Objective:

The objective of this exercise is to implement and test a Python class representing a simple banking account using the Test-Driven Development (TDD) approach. This involves writing test cases to define the expected behavior of the class and then implementing the class to satisfy these test cases.

Steps:

1. *Write Test:*
    - Begin by writing test cases that specify the expected behavior of the banking account class.
    - Define test cases for depositing funds, withdrawing funds, and checking the account balance.
    - Ensure that the test cases cover various scenarios, such as valid and invalid inputs, account balance updates, and error handling.
2. *Run Test:*
    - Execute the test cases using a testing framework like pytest.
    - Expect the test cases to fail initially since the banking account class implementation does not exist yet.
3. *Write Code:*
    - Implement the banking account class with methods corresponding to the test cases' requirements.
    - Define methods for depositing funds, withdrawing funds, and checking the account balance.
    - Include error handling logic to handle scenarios such as insufficient funds for withdrawals.
4. *Run Test Again:*
    - Re-run the test cases to validate the correctness of the class implementation.
    - Ensure that the implemented methods behave as expected and satisfy the requirements specified in the test cases.
    - Expect all test cases to pass if the class functions correctly.
5. *Refactor (Optional):*
    - Refactor the class implementation as needed to improve readability, performance, or maintainability.
    - Ensure that all test cases continue to pass after refactoring, indicating that the class's behavior remains consistent.

By following these steps, you can effectively develop and test the banking account class using the TDD approach. This iterative process emphasizes writing tests before writing code, leading to a more robust and reliable class implementation.