidInput'

    # Triangle Inequality Theorem
    # This information was not in the requirements spec but
    # is important for correctness
    # the sum of any two sides must be strictly greater than the third side
    if not(((b + c) > a) and ((a + c) > b) and ((a + b) > c)):
        return 'NotATriangle'

    iso = False
    right = False
    # now we know that we have a valid triangle
    if (math.isclose(a,b) and math.isclose(b,c) and math.isclose(a,c)):
        return 'Equilateral'
    if (math.isclose(a,b) or math.isclose(b,c) or math.isclose(a,c)):
        iso = True
    if (math.isclose(a**2 + b**2, c**2) or math.isclose(b**2 + c**2, a**2) or math.isclose(a**2 + c**2, b**2)):
        right = True
    if iso:
        if right:
            return "Isosceles and Right"
        else:
            return "Isosceles"
    else:
        if right:
            return "Scalene and Right"
        else:
            return "Scalene"
```

Screenshot of running the test set on the improved Triangle code:



Assignment Summary:

	Test Run 1	Test Run 2
Tests Planned	23	23
Tests Executed	23	23
Tests Passed	14	23
Defects Found	9	0
Defects Fixes	9	0