In Python's pytest framework, @pytest.fixture is a decorator used to define reusable components for setting up and tearing down test dependencies. Fixtures help streamline and simplify testing by allowing you to define common setup steps that can be shared across multiple test functions.

**Key Features of @pytest.fixture:**

1. **Reusable Test Setup**: A fixture provides a common setup that multiple test functions can use. For example, initializing a database connection, creating a temporary file, or preparing mock data.
2. **Modular Design**: Fixtures promote modular and maintainable tests by separating setup logic from the test logic.
3. **Automatic Dependency Injection**: Pytest automatically detects and injects the appropriate fixture into your test function based on the function signature.
4. **Scope**: You can specify the fixture's scope to control its lifecycle. Common scopes are:
   * function: The default. The fixture is invoked once per test function.
   * class: The fixture is invoked once per class of tests.
   * module: The fixture is invoked once per module of tests.
   * session: The fixture is invoked once per test session.

**Example: Using @pytest.fixture**

**Basic Usage**

import pytest

# Define a fixture

@pytest.fixture

def sample\_data():

return {"key": "value"}

# Use the fixture in a test

def test\_example(sample\_data):

assert sample\_data["key"] == "value"

**Fixture with Setup and Teardown**

You can use yield to include teardown logic.

@pytest.fixture

def resource():

# Setup

res = open("example.txt", "w")

yield res

# Teardown

res.close()

def test\_resource(resource):

resource.write("Hello, pytest!")

@pytest.fixture(scope="module") def db\_connection(): connection = connect\_to\_database() yield connection connection.close()

**Benefits of Fixtures**

* **Eliminate Redundant Code**: Avoid repeating setup logic across multiple tests.
* **Enhanced Readability**: Tests focus on the behavior being tested rather than setup details.
* **Better Resource Management**: Fixtures handle resource setup and cleanup efficiently.

**Advanced Features**

1. **Parameterized Fixtures**: Pass parameters to a fixture to test different configurations.

@pytest.fixture(params=[1, 2, 3])

def number(request):

return request.param

def test\_numbers(number):

assert number in [1, 2, 3]

@pytest.fixture

def db\_connection():

return connect\_to\_database()

@pytest.fixture

def db\_cursor(db\_connection):

return db\_connection.cursor()

to untrack a repo: rm -rf .git