SET-221
Software Testing Technologies

LAB # 12

LAB Title

Introducing Pytest and Fixtures

Assessment of CLO: 03, PLO: 05

Student Name:			
Roll No.			
Semester		Session	

S. No.	Perf. Level / Criteria	Excellent (2.5)	Good (2)	Satisfactory (1.5)	Needs Improvement (0 ~ 1)	Marks Obtained
1	Project Execution & Implementation	Fully functional, optimized, and well-structured.	Minor errors, mostly functional.	Some errors, requires guidance.	Major errors, non-functional, or not Performed.	
2	Results & Debugging Or Troubleshooting	Accurate results with effective debugging Or Troubleshooting.	Mostly correct, some debugging Or Troubleshooting needed.	Partial results, minimal debugging Or Troubleshooting.	Incorrect results, no debugging Or Troubleshooting, or not attempted.	
3	Problem-Solving & Adaptability (VIVA)	Creative approach, efficiently solves challenges.	Adapts well, minor struggles.	Some adaptability, needs guidance.	Lacks innovation or no innovation, unable to solve problems.	
4	Report Quality & Documentation	Clear, structured, with detailed visuals.	Mostly clear, minor gaps.	Some clarity issues, missing details.	Poorly structured, lacks clarity, or not submitted.	
Total Marks Obtained Out of 10						

Experiment evaluated by

Instructor's Name	Engr.Bushra Aziz		
Date		Signature	

Lab Experiment 12: Introducing Pytest and Fixtures

Objective

To write a simple Python test and execute it using the Pytest framework within PyCharm. To understand what Pytest fixtures are and how to use them for setting up preconditions and tearing down resources for your tests within PyCharm.

Section 1: Running Your First Pytest Program in PyCharm

Steps

Step 1: Create a New PyCharm Project Open PyCharm and create a new project:

1. Go to File > New Project....
2. In the "New Project" dialog:
    - Set the "Location" to a suitable path (e.g., ~/PycharmProjects/pytest_lab).
    - Choose "New environment using Virtualenv" (recommended for isolated project dependencies).
    - Ensure the "Base interpreter" points to your Python 3.6+ installation.
    - Click Create.

Step 2: Install Pytest in PyCharm PyCharm provides an integrated terminal and project interpreter settings for managing packages.

Option A: Using PyCharm's Terminal (Recommended)

1. Open the Terminal tab at the bottom of the PyCharm window.
2. Ensure your virtual environment is activated (you should see (venv) or similar prefix in the terminal prompt).
3. Run the installation command:
4. pip install pytest
5.

    You should see output indicating that pytest and its dependencies have been successfully installed.

Option B: Using Project Interpreter Settings

1. Go to File > Settings (or PyCharm > Preferences on macOS).
2. Navigate to Project: <your_project_name> > Python Interpreter.
3. Click the + button at the bottom of the "Packages" list.
4. Search for pytest, select it, and click Install Package.

Step 3: Write Your First Test File Create a new Python file for your tests within your PyCharm project:

1. In the Project tool window (usually on the left), right-click on your project root (e.g., pytest_lab).
2. Select New > Python File.
3. Name the file test_first_program.py. Pytest automatically discovers files that start with test_ or end with _test.py.

Add the following code to test_first_program.py:

```
# test_first_program.py

def test_addition():
    """
    This test checks if 2 + 2 equals 4.
    """
    assert 2 + 2 == 4

def test_string_concatenation():
    """
    This test checks if two strings concatenate correctly.
    """
    assert "hello" + " world" == "hello world"

def test_list_length():
    """
    This test checks the length of a list.
    """
    my_list = [1, 2, 3, 4, 5]
    assert len(my_list) == 5

# A deliberately failing test to see failure output
def test_failing_example():
    """
    This test is designed to fail to demonstrate Pytest's failure reporting.
    """
    assert 10 > 20
```

Step 4: Run Pytest in PyCharm PyCharm has excellent integration with Pytest.

Option A: Run from the Editor (Recommended for individual tests/files)

1. Open test_first_program.py in the editor.
2. You'll notice green "play" (run) icons next to your test functions and at the top of the file.
3. Click the green play icon next to the test_addition function, or the one at the top of the file to run all tests in test_first_program.py.
4. Select Run 'pytest in test_first_program.py' (or Run 'test_addition').

Option B: Run from PyCharm's Terminal

1. Open the Terminal tab at the bottom of the PyCharm window.
2. Ensure you are in your project's root directory (or the pytest_lab directory if you cd'd into it).
3. Run Pytest:
4. pytest

Step 5: Observe the output.

Section 2: Using Fixtures in Pytest with PyCharm

What are Fixtures?

