SSW-567 Testing, Qual. Assur. & Maint

HW-01

Zituo Yan

“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 “

# ASSIGNMENT:

The objective of this assignment is for you to (a) develop a set of tests for an existing triangle classification program, (b) use those tests to find and fix defects in that program, and (c) report on your testing results for the Triangle problem

# SUMMARY:

1. Results of running test set against the initial buggy implementation in the original ***Triangle.py:***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| TestID | Input | Expected Results | Actual Result | Pass or Fail |
| 1 | 3,4,5 | Right | InvalidInput | F |
| 2 | 5,3,4 | Right | InvalidInput | F |
| 3 | 4,5,3 | Right | InvalidInput | F |
| 4 | 1,1,1 | Equilateral | InvalidInput | F |
| 5 | 2,2,3 | Not equilateral | Not equilateral | P |
| 6 | 3,2,2 | Not equilateral | Not equilateral | P |
| 7 | 2,3,2 | Not equilateral | Not equilateral | P |
| 8 | 4,6,6 | Scalene | InvalidInput | F |
| 9 | 6,4,6 | Scalene | InvalidInput | F |
| 10 | 6,6,4 | Scalene | InvalidInput | F |
| 11 | 3,4,5 | Not scalene | Not scalene | P |
| 12 | 3,4,6 | Isosceles | InvalidInput | F |
| 13 | 2,3,5 | NotATriangle | InvalidInput | F |
| 14 | 5,3,2 | NotATriangle | InvalidInput | F |
| 15 | 2,5,3 | NotATriangle | InvalidInput | F |
| 16 | 201,2,2 | InvalidInput | InvalidInput | P |
| 17 | 2,201,2 | InvalidInput | InvalidInput | P |
| 18 | 2,2,201 | InvalidInput | InvalidInput | P |
| 19 | -2,2,2 | InvalidInput | InvalidInput | P |
| 20 | 2,-2,2 | InvalidInput | InvalidInput | P |
| 21 | 2,2,-2 | InvalidInput | InvalidInput | P |
| 22 | ‘2’,2,2 | InvalidInput | ERROR | F |
| 23 | 2.’2’,2 | InvalidInput | ERROR | F |
| 24 | 2,2,’2’ | InvalidInput | ERROR | F |

1. Test result after improvement:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| TestID | Input | Expected Results | Actual Result | Pass or Fail |
| 1 | 3,4,5 | Right | Right | P |
| 2 | 5,3,4 | Right | Right | P |
| 3 | 4,5,3 | Right | Right | P |
| 4 | 1,1,1 | Equilateral | Equilateral | P |
| 5 | 2,2,3 | Not equilateral | Not equilateral | P |
| 6 | 3,2,2 | Not equilateral | Not equilateral | P |
| 7 | 2,3,2 | Not equilateral | Not equilateral | P |
| 8 | 4,6,6 | Scalene | Scalene | P |
| 9 | 6,4,6 | Scalene | Scalene | P |
| 10 | 6,6,4 | Scalene | Scalene | P |
| 11 | 3,4,5 | Not scalene | Not scalene | P |
| 12 | 3,4,6 | Isosceles | Isosceles | P |
| 13 | 2,3,5 | NotATriangle | NotATriangle | P |
| 14 | 5,3,2 | NotATriangle | NotATriangle | P |
| 15 | 2,5,3 | NotATriangle | NotATriangle | P |
| 16 | 201,2,2 | InvalidInput | InvalidInput | P |
| 17 | 2,201,2 | InvalidInput | InvalidInput | P |
| 18 | 2,2,201 | InvalidInput | InvalidInput | P |
| 19 | -2,2,2 | InvalidInput | InvalidInput | P |
| 20 | 2,-2,2 | InvalidInput | InvalidInput | P |
| 21 | 2,2,-2 | InvalidInput | InvalidInput | P |
| 22 | ‘2’,2,2 | InvalidInput | InvalidInput | P |
| 23 | 2.’2’,2 | InvalidInput | InvalidInput | P |
| 24 | 2,2,’2’ | InvalidInput | InvalidInput | P |

|  |  |  |
| --- | --- | --- |
|  | Test Run1 | Test Run2 |
| Tests planned | 24 | 24 |
| Tests Executed | 21 | 24 |
| Test Passed | 10 | 24 |
| Defects found | 6 | 0 |
| Defects fixed | 0 | 6 |

The most important strategy for me is to test the board value of test cases. I choose to test every board value at least once to make sure facing every situation.

