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*** is a starter implementation of the triangle classification program.
    - ***TestTriangle.py*** **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 contains an implementation of the classifyTriangle() function with a few bugs.

TestTriangle.py contains the initial set of test cases

Part 1:

1. Review the Triangle.py file which includes the classifyTriangle() function implemented in Python.
2. Enhance the set of test cases for the Triangle problem that adequately tests the classifyTriangle() function to find problems. The test cases will be in a separate test program file called TestTriangle.py . You should not fix any bugs in Triangle.py at this time, just make changes to TestTriangle.py
3. Run your test set against the classifyTriangle() function by running:
    - python -m unittest TestTriangle
4. Create a test report in the format specified below. This report shows the results of testing the **initial** classifyTriangle() implementation.
5. Commit and push your changes to the TestTriangle.py program to GitHub

Part 2:

1. After you've completed part 1 that defines your test set and after running it against the buggy classifyTriangle() function, update the logic in classifytTriangle() to fix all of the logic bugs you found by code inspection and with your test cases.
2. Run the same test set on your improved classifyTriangle() function and create a test report on your improved logic
3. Commit and push your changes to the Triangle.py program to GitHub

Author: Chaeli Vieira
Summary:

Part 1:

Test ID	Input	Expected Results	Actual Result	Pass or Fail
testRightTriangleA	3,4,5	Right	InvalidInput	F
testRightTriangleB	5,3,4	Right	InvalidInput	F
testRightTriangleC	5,4,3	Right	InvalidInput	F
testEquilateralTrianglesA	1,1,1	Equilateral	InvalidInput	F
testEquilateralTrianglesB	2,2,2	Equilateral	InvalidInput	F
testIsocelesA	1,2,2	Isoceles	InvalidInput	F
testIsocelesB	2,2,1	Isoceles	InvalidInput	F
testIsocelesC	3,2,2	Isoceles	InvalidInput	F
testNotATriangleA	1,2,1	NotATriangle	InvalidInput	F
testNotATriangleB	1,2,3	NotATriangle	InvalidInput	F
testNotATriangleC	10,20,30	NotATriangle	InvalidInput	F
testScaleneA	7,9,8	Scalene	InvalidInput	F
testScaleneB	7,8,9	Scalene	InvalidInput	F
testScaleneC	9,8,7	Scalene	InvalidInput	F
testBadInputA	"a","a","a"	InvalidInput	InvalidInput	P
testBadInputB	-1,-1,-1	InvalidInput	InvalidInput	P
testBadInputC	200,201,200	InvalidInput	InvalidInput	P
testBadInputD	100,"a",100	InvalidInput	InvalidInput	P

Part 2:

Test Run 2:

Test ID	Input	Expected Results	Actual Result	Pass or Fail
testRightTriangleA	3,4,5	Right	**Scalene**	**F**
testRightTriangleB	5,3,4	Right	**Scalene**	**F**
testRightTriangleC	5,4,3	Right	**Scalene**	**F**
testEquilateralTrianglesA	1,1,1	Equilateral	**Equilateral**	**P**
testEquilateralTrianglesB	2,2,2	Equilateral	**Equilateral**	**P**
testIsocelesA	1,2,2	Isoceles	**Isoceles**	**P**
testIsocelesB	2,2,1	Isoceles	**Isoceles**	**P**
testIsocelesC	3,2,2	Isoceles	**Isoceles**	**P**
testNotATriangleA	1,2,1	NotATriangle	**NotATriangle**	**P**
testNotATriangleB	1,2,3	NotATriangle	**NotATriangle**	**P**
testNotATriangleC	10,20,30	NotATriangle	**NotATriangle**	**P**
testScaleneA	7,9,8	Scalene	**Scalene**	**P**
testScaleneB	7,8,9	Scalene	**Scalene**	**P**
testScaleneC	9,8,7	Scalene	**Scalene**	**P**
testBadInputA	"a","a","a"	InvalidInput	**InvalidInput**	**P**
testBadInputB	-1,-1,-1	InvalidInput	**InvalidInput**	**P**
testBadInputC	200,201,200	InvalidInput	**InvalidInput**	**P**
testBadInputD	100,"a",100	InvalidInput	**InvalidInput**	**P**

