1. Assignment Description

Project Objectives

- Identify appropriate test strategies for the Triangle problem
- Create and run a test set that you feel adequately tests the program logic in the classifyTriangle() function in buggyTriangle.py and upload your initial findings in a formal test report described below.
- Fix all of the bugs you find in the classifyTriangle() function in buggyTriangle.py Don't simply replace my 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.

To Do:

1. Download and review the ***buggyTriangle.py*** file which includes the classifyTriangle() function implemented in Python. You may choose either to use the Python code or you may replicate the Python logic in the language of your choice, e.g. Java. If you choose to use some other language, you can just copy your code from Assignment 1 ***BUT*** replace your classifyTriangle() logic with the logic from buggyTriangle.py.
2. Create a set of test cases for the Triangle problem that adequately tests the classifyTriangle() function to find problems.
3. Run your test set against the classifyTriangle() function and upload a test report in the format specified below. This report shows the results of testing the initial classifyTriangle() implementation.
4. After you've completed defining your test set and 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.
5. Run the same test set on your improved classifyTriangle() function and upload a test report on your improved logic

Deliverables:

1. Include a written test report in your assignment summary with the results of running your test set against the buggy implementation of classifyTriangle in ***buggyTriangle.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()
3. Upload a copy of your test set. Your test set may be part of the same source code file with classifyTriangle() or a separate file.
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.

2. Authors

Philip Warmuz, Constantine Davantzis, Julie Traweek

3. Summary

The logic in buggyTriangle had a significant effect on the test cases. The logic essentially contributed to failing all tests. The test cases were sound so a comparison between the original buggyTriangle.py and fixedTriangle.py gave us verification.

Initial Test Run on buggyTriangle.py

Not all expected results are listed because of the depth some test cases

Test ID	Input	Expected Results	Actual Result	Pass or Fail
test_case_00	3,4,5	'Right'	AssertionError	Fail
test_case_01	1,1,1	'Equilateral'	AssertionError	Fail
test_case_02	10,10,10 != 10,10,8 =	'Isosceles'	AssertionError	Fail
test_case_03	8, 6, 7	'Scalene'	AssertionError	Fail
test_case_04	195, 10, 201 10, 201, 195 201, 195, 10	'InvalidInput'	'InvalidInput'	Pass
test_case_05	-2, 2, 3 2, -2, 3 2, 2, -3 0, 2, 3 2, 0, 3 2, 2, 0	'InvalidInput'	'InvalidInput'	Pass
test_case_06	"3", 3, 3 3, "3", 3 3, 3, "3" 5.5, 5.5, 5.5 2.5, 6, 6.5 8, 8, 9.2 3.5, 4.5, 6	'InvalidInput'	InvalidInput'	Pass
test_case_07	11, 6, 4 6, 11, 4 6, 4, 11	'NotATriangle'	AssertionError	Fail
test_case_08	3,4,5 4,5,3, 5,3,4	'Right'	AssertionError	Fail

Second Test Run on fixedTriangle.py

Not all expected results are listed because of the depth some test cases

Test ID	Input	Expected Results	Actual Result	Pass or Fail
test_case_00	3,4,5	‘Right’	‘Right’	Pass
test_case_01	1,1,1	‘Equilateral’	‘Equilateral’	Pass
test_case_02	10,10,10 != 10,10,8 =	‘Isosceles’	!= ‘Equilateral’ = ‘Isosceles’	Pass
test_case_03	8, 6, 7	‘Scalene’	‘Scalene’	Pass
test_case_04	195, 10, 201 10, 201, 195 201, 195, 10	‘InvalidInput’	‘InvalidInput’	Pass
test_case_05	-2, 2, 3 2, -2, 3 2, 2, -3 0, 2, 3 2, 0, 3 2, 2, 0	‘InvalidInput’	‘InvalidInput’	Pass
test_case_06	“3”, 3, 3 3, “3”, 3 3, 3, “3” 5.5, 5.5, 5.5 2.5, 6, 6.5 8, 8, 9.2 3.5, 4.5, 6	‘InvalidInput’	‘InvalidInput’	Pass
test_case_07	11, 6, 4 6, 11, 4 6, 4, 11	‘NotATriangle’	‘NotATriangle’	Pass
test_case_08	3,4,5 4,5,3, 5,3,4	‘Right’	‘Right’	Pass

Test Case Strategy Summary

	Tests Planned	Tests Executed	Tests Passed	Defects Found	Defects Fixed
Test Run 1	9	9	3	5	0
Test Run 2	9	9	9	0	5

Reflection

The team decided we had a sufficient number of test cases when we were able to reach each point in the triangle function that returns a value, and the tests passed. The team additionally experimented with boundary limits and input types to ensure we covered the cases where the function input was invalid.

Additionally the team noticed that despite tests passing during the first test run, the logic that reaches an “InvalidInput” may not be correct for that specific test case. A better program would include better defined errors so that the test cases could ensure the input was marked as invalid for the correct reason.

4. Team Member Roles and Contributions

Phil:Initial fixedTriangle.py with corrections, some test cases, some document writing
Constantine: Refactored code, fixed function import bug, some test cases, some documentation writing
Julie: Some test cases, some documentation writing

5. Honor Pledge

“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.”

6. Detailed Results

A careful description of techniques you used

In order to allow everyone to contribute and have separate results the original buggyTriangle code was duplicated into fixedTriangle.py in order to perform the logic corrections. Some code was altered on both files to allows us to link our test case file to both instances. The logic was not changed in buggyTriangle.py to preserve it’s intent.

Details of any assumptions or constraints made

We knew the code was going to be buggy so we assumed we would need to read it carefully and make sure both the logic and syntax was correct.

A description of whatever data inputs you used

Comments were left to describe what the test case is supposed to do. Comments were left in fixedTriangle.py to document corrections. There may have been a small correction that wasn’t documented, it’s common to just fix something small and move on.