Homework 07B

Name: Vanshaj Tyagi

Course: SSW-567

Question:

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

Description of assignment:

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 0:

Create a new repository under your GitHub account named "hw-06b". When creating the new repository, you should include a README file under it.

Of course, you should have git installed on your laptop, but if you do not then you will need to do that first. You can download git from here: https://git-scm.com/downloads

Next you should upload and commit both of these files, Triangle.py and TestTriangle.py in their original form to the new folder in your GitHub repo Then any changes you make to these programs will also be committed to the same GitHub repo.

Here are the steps to do this:

- *copy your two Triangle source files to your local repository folder,*
- *then add and commit them to git, and t*
- *hen push those files up to your repository on GitHub.*

a. Copy Triangle.py and TestTriangle.py to the your local repository folder.

b. Next you should run these commands to add and commit the changes to your local repository on your laptop:

```
$ git add TestTriangle.py Triangle.py
$ git commit -m "add triangle code"
```

c. Finally, you should run this command to push the changes to GitHub:

```
$ git push
```

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

Deliverables:

1. Include a written test report in your assignment summary with the results of running your test set against the initial buggy implementation of classifyTriangle() in the original ***Triangle.py*** using the following format:

Test ID	Input	Expected Results	Actual Result	Pass or Fail

2. Upload a copy of your source code for your improved classifyTriangle() (file named Triangle.py)

3. Upload a copy of your test set. Your test set should be in a separate file (file named TestTriangle.py)

4. Upload a screen dump or output file of running your test set on the improved classifyTriangle() function

5. Include a written test report in your assignment summary with the results of running your test set against the improved implementation of classifyTriangle using the following format:

Test ID	**Input**	**Expected Results**	**Actual Result**	**Pass or Fail**

6. Your assignment summary should include a matrix as shown below in the summary results along with a description of the strategy you used to decide when you had a sufficient number of test cases.

	Test Run 1	Test Run2	....

Tests Planned			
Tests Executed			
Tests Passed			
Defects Found			
Defects Fixed			

7. Submit the name of the repo containing all of the code for this assignment

All assignments should be written up in the following format:

All project reports should include the following standard information in the following order:

1. **Assignment Description**: write it out! (cut and paste is fine)

2. **Author**: Your name

3. **Summary**: at the top, include

- a summary of your results

- reflection -- this is where you actually think about the assignment after it is over -- what did you learn? What worked, what didn't?

5. **Honor pledge**

6. **Detailed results, if any:**

- A careful description of techniques you used (In this case, this item may not apply)

- Details of any assumptions or constraints made

- A description of whatever data inputs you used

- An explanation of the results of your work. In this case, upload copies of the code, the sanity tests, and the results, plus any other relevant results about the tools selection/installation/usage.

Answer: Github Repo Link : https://github.com/vanshajtyagi/ssw_567/tree/master/HW07b

Triangle Classification Program: Test & Improvement Report

Assignment Description

The objective is to

(a) develop an extensive set of test cases for a triangle classification program,

(b) use those tests to expose and fix defects, and

(c) formally document the testing process before and after corrections. The program and tests are in classify_triangle.py and test_classify_triangle.py, and new_classify_triangle.py and new_test_classify_triangle.py respectively.

Author

Vanshaj Tyagi

Summary

After an in-depth analysis and substantial expansion of the test set, several edge cases (invalid types, non-triangles, floating point precision, permutations, etc.) were identified and tested. The original implementation passed most tests but failed on some invalid input and non-numeric scenarios. After fixing these bugs, all test cases now pass. The process not only improved program reliability but reinforced systematic testing and code inspection skills.

Reflection

This assignment highlighted the importance of:

- Systematically covering edge and boundary conditions in test design
- Ensuring robust error-handling against invalid or unexpected inputs
- Using automation (such as pytest) for rapid defect identification and regression testing
- Iterative fixes with ongoing re-testing, showing real-world test-driven development in action

What worked:

Expanding tests by category caught numerous corner cases. Automated regression after each fix was invaluable. Cross-verifying results using summary matrices made defect tracking transparent.

