**ASHABA JOSHUA JASPER A94164**

**MUKISA HASSAN BAHATI A97028**

**KAMUSIIME DAPHINE A94405**

**Red Phase:**

Primary focus was on writing unit tests for the Calculator class without implementing the class itself. The goal was to define test cases that cover various functionalities such as addition, subtraction, multiplication, division, square roots, exponentials, and absolute values. Each test case was intentionally designed to fail as there was no implementation for the Calculator class at this stage. This phase aimed to clearly define the expected behavior of the Calculator class based on the requirements.

**Green Phase:**

Emphasis is on implementing the Calculator class to satisfy the requirements specified in the test cases written during the red phase. Each method (add, subtract, multiply, divide, find square root, get exponentials and absolute values) was implemented with the aim of making the corresponding test cases pass. This phase involved writing the minimum amount of code necessary to pass the failing tests. Basic arithmetic operations were implemented straightforwardly without considering code optimization or refactoring.

**Refactor Phase:**

During the Refactor phase, the focus was on reviewing and improving the code while ensuring that all test cases continued to pass. The code was refactored to enhance readability, maintainability, and adherence to best practices. This involved renaming variables, adding more descriptive docstrings to the methods, ensuring consistent formatting, and addressing any code smells or duplication.

Refactoring decisions were based on improving code clarity and organization without altering the behavior of the Calculator class.

**Challenges Faced:**

One challenge encountered during the implementation was ensuring that the test cases covered a wide range of scenarios for each arithmetic operation. It required careful consideration to include tests for edge cases such as zero, negative numbers, and large operands.

Ensuring that the implementation satisfied the requirements specified by the test cases while keeping the code simple and clear was another challenge.

**Rationale Behind Refactoring Decisions:**

Refactoring decisions were driven by the goal of improving code quality and maintainability while preserving the functionality of the Calculator class.

Descriptive method names and docstrings were added to improve code documentation and readability. Code formatting was standardized to adhere to PEP 8 guidelines.

These changes were made to enhance code clarity and make it easier for developers to understand and maintain the codebase in the future.