**Key Terms**

* **Testing** - Validating code behavior through automated scripts to check for correctness and catch issues.
* **Unit Test** - Testing isolated chunks of code like functions or classes.
* **Test Function** - An isolated test wrapped in a Python function.
* **Test Class** - A class that contains related test methods.
* **Assertion** - Boolean checks in test code to verify values match expectations.
* **Fixture** - Shared test data or state managed by the testing framework.

def test\_multiply(self):

    calculator = Calculator()

    assert calculator.multiply(3, 5) == 15

# Test function using pytest fixture for temp data

import pytest

@pytest.fixture

def input\_value():

    return 10

def test\_stuff(input\_value):

    assert input\_value == 10

**Key Terms**

* **Plain Asserts** - Basic assertion statements in Python used to verify values and results.
* **Test Classes** - Classes that contain multiple related test methods and setup/teardown logic.
* **Parametrize** - Pytest decorator to run a test multiple times with different arguments.
* **Setup Method** - Code that runs before each test method