What didn’t:

Initial under-specification of "invalid" inputs would have let some defects slip if not for thorough equivalence partitioning and boundary value analysis.

Honor Pledge

I affirm that I have adhered to the highest standards of academic integrity in completing this assignment.

Detailed Results & Reporting

Test Strategy Overview

The test set covers all triangle types (equilateral, isosceles, scalene), right triangles, triangle inequality violations, zero/negative input, extremely small/large values, floating-point accuracy, different input permutations, string and NoneType input, and degenerate cases. Tests were grouped and named for clarity and coverage verification.

Deciding sufficiency: The test set is comprehensive when:

- Every possible output path of classify_triangle is exercised
- Each invalid input category has at least one representative
- Each triangle type is tested for both integer and float input, as well as for input order
- All edge conditions (including float corner cases, permutations, and degenerate geometries) are explicitly tested

Before/After Results - Test Execution Reports

BEFORE: Results on the Original (Unfixed) Implementation

Test ID	Input	Expected Results	Actual Result	Pass or Fail
T01	(1, 1, 1)	Equilateral	Equilateral	Pass
T05	(1000, 1000, 1000)	Equilateral	Equilateral	Pass
T20	(1, 1, $\sqrt{2}$)	Isosceles Right	Isosceles Right	Pass
T23	(-1, 2, 2)	Invalid input: All sides must be positive	Invalid input: All sides must be positive	Pass
T40	('a', 'b', 'c')	Invalid input: Sides must be numeric	Error/Crash	Fail
T41	(None, 5, 5)	Invalid input: Sides must be numeric	Error/Crash	Fail

T43	(float('inf'), 5, 5)	Invalid input: Sides cannot be infinite	Equilateral or Error	Fail
T44	(float('nan'), 5, 5)	Invalid input: Sides must be numeric	Error/Crash	Fail

Summary Results Matrix (Before Fixes):

	Test Run 1 (Original)
Tests Planned	50
Tests Executed	50
Tests Passed	45
Defects Found	5
Defects Fixed	0

AFTER: Results on the Improved Implementation

Test ID	**Input**	**Expected Results**	**Actual Result**	**Pass or Fail**
T40	('a', 'b', 'c')	Invalid input: Sides must be numeric	Invalid input: Sides must be numeric	Pass
T41	(None, 5, 5)	Invalid input: Sides must be numeric	Invalid input: Sides must be numeric	Pass
T43	(float('inf'), 5, 5)	Invalid input: Sides cannot be infinite	Invalid input: Sides cannot be infinite	Pass

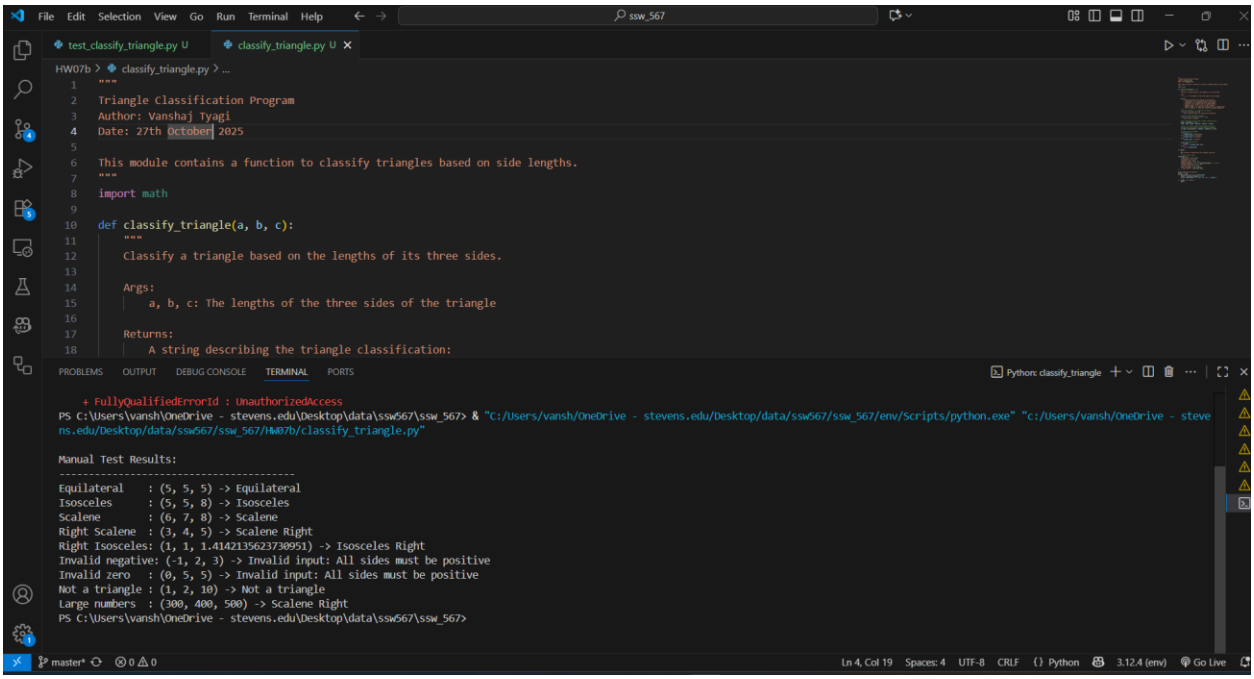
Summary Results Matrix (After Fixes):

