# HW 01: Testing triangle classification

**Karthik Darapaneni**

I found that the requirements specification effectively outlined the necessary classifications for triangles, including Equilateral, Isosceles, Scalene, and Right triangles. However, it would have been beneficial to explicitly address the handling of invalid inputs, such as negative values and non-triangle combinations, to ensure clarity on expected behavior from the outset. Additionally, providing guidance on the precision required for comparing right triangles would have been valuable.

One of the challenges encountered was ensuring the accuracy of computed values when using floating-point arithmetic to determine right triangles. I addressed this issue by implementing rounding, but it required careful attention.

When testing in a Jupyter notebook, adjustments to the unittest command (unittest.main(argv=[''], exit=False)) were necessary to prevent premature exiting of the notebook. This adaptation was specific to Jupyter and required a certain level of knowledge to execute.

To confirm the sufficiency of the test cases, I applied the following criteria:

- Ensured coverage of all triangle types, including Equilateral, Isosceles (both right and non-right), and Scalene (both right and non-right) triangles.
- Included test cases for invalid triangles, covering scenarios where side lengths did not satisfy the triangle inequality and where negative or zero-length sides were present.
- Tested both integer and floating-point side lengths to account for real-world usage where input values may not always be whole numbers.

I manually reviewed each test case to validate that the output aligned with the expected classifications, ensuring that the function handled diverse inputs correctly. By addressing potential edge cases and ensuring comprehensive coverage, I determined that the test cases were sufficient.