Lab Experiment 12: Introducing Pytest and Fixtures

In testing, "setup" refers to the actions needed to get the system into a known state before a test runs (e.g., opening a browser, connecting to a database, creating temporary files). "Teardown" refers to the actions needed to clean up after a test (e.g., closing the browser, disconnecting, deleting temporary files).

Pytest fixtures are functions that provide a flexible and explicit way to manage these setup and teardown operations. They promote:

- **Reusability:** A single fixture can be used by multiple tests.
- **Modularity:** Setup logic is separated from test logic.
- **Readability:** Tests become cleaner as they only focus on the specific behavior being tested.

Steps

Step 1: Create a conftest.py File Fixtures are often defined in a special file named conftest.py. Pytest automatically discovers fixtures defined in conftest.py files in the test directory or its parent directories.

In PyCharm:

1. In the Project tool window, right-click on your project root (pytest_lab).
2. Select New > Python File.
3. Name the file conftest.py.

Add the following code to conftest.py:

```
# conftest.py
import pytest

@pytest.fixture
def setup_data():
    """
    A simple fixture that provides some data for tests.
    This function will run before any test that requests 'setup_data'.
    """
    print("\n--- Setting up data for test ---")
    data = {"name": "Alice", "age": 30}
    yield data  # 'yield' passes the data to the test and marks the setup phase
    print("--- Tearing down data after test ---")
    # Code after 'yield' runs as teardown after the test completes
```

Explanation of conftest.py:

- import pytest: Necessary to use Pytest functionalities.
- @pytest.fixture: This decorator marks the setup_data function as a Pytest fixture.
- yield data: This is the key part. The value after yield (data in this case) is what the test function will receive. Any code *after* the yield statement will be executed as teardown logic *after* the test function has completed.

Step 2: Modify Your Test File to Use the Fixture Now, let's modify test_first_program.py to use the setup_data fixture.

Lab Experiment 12: Introducing Pytest and Fixtures
Open test_first_program.py and update its content:

```
# test_first_program.py

# (You can keep your existing tests or replace them with these)

def test_user_name(setup_data):
    """
    This test uses the 'setup_data' fixture to get user information.
    """
    print(f"\nRunning test_user_name with data: {setup_data}")
    assert setup_data["name"] == "Alice"

def test_user_age(setup_data):
    """
    This test also uses the 'setup_data' fixture.
    """
    print(f"\nRunning test_user_age with data: {setup_data}")
    assert setup_data["age"] > 25

# You can still have tests that don't use fixtures
def test_simple_math():
    assert 1 + 1 == 2

# If you kept the failing test from Section 1, it might look like this:
# def test_failing_example():
#     assert 10 > 20
```

Explanation of Test File:

- Notice that setup_data is passed as an argument to the test functions (test_user_name and test_user_age). Pytest automatically detects that setup_data is a fixture and injects its return value (the data dictionary) into the test function.

Step 3: Run Pytest Again with Print Statements Displayed To see the print statements from your fixtures and tests, you need to run Pytest with the -s flag.

Option A: Using PyCharm's Run/Debug Configurations (Recommended) This is the best way to consistently apply command-line arguments in PyCharm.

1. Go to Run > Edit Configurations....
2. In the left pane, find your existing Pytest configuration (e.g., pytest in test_first_program.py). If you don't have one, click + and choose Python tests > Pytest.
3. In the "Additional Arguments" field, type -s.
4. Click Apply and then OK.
5. Now, run your tests using the green play button in the toolbar, or by right-clicking the file and selecting Run.

Option B: Using PyCharm's Terminal

1. Open the Terminal tab at the bottom of the PyCharm window.
2. Run Pytest with the -s flag:

3. pytest -s
4.

Step 4: Observe the Output

Task

1. Create a new Python file named test_math_operations.py. Write at least three new test functions in this file to test basic arithmetic operations (e.g., multiplication, division, modulo).Ensure at least one test is designed to fail .Run only the tests in test_math_operations.py.
2. Modify your conftest.py file to add a new fixture called sample_string. This fixture should yield a string like "Pytest is awesome!".
   Create a new test file named test_string_fixture.py.

   Write two test functions in test_string_fixture.py that use the sample_string fixture:

   - One test should check if the string contains a specific substring (e.g., "awesome").
   - Another test should check the length of the string.

   Run these new tests and observe the fixture's setup and teardown messages (remember the -s flag).

3. In test_math_operations.py (from Task 1), ensure you have a failing test.Use PyCharm's debugging features:
   - Set a breakpoint on the assert line of the failing test.
   - Right-click the green play icon next to the failing test and select Debug 'test_...'.
   - Step through the code and inspect variables to understand why the assertion is failing.
   - Fix the test so it passes.