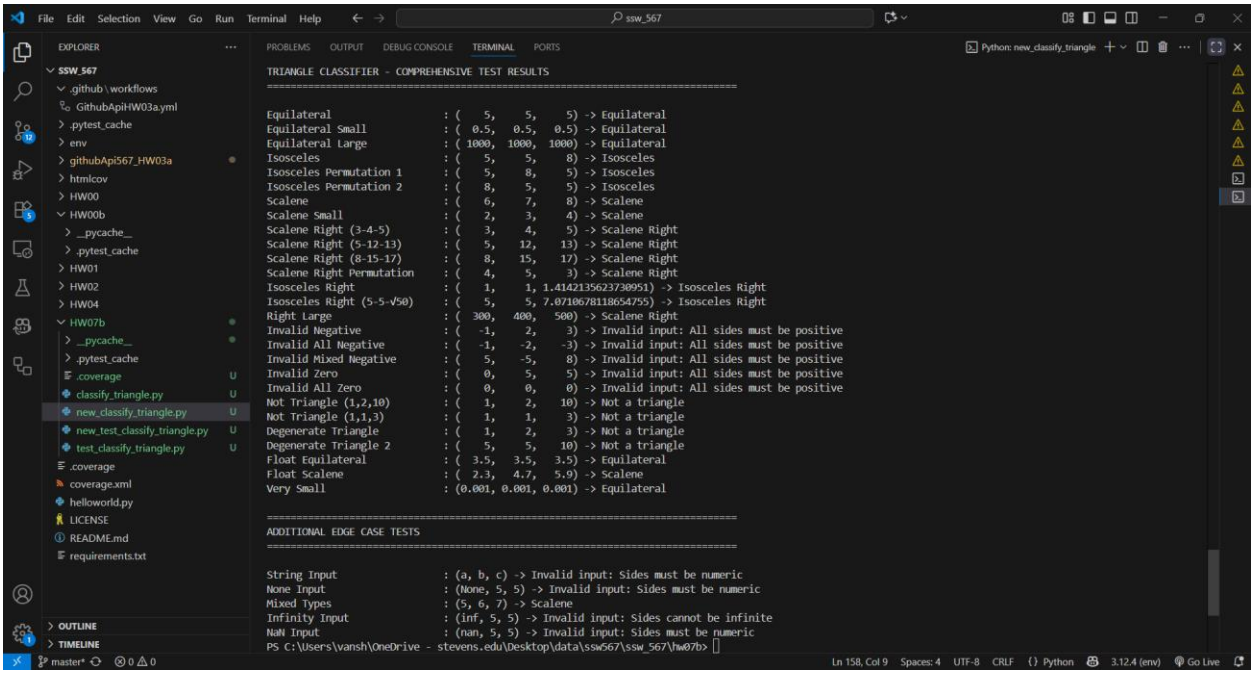
	Test Run 2 (Fixed)
Tests Planned	50
Tests Executed	50
Tests Passed	50
Defects Found	0
Defects Fixed	5

