- https://github.com/naccardo3/HW2A/tree/main
- The primary goal was to develop a set of tests for an existing triangle classification program, identify and fix bugs, and report the testing results for the Triangle problem. The process involved:
    - Enhancing an existing test suite to adequately test the triangle classification program (classifyTriangle() function).
    - Running the tests against the original (buggy) triangle program to identify any defects.
    - Updating the classifyTriangle() function to resolve the defects found in the initial implementation.
    - Running the test cases again to verify that all bugs were fixed and reporting the final results.
- Nicolas Accardo
- Summary
    - Summary of Results: Initially, most of the test cases failed due to incorrect input validation and faulty logic in the classifyTriangle() function. After enhancing the test cases I identified bugs related to the triangle inequality rule, right triangle classification, and equilateral/isosceles checks. I updated the function to fix these issues after which all test cases passed successfully.
    - Reflection: This assignment was a valuable exercise in debugging and test-driven development. I learned how critical it is to create comprehensive test cases that cover all possible scenarios, including edge cases. Initially, some of my test cases failed, which helped me identify and fix multiple logic errors in the original implementation. The experience also emphasized the importance of input validation and adhering to mathematical rules (e.g., the triangle inequality). The process of gradually fixing the function based on failing tests was rewarding. What worked well was my approach to iteratively improving the function based on specific test failures. However, I found that understanding the exact nature of some bugs (e.g., triangle inequality) took more time than expected.
- Honor Pledge
    - I pledge my honor that I have abided by the Stevens Honor System
- Detailed Results
    - Techniques Used: I followed a test-driven development (TDD) approach by writing comprehensive test cases to cover all possible triangle classifications. Once the tests were in place, I used the failing test results to identify bugs in the classifyTriangle() function. I iteratively updated the function and re-ran the tests until all tests passed.
    - Assumptions and Constraints: The key assumption made was that the inputs are expected to be integers between 1 and 200, and any input outside this range or non-integer inputs should return InvalidInput. Additionally, the triangle inequality rule was assumed to hold for determining whether the sides form a valid triangle.
    - Description of Data Inputs: The input to the function is three integer values representing the sides of a triangle. Valid inputs must meet the condition that all sides are positive integers, and the sum of any two sides must be greater than the third side for the triangle to be classified.
    - Explanation of Results: After identifying bugs in the original classifyTriangle() function, I made updates to the logic to correctly handle input validation and classification for different triangle types. The function now correctly identifies and classifies equilateral, isosceles, scalene, and right triangles, and it properly handles invalid inputs.

Failed Test Cases				
Test ID	Input	Expected Result	Actual Results	Pass or Fail
testRightTriangle	(3, 4, 5)	Right	InvalidInput	Fail
testRightTriangle	(5, 3, 4)	Right	InvalidInput	Fail
testEquilateral	(1, 1, 1)	Equilateral	InvalidInput	Fail
testIsoscelesA	(2, 2, 3)	Isosceles	InvalidInput	Fail
testIsoscelesB	(4, 4, 2)	Isosceles	InvalidInput	Fail
testScaleneA	(4, 5, 6)	Scalene	InvalidInput	Fail
testScaleneB	(10, 8, 7)	Scalene	InvalidInput	Fail
testInvalidTriang	(1, 1, 3)	NotATriangle	InvalidInput	Fail
testInvalidTriang	(10, 1, 1)	NotATriangle	InvalidInput	Fail
testInvalidInputA	(0, 1, 1)	InvalidInput	InvalidInput	Pass
testInvalidInputB	(-1, 1, 1)	InvalidInput	InvalidInput	Pass

Passed Test Cases				
Test ID	Input	Expected Result	Actual Results	Pass or Fail
testRightTriangle	(3, 4, 5)	Right	Right	Pass
testRightTriangle	(5, 3, 4)	Right	Right	Pass
testEquilateral	(1, 1, 1)	Equilateral	Equilateral	Pass
testIsoscelesA	(2, 2, 3)	Isosceles	Isosceles	Pass
testIsoscelesB	(4, 4, 2)	Isosceles	Isosceles	Pass
testScaleneA	(4, 5, 6)	Scalene	Scalene	Pass
testScaleneB	(10, 8, 7)	Scalene	Scalene	Pass
testInvalidTriang	(1, 1, 3)	NotATriangle	NotATriangle	Pass
testInvalidTriang	(10, 1, 1)	NotATriangle	NotATriangle	Pass
testInvalidInputA	(0, 1, 1)	InvalidInput	InvalidInput	Pass
testInvalidInputB	(-1, 1, 1)	InvalidInput	InvalidInput	Pass

	Test Run 1	Test Run 2	Test Run 3	Test Run 4
Tests Planned	11	11	11	11
Tests Executed	11	11	11	11
Tests Passed	2	4	8	11
Defects Found	9	7	3	0
Defects Fixed	0	2	4	3