TITLE: PYTHON CALCULATOR WITH UNIT TESTING

Team Members:

1. **Name:** Bhanu Priya R
   CAN_ID: CAN_35273311
2. **Name:** Muppalla Mohitha
   CAN_ID: CAN_35272392
3. **Name:** M Chandra Shekara
   CAN_ID: CAN_35269198
4. **Name:** Nikhila S
   CAN_ID: CAN_35266392
5. **Name:** Narendra Babu G
   CAN_ID: CAN_35277509

PHASE 2: DESIGN

2.1 SYSTEM ARCHITECTURE

The Python calculator project is structured using a modular architecture that separates core logic, execution, and testing. The core logic resides in the Calculator class inside the Calculator.py file. The execution is handled in run_calculator.py, which demonstrates usage of the class, and testing is done using test_calculator.py with Python's unittest module. This separation improves code readability, maintainability, and scalability.

2.2 CLASS DESIGN

The design revolves around a single class named Calculator, which encapsulates the arithmetic operations. The class is designed using Object-Oriented Programming principles, offering methods for addition, subtraction, multiplication, and division. The design ensures the code is modular and allows future expansion, such as adding more mathematical operations.

2.3 METHOD DESIGN

Each method in the class (add, subtract, multiply, divide) is implemented with clear functionality:

- Accepts two parameters (number1 and number2) which must be integers or floats.
- Uses type hints (Union [int, float]) to ensure input correctness.
- Returns the result of the operation.
- Includes basic documentation using docstrings to describe the method's purpose, input, output, and possible exceptions.

2.4 INPUT AND OUTPUT HANDLING

Inputs to the calculator methods are passed directly as parameters from the main script. There is no interactive user input in the current version, making it suitable for programmatic usage and testing. The output is returned by each method and displayed using print() statements in the main script. This simple I/O model ensures clarity and makes integration with GUIs or APIs easier in future development.

2.5 ERROR AND EXCEPTION HANDLING

The calculator is designed to handle common errors gracefully. Each method includes checks and exception handling blocks:

- ZeroDivisionError is raised and handled specifically in the divide method when dividing by zero.
- TypeError is raised if the inputs are not integers or floats. This approach ensures the application doesn't crash and gives clear error messages to developers or users.

2.6 UNIT TESTING DESIGN

A dedicated testing file (test_calculator.py) uses Python's unittest framework to validate the correctness of each method:

- Valid test cases check the expected results for addition, subtraction, multiplication, and division.
- Invalid test cases verify that exceptions like ZeroDivisionError and TypeError are raised as expected.This systematic testing process ensures that each function behaves correctly under all expected conditions and supports future modifications confidently.

2.7 SUMMARY

The design of the Python Calculator project follows a clear, modular architecture that separates core logic, execution, and testing into distinct files, promoting maintainability and scalability. The core functionality is encapsulated within a single Calculator class designed with object-oriented principles, providing basic arithmetic operations with robust input validation and error handling. Each method is well-documented and enforces type safety through type hints. Input and output handling is streamlined for programmatic use, simplifying future integration with other interfaces. Comprehensive unit testing is integrated using Python's unittest framework, covering both valid computations and exception scenarios. Overall, the design ensures clarity, reliability, and ease of future enhancements.