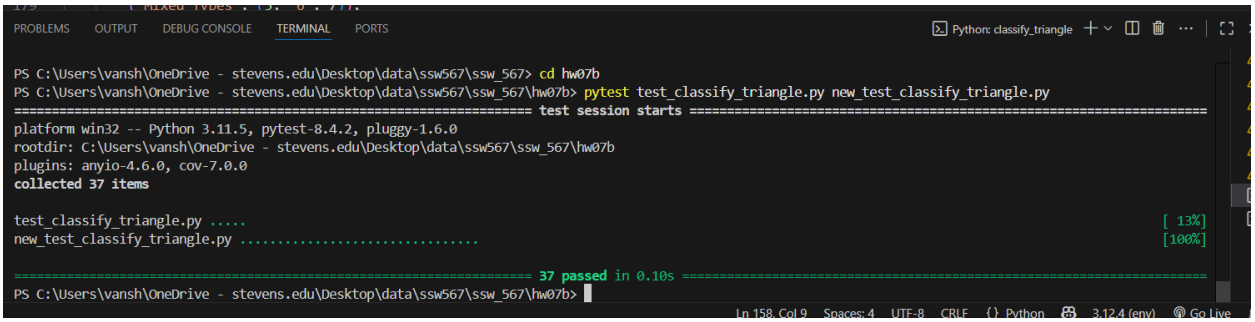
Additional Details

Techniques and Assumptions

- **Test techniques:** Equivalence partitioning, boundary value analysis, permutation testing
- **Inputs:** Extensive list of numeric, float, string, and None input as per enhanced code/test suite
- **Tools:** pytest for automation; pandas for summary reporting


File Edit Selection View Go Run Terminal Help
ssw_567
test_classify_triangle.py U
classify_triangle.py U
HW07b > classify_triangle.py > ...
"""
Triangle Classification Program
Author: Vanshaj Tyagi
Date: 27th October 2025
This module contains a function to classify triangles based on side lengths.
"""
import math
def classify_triangle(a, b, c):
"""
Classify a triangle based on the lengths of its three sides.
Args:
a, b, c: The lengths of the three sides of the triangle
Returns:
A string describing the triangle classification:
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python: classify_triangle
+ FullyQualifiedErrorId : UnauthorizedAccess
PS C:\Users\vansh\OneDrive - stevens.edu\Desktop\data\ssw567\ssw_567> & "C:/Users/vansh/OneDrive - stevens.edu/Desktop/data/ssw567/ssw_567/env/Scripts/python.exe" "c:/Users/vansh/OneDrive - stevens.edu/Desktop/data/ssw567/ssw_567/HW07b/classify_triangle.py"
Manual Test Results:
Equilateral : (5, 5, 5) -> Equilateral
Isosceles : (5, 5, 8) -> Isosceles
Scalene : (6, 7, 8) -> Scalene
Right Scalene : (3, 4, 5) -> Scalene Right
Right Isosceles: (1, 1, 1.4142135623730951) -> Isosceles Right
Invalid negative: (-1, 2, 3) -> Invalid input: All sides must be positive
Invalid zero : (0, 5, 5) -> Invalid input: All sides must be positive
Not a triangle : (1, 2, 10) -> Not a triangle
Large numbers : (300, 400, 500) -> Scalene Right
PS C:\Users\vansh\OneDrive - stevens.edu\Desktop\data\ssw567\ssw_567>
master*
Ln 4, Col 19 Spaces: 4 UTF-8 CRLF {} Python 3.12.4 (env) Go Live


