Lab 2: Test-Driven Development (TDD) vs Behavior-Driven Development (BDD)

Objective

The objective of this lab is to understand and compare **Test-Driven Development (TDD)** and **Behavior-Driven Development (BDD)**. Both practices promote writing tests before code, but they differ in focus, style, and audience. This lab aims to illustrate their differences through theory and practical examples.

Materials Used

- A computer with Python installed (version 3.x)
- Code editor (e.g. Visual Studio Code)
- **pytest** (for TDD)
- **behave** (for BDD)

Theory

Test-Driven Development (TDD)

TDD is a development approach where tests are written before the actual code. The goal is to create small, incremental unit tests that guide the code design.

Key characteristics:

- Focus on code correctness at unit level.
- Tests written in programming language (e.g., Python `assert`).
- Follows the Red → Green → Refactor cycle.

Example unit test:

```
def test_add():
    assert add(2, 3) == 5
```

Behavior-Driven Development (BDD)

BDD extends TDD by focusing on system behavior from the user's perspective. Tests are written in a human-readable format, often using `Given-When-Then` syntax.

Key characteristics:

- Focus on behavior expected by the user.
- Scenarios written in plain language (Gherkin syntax).
- Enhances collaboration between developers, testers, and business stakeholders.

Example scenario:

```
Scenario: Adding two numbers
  Given I have entered 2 and 3
  When I press add
  Then the result should be 5
```

Implementation

TDD Example

1. Write a failing test:

```python
def test_add():
    assert add(2, 3) == 5
```

2. Write code to pass the test:

```python
def add(a, b):
    return a + b
```

3. Run test:

```
pytest test_add.py
```

BDD Example

1. Write a feature file (`add.feature`):

```
Feature: Addition

  Scenario: Add two numbers
```

```
    Given I have entered 2 and 3
    When I press add
    Then the result should be 5
```

2. Implement step definitions in Python:

```python
from behave import given, when, then

@given('I have entered {a:d} and {b:d}')
def step_given_numbers(context, a, b):
    context.a = a
    context.b = b

@when('I press add')
def step_when_add(context):
    context.result = context.a + context.b

@then('the result should be {expected:d}')
def step_then_result(context, expected):
    assert context.result == expected
```

3. Run test:

```
behave add.feature
```

Conclusion

In this lab, I explored TDD and BDD. Both help improve code by writing tests first. TDD checks if code works at a small level, while BDD focuses on user-friendly behavior.

I practiced writing unit tests with TDD and feature tests with BDD. Learning both methods will help me build better, more reliable software.