Test Run 3:

Test ID	Input	Expected Results	Actual Result	Pass or Fail
testRightTriangleA	3,4,5	Right	**Right**	**P**
testRightTriangleB	5,3,4	Right	**Right**	**P**
testRightTriangleC	5,4,3	Right	**Right**	**P**
testEquilateralTrianglesA	1,1,1	Equilateral	**Equilateral**	**P**
testEquilateralTrianglesB	2,2,2	Equilateral	**Equilateral**	**P**
testIsocelesA	1,2,2	Isoceles	**Isoceles**	**P**
testIsocelesB	2,2,1	Isoceles	**Isoceles**	**P**

testIsoscelesC	3,2,2	Isoceles	**Isoceles**	**P**
testNotATriangleA	1,2,1	NotATriangle	**NotATriangle**	**P**
testNotATriangleB	1,2,3	NotATriangle	**NotATriangle**	**P**
testNotATriangleC	10,20,30	NotATriangle	**NotATriangle**	**P**
testScaleneA	7,9,8	Scalene	**Scalene**	**P**
testScaleneB	7,8,9	Scalene	**Scalene**	**P**
testScaleneC	9,8,7	Scalene	**Scalene**	**P**
testBadInputA	"a","a","a"	InvalidInput	**InvalidInput**	**P**
testBadInputB	-1,-1,-1	InvalidInput	**InvalidInput**	**P**
testBadInputC	200,201,200	InvalidInput	**InvalidInput**	**P**
testBadInputD	100,"a",100	InvalidInput	**InvalidInput**	**P**

	Test Run 1	Test Run 2	Test Run 3
Tests Planned	18	18	18
Tests Executed	18	18	18
Tests Passed	4	15	18
Defects Found	4	1	0
Defects Fixed	0	4	1

Repo: https://github.com/chaelivieira/SSW567HW2a

Reflection:

I learned that sometimes you have to do multiple test runs in order to find all the bugs in the code. I had to do three test runs in order to get all the bugs in the original code. Its also important to test different combinations of the same numbers in some cases.

Honor pledge:

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

Detailed results:

The technique I used was after I created my tests, I used an online triangle calculator to ensure that all my tests were accurate. Then I had to print all results using the same inputs I used in my test cases in order to see what results I was actually getting, rather than just the error message my test cases provided. By doing this I was able to find that the way the original code checked the inputs was incorrect so I was able to spot the error. I also looked through the rest of the code and made any changed I saw would be an issue. I then ran my second set of tests, which I found that there was still some bugs. I went back and found the errors in my right triangle logic and then was able to run a third and final test, in which all test cases passed.

One assumption that I made in the code was that the order in which the inputs of sides should not matter (ex: (3,4,5), (5,4,3), (4,5,3) should all have the same output) and I adjusted the code accordingly

My data inputs were:

```
print (classifyTriangle(3,4,5))
 print (classifyTriangle(5,3,4))
 print (classifyTriangle(5,4,3))
 print (classifyTriangle(1,1,1))
 print (classifyTriangle(2,2,2))
 print (classifyTriangle(1,2,2))
 print (classifyTriangle(2,2,1))
 print (classifyTriangle(3,2,2))
 print (classifyTriangle(1,2,1))
 print (classifyTriangle(1,2,3))
 print (classifyTriangle(10,20,30))
 print (classifyTriangle(7,9,8))
 print (classifyTriangle(7,8,9))
 print (classifyTriangle(9,8,7))
 print (classifyTriangle("a","a","a"))
 print (classifyTriangle(-1,-1,-1))
 print (classifyTriangle(200,201,200))
 print (classifyTriangle(100,"a",100))
```

The screenshots of the results of my code are included in the zip file as well as in the git repo.