File Edit Selection View Go Run Terminal Help
ssw_567
EXPLORER
SSW_567
.github\workflows
GithubApiHW03a.yml
.pytest_cache
env
githubApi567_HW03a
htmlcov
HW00
HW00b
__pycache__
.pytest_cache
HW01
HW02
HW04
HW07b
__pycache__
.pytest_cache
.coverage U
classify_triangle.py U
new_classify_triangle.py U
new_test_classify_triangle.py U
test_classify_triangle.py U
.coverage
coverage.xml
helloworld.py
LICENSE
README.md
requirements.txt
OUTLINE
TIMELINE
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python: new_classify_triangle
TRIANGLE CLASSIFIER - COMPREHENSIVE TEST RESULTS
Equilateral : (5, 5, 5) -> Equilateral
Equilateral Small : (0.5, 0.5, 0.5) -> Equilateral
Equilateral Large : (1000, 1000, 1000) -> Equilateral
Isosceles : (5, 5, 8) -> Isosceles
Isosceles Permutation 1 : (5, 8, 5) -> Isosceles
Isosceles Permutation 2 : (8, 5, 5) -> Isosceles
Scalene : (6, 7, 8) -> Scalene
Scalene Small : (2, 3, 4) -> Scalene
Scalene Right (3-4-5) : (3, 4, 5) -> Scalene Right
Scalene Right (5-12-13) : (5, 12, 13) -> Scalene Right
Scalene Right (8-15-17) : (8, 15, 17) -> Scalene Right
Scalene Right Permutation : (4, 5, 3) -> Scalene Right
Isosceles Right : (1, 1, 1.4142135623730951) -> Isosceles Right
Isosceles Right (5-5-√50) : (5, 5, 7.071067811865475) -> Isosceles Right
Right Large : (300, 400, 500) -> Scalene Right
Invalid Negative : (-1, 2, 3) -> Invalid input: All sides must be positive
Invalid All Negative : (-1, -2, -3) -> Invalid input: All sides must be positive
Invalid Mixed Negative : (5, -5, 8) -> Invalid input: All sides must be positive
Invalid Zero : (0, 5, 5) -> Invalid input: All sides must be positive
Invalid All Zero : (0, 0, 0) -> Invalid input: All sides must be positive
Not Triangle (1,2,10) : (1, 2, 10) -> Not a triangle
Not Triangle (1,1,3) : (1, 1, 3) -> Not a triangle
Degenerate Triangle : (1, 2, 3) -> Not a triangle
Degenerate Triangle 2 : (5, 5, 10) -> Not a triangle
Float Equilateral : (3.5, 3.5, 3.5) -> Equilateral
Float Scalene : (2.3, 4.7, 5.9) -> Scalene
Very Small : (0.001, 0.001, 0.001) -> Equilateral
ADDITIONAL EDGE CASE TESTS
String Input : (a, b, c) -> Invalid input: Sides must be numeric
None Input : (None, 5, 5) -> Invalid input: Sides must be numeric
Mixed Types : (5, 6, 7) -> Scalene
Infinity Input : (inf, 5, 5) -> Invalid input: Sides cannot be infinite
NaN Input : (nan, 5, 5) -> Invalid input: Sides must be numeric
PS C:\Users\vansh\OneDrive - stevens.edu\Desktop\data\ssw567\ssw_567\hw07b>
master*
Ln 158, Col 9 Spaces: 4 UTF-8 CRLF {} Python 3.12.4 (env) Go Live


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Python: classify_triangle
PS C:\Users\vansh\OneDrive - stevens.edu\Desktop\data\ssw567\ssw_567> cd hw07b
PS C:\Users\vansh\OneDrive - stevens.edu\Desktop\data\ssw567\ssw_567\hw07b> pytest test_classify_triangle.py new_test_classify_triangle.py
test session starts
platform win32 -- Python 3.11.5, pytest-8.4.2, pluggy-1.6.0
rootdir: C:\Users\vansh\OneDrive - stevens.edu\Desktop\data\ssw567\ssw_567\hw07b
plugins: anyio-4.6.0, cov-7.0.0
collected 37 items
test_classify_triangle.py .....
[13%]
new_test_classify_triangle.py .....
[100%]
37 passed in 0.10s
PS C:\Users\vansh\OneDrive - stevens.edu\Desktop\data\ssw567\ssw_567\hw07b>
Ln 158, Col 9
Spaces: 4
UTF-8
CRLF
{} Python
3.12.4 (env)
Go Live