

Homework 00b

Name: Vanshaj Tyagi

Course: SSW-567-A

Question:

Think back to your elementary school geometry class where you learned about scalene, isosceles, equilateral, and right triangles. Recall that:

- **equilateral** triangles have all three sides with the same length
- **isosceles** triangles have two sides with the same length
- **scalene** triangles have three sides with different lengths
- **right** triangles have three sides with lengths, a , b , and c where $a^2 + b^2 = c^2$

Your assignment is to write a program in Python to classify triangles and use an automated test platform, e.g. unittest or pytest, and write test cases to test your implementation of classifying triangles. The goal is for you to gain experience using automated test tools and to think through the issues associated with testing a "system".

These are your Requirements Specifications for this program:

*“Write a function **classify_triangle()** that takes three parameters: a , b , and c . The three parameters represent the lengths of the sides of a triangle. The function returns a string that specifies whether the triangle is scalene, isosceles, or equilateral, and whether it is a right triangle as well.”*

Hint: Write a function called `classify_triangle(a, b, c)` where a , b , and c are the lengths of the sides of the triangles. You may either allow the user to enter values that you pass to `classify_triangle()` or your "main" routine can just invoke `classify_triangle()` with values. This approach allows you to easily invoke `classify_triangle()` from your test framework.

You will not be graded on how well you write the program, but do try to do a decent job. You may want to include a few bugs in your program to demonstrate that your test script is working properly and is able to find problems.

You'll find a [partial solution](#) in Canvas to show how to use Python's unittest framework. Your solution should be managed using GitHub within a new repository created just for this assignment.

Assignment Deliverables:

Deliverable 1: Upload the file(s) with Python source code for your classify triangle solution to Canvas. The file(s) should include the source code used to solve the problem and the test cases for the code. Your test cases should demonstrate that you've adequately tested your solution.

Deliverable 2: Upload a text file or screen shot to show the input and output of running the program and demonstrating that your program has been adequately tested.

Deliverable 3: Describe your experience with this assignment, specifically:

- What challenges did you encounter with this assignment, if any?
- What did you think about the requirements specification for this assignment?
- What challenges did you encounter with the tools?
- Describe the criteria you used to determine that you had sufficient test cases, i.e. how did you know you were done?

Deliverable 4: Submit the URL of the GitHub repo containing your complete solution. The files in the repo should match what you have uploaded to Canvas.

Answer:

1. All files are uploaded to Canvas and can be found on Github: [Link](#)
2. SS for invalid test case code examples: 1 function

```
class TestTriangleClassification:
    def test_triangle_cases(self):
        """Test various triangle cases including edge cases"""
        assert classify_triangle(-1, 2, 2) == "Invalid input: All sides must be positive"
        assert classify_triangle(1, 2, 4) == "Not a triangle"
        assert classify_triangle(0, 0, 0) == "Invalid input: All sides must be positive"
```