# RESULT:

1. Triangle.py: Implement of classify triangle:

if not (isinstance(a, int) and isinstance(b, int) and isinstance(c, int)):  
 return 'InvalidInput'  
# require that the input values be >= 0 and <= 200  
elif a > 200 or b > 200 or c > 200:  
 return 'InvalidInput'  
elif 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 >= (a + c)) or (c >= (a + b)):  
 return 'NotATriangle'  
  
edges = [a, b, c]  
edges.sort()  
# now we know that we have a valid triangle   
if a == b and b == c:  
 return 'Equilateral'  
elif ((edges[0] \*\* 2) + (edges[1] \*\* 2)) == (edges[2] \*\* 2):  
 return 'Right'  
elif len(set(edges)) == 2:  
 return 'Scalene'  
else:  
 return 'Isosceles'

1. TestTriangle.py: Automate test of Triangle.py

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')  
  
 def testRightTriangleB(self):  
 self.assertEqual(classifyTriangle(5, 3, 4), 'Right', '5,3,4 is a Right triangle')  
  
 def testRightTriangleC(self):  
 self.assertEqual(classifyTriangle(4, 5, 3), 'Right', '4,5,3 is a Right triangle')  
  
 def testEquilateralTrianglesA(self):  
 self.assertEqual(classifyTriangle(1, 1, 1), 'Equilateral', '1,1,1 should be equilateral')  
  
 def testEquilateralTrianglesB(self):  
 self.assertNotEqual(classifyTriangle(2, 2, 3), 'Equilateral', '2,2,3 is not a equilateral triangle')  
  
 def testEquilateralTrianglesC(self):  
 self.assertNotEqual(classifyTriangle(3, 2, 2), 'Equilateral', '3,2,2 is not a equilateral triangle')  
  
 def testEquilateralTrianglesD(self):  
 self.assertNotEqual(classifyTriangle(2, 3, 2), 'Equilateral', '2,3,2 is not a equilateral triangle')  
  
 def testScaleneTrianglesA(self):  
 self.assertEqual(classifyTriangle(4, 6, 6), 'Scalene', '4,6,6 should be scalene')  
  
 def testScaleneTrianglesB(self):  
 self.assertEqual(classifyTriangle(6, 4, 6), 'Scalene', '6,4,6 should be scalene')  
  
 def testScaleneTrianglesC(self):  
 self.assertEqual(classifyTriangle(6, 6, 4), 'Scalene', '6,6,4 should be scalene')  
  
 def testScaleneTrianglesD(self):  
 self.assertNotEqual(classifyTriangle(3, 4, 5), 'Scalene', '3,4,5 is not a scalene triangle')  
  
 def testIsoscelesTriangles(self):  
 self.assertEqual(classifyTriangle(3, 4, 6), 'Isosceles', '3,4,6 should be isosceles')  
  
 def testNotTriangleA(self):  
 self.assertEqual(classifyTriangle(2, 3, 5), 'NotATriangle', '2,3,5 is not a triangle')  
  
 def testNotTriangleB(self):  
 self.assertEqual(classifyTriangle(5, 3, 2), 'NotATriangle', '5,3,2 is not a triangle')  
  
 def testNotTriangleC(self):  
 self.assertEqual(classifyTriangle(2, 5, 3), 'NotATriangle', '2,5,3 is not a triangle')  
  
 def testInvalidInput1(self):  
 self.assertEqual(classifyTriangle(201, 2, 2), 'InvalidInput', '201,2,2 is invalid input')  
  
 def testInvalidInput2(self):  
 self.assertEqual(classifyTriangle(2, 201, 2), 'InvalidInput', '2,201,2 is invalid input')  
  
 def testInvalidInput3(self):  
 self.assertEqual(classifyTriangle(2, 2, 201), 'InvalidInput', '2,2,201 is invalid input')  
  
 def testInvalidInput4(self):  
 self.assertEqual(classifyTriangle(-2, 2, 2), 'InvalidInput', '-2,2,2 is invalid input')  
  
 def testInvalidInput5(self):  
 self.assertEqual(classifyTriangle(2, -2, 2), 'InvalidInput', '2,-2,2 is invalid input')  
  
 def testInvalidInput6(self):  
 self.assertEqual(classifyTriangle(2, 2, -2), 'InvalidInput', '2,2,-2 is invalid input')  
  
 def testInvalidInput7(self):  
 self.assertEqual(classifyTriangle('2', 2, 2), 'InvalidInput', "'2',2,2 is invalid input")  
  
 def testInvalidInput8(self):  
 self.assertEqual(classifyTriangle(2, '2', 2), 'InvalidInput', "2,'2',2 is invalid input")  
  
 def testInvalidInput9(self):  
 self.assertEqual(classifyTriangle(2, 2, '2'), 'InvalidInput', "2,2,'2' is invalid input")