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.

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 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.

Author: Alexandra Anthony

Summary:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test ID** | **Input** | **Expected Results** | **Actual Result** | **Pass or Fail** |
| 1 | 3,4,5 | Right | InvalidInput | Fail |
| 2 | 5,3,4 | Right | InvalidInput | Fail |
| 3 | 6,8,10 | Right | InvalidInput | Fail |
| 4 | 1,1,1 | Equilateral | InvalidInput | Fail |
| 5 | 2, 2, 3 | Isoceles | InvalidInput | Fail |
| 6 | 5, 5, 8 | Isoceles | InvalidInput | Fail |
| 7 | 4, 5, 6 | Scalene | InvalidInput | Fail |
| 8 | 0, 5, 6 | InvalidInput | InvalidInput | Passed |
| 9 | 201, 5, 6 | InvalidInput | InvalidInput | Passed |
| 10 | -1, 5, 6 | InvalidInput | InvalidInput | Passed |
| 11 | 4.5, 5, 6 | InvalidInput | InvalidInput | Passed |
| 12 | 1, 10, 12 | NotATriangle | InvalidInput | Fail |
| 13 | 1, 2, 3 | NotATriangle | InvalidInput | Fail |

|  |  |  |
| --- | --- | --- |
|  | Test Run 1 | Test Run2 |
|  |
|  |
| Tests Planned | 13 | 13 |  |
|  |
|  |
| Tests Executed | 13 | 13 |  |
|  |
|  |
| Tests Passed | 4 | 13 |  |
|  |
|  |
| Defects Found | 9 | 0 |  |
|  |
|  |
| Defects Fixed | 9 | 0 |  |
|  |
|  |

Reflection:

This assignment was well-paced. I think the instructions could have been a bit less repetitive and such but otherwise it was good! I learned how to present testing work as well as how to change code effectively in a testing environment.

Honor pledge: “I pledge my honor that I have abided by the stevens honor code” ~AA

Detailed results:

*# -\*- coding: utf-8 -\*-*

*"""*

*Updated Jan 21, 2018*

*The primary goal of this file is to demonstrate a simple unittest implementation*

*@author: Alexandra Anthony*

*"""*

import unittest

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*

class TestTriangles(unittest.TestCase):

*# define multiple sets of tests as functions with names that begin*

    def testRightTriangleA(*self*):

*self*.assertEqual(classifyTriangle(3,4,5), 'Right', '3,4,5 is a Right triangle') *#ID1*

    def testRightTriangleB(*self*):

*self*.assertEqual(classifyTriangle(5,3,4), 'Right', '5,3,4 is a Right triangle')*#ID2*

    def testRightTriangleC(*self*):

*self*.assertEqual(classifyTriangle(6,8,10), 'Right', '6,8,10 is a Right triangle')*#ID3*

*# Equilateral triangle tests*

    def testEquilateralTriangle(*self*):

*self*.assertEqual(classifyTriangle(1,1,1), 'Equilateral', '1,1,1 should be Equilateral')*#ID4*

*# Isosceles triangle tests*

    def testIsoscelesTriangleA(*self*):

*self*.assertEqual(classifyTriangle(2, 2, 3), 'Isoceles', '2,2,3 should be Isosceles')*#ID5*

    def testIsoscelesTriangleB(*self*):

*self*.assertEqual(classifyTriangle(5, 5, 8), 'Isoceles', '5,5,8 should be Isosceles')*#ID6*

*# Scalene triangle tests*

    def testScaleneTriangle(*self*):

*self*.assertEqual(classifyTriangle(4, 5, 6), 'Scalene', '4,5,6 should be Scalene')*#ID7*

*# Invalid input tests*

    def testInvalidInputA(*self*):

*self*.assertEqual(classifyTriangle(0, 5, 6), 'InvalidInput', '0,5,6 should be InvalidInput')*#ID8*

    def testInvalidInputB(*self*):

*self*.assertEqual(classifyTriangle(201, 5, 6), 'InvalidInput', '201,5,6 should be InvalidInput')*#ID9*

    def testInvalidInputC(*self*):

*self*.assertEqual(classifyTriangle(-1, 5, 6), 'InvalidInput', '-1,5,6 should be InvalidInput')*#ID10*

    def testInvalidInputD(*self*):

*self*.assertEqual(classifyTriangle(4.5, 5, 6), 'InvalidInput', '4.5,5,6 should be InvalidInput')*#ID11*

*# Not a triangle tests*

    def testNotATriangleA(*self*):

*self*.assertEqual(classifyTriangle(1, 10, 12), 'NotATriangle', '1,10,12 should be NotATriangle')*#ID12*

    def testNotATriangleB(*self*):

*self*.assertEqual(classifyTriangle(1, 2, 3), 'NotATriangle', '1,2,3 should be NotATriangle')*#ID13*

if \_\_name\_\_ == '\_\_main\_\_':

    print('Running unit tests')

    unittest.main()

*# -\*- 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*

*"""*

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*

*"""*

*# 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';

*# 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'

*# This information was not in the requirements spec but*

*# is important for correctness*

*# the sum of any two sides must be strictly less than the third side*

*# of the specified shape is not a triangle*

    if (*a* + *b* <= *c*) or (*b* + *c* <= *a*) or (*c* + *a* <= *b*):

        return 'NotATriangle'

*# now we know that we have a valid triangle*

    if *a* == *b* == *c*:

        return 'Equilateral'

    elif round(*a*\*\*2 + *b*\*\*2, 5) == round(*c*\*\*2, 5) or round(*b*\*\*2 + *c*\*\*2, 5) == round(*a*\*\*2, 5) or round(*c*\*\*2 + *a*\*\*2, 5) == round(*b*\*\*2, 5):

        return 'Right'

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

        return 'Isoceles'

    else:

        return 'Scalene'

**1.** Techniques Used

Black Box Testing: The focus is on testing the functionality of the classifyTriangle function without looking at structure.

* Unit Testing: Used unittest to create small test cases.

2. Assumptions and Constraints

* Assumptions:
  + The sides a, b, and c must be non-negative integers.
  + Inputs larger than 200 are considered invalid, based on existing constraints in the original code.
  + The triangle inequality theorem must hold.
* Constraints:
  + The function will return InvalidInput if any side is less than or equal to 0, or if any input is not an integer.
  + The function will return NotATriangle if the sum of any two sides is less than or equal to the third side.
  + All tests must pass and that the logic must be fixed rather than rewritten.

3. Data Inputs

* Sanity Tests: These tests include basic checks for typical triangle classifications and edge cases as seen in tables above.

4. Results and Explanations

* Test Results: The test cases helped expose defects in the original code such as incorrect input validation and triangle inequality errors. After testing and fixes, the results for each test case matched the expected outputs.
* Code Changes:
  + Input Validation: Fixed incorrect checks for invalid inputs and moved it in the structure.
  + Triangle Inequality: Correctly implemented the triangle inequality theorem to verify if the sides form a valid triangle.
  + Equilateral, Isosceles, Scalene, and Right Triangle Classification: Improved logic to correctly classify these triangle types.

5. Tools Used

* Python
* unittest Library