SS for valid test case code examples: 4 functions

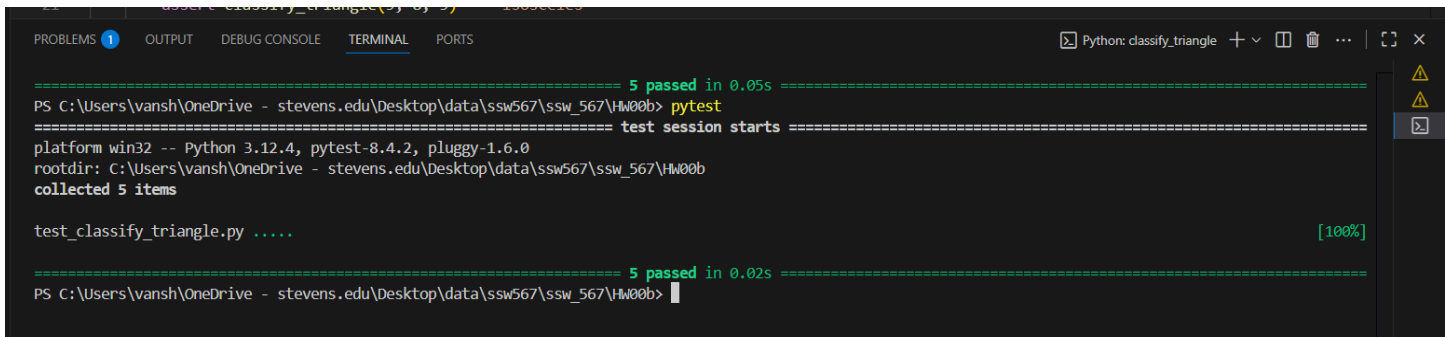
```
def test_equilateral_triangles(self):
    assert classify_triangle(5, 5, 5) == "Equilateral"
    assert classify_triangle(10, 10, 10) == "Equilateral"
    assert classify_triangle(0.5, 0.5, 0.5) == "Equilateral"
    assert classify_triangle(100, 100, 100) == "Equilateral"
    assert classify_triangle(7, 7, 7) == "Equilateral"

def test_isosceles_triangles(self):
    """Test isosceles triangles (two sides equal)"""
    # Test all permutations of equal sides
    assert classify_triangle(5, 5, 8) == "Isosceles"
    assert classify_triangle(5, 8, 5) == "Isosceles"
    assert classify_triangle(8, 5, 5) == "Isosceles"
    assert classify_triangle(3, 3, 4) == "Isosceles"
    assert classify_triangle(10, 6, 10) == "Isosceles"
    assert classify_triangle(7, 10, 7) == "Isosceles"

def test_scalene_triangles(self):
    """Test scalene triangles (all sides different)"""
    assert classify_triangle(3, 7, 5) == "Scalene"
    assert classify_triangle(4, 6, 8) == "Scalene"
    assert classify_triangle(2, 3, 4) == "Scalene"
    assert classify_triangle(6, 10, 14) == "Scalene"
    assert classify_triangle(5, 9, 7) == "Scalene"

def test_right_triangles(self):
    """Test right triangles (satisfy Pythagorean theorem)"""
    # Classic Pythagorean triples
    assert classify_triangle(3, 4, 5) == "Scalene Right"
    assert classify_triangle(4, 3, 5) == "Scalene Right"
    assert classify_triangle(5, 3, 4) == "Scalene Right"
```

Output of the code



```
===== 5 passed in 0.05s =====
PS C:\Users\vansh\OneDrive - stevens.edu\Desktop\data\ssw567\ssw_567\HW00b> pytest
===== test session starts =====
platform win32 -- Python 3.12.4, pytest-8.4.2, pluggy-1.6.0
rootdir: C:\Users\vansh\OneDrive - stevens.edu\Desktop\data\ssw567\ssw_567\HW00b
collected 5 items

test_classify_triangle.py ..... [100%]

===== 5 passed in 0.02s =====
PS C:\Users\vansh\OneDrive - stevens.edu\Desktop\data\ssw567\ssw_567\HW00b>
```

3. Experience with this assignment

- 3.1 The primary challenge was handling floating-point precision in right triangle detection, solved using a small tolerance value ($1e-10$). Ensuring comprehensive test coverage required careful consideration of boundary conditions and edge cases.
- 3.2 The specification was clear but left room for interpretation regarding error handling and return format. I chose descriptive string returns that combine classifications (e.g., "Scalene Right").
- 3.3 I determined test sufficiency using equivalence partitioning (one test per triangle type), boundary value analysis (edge cases near triangle inequality), and error condition coverage. The test suite includes parametrized tests and multiple assertion patterns to ensure robustness.
- 3.4 PyTest proved excellent for organized testing with clear class-based structure and descriptive test names. The framework's assertion handling and test discovery made development efficient. The implementation demonstrates thorough understanding of geometric principles, software testing methodologies, and defensive programming practices. The test suite would effectively catch bugs and regressions in future modifications.

4. [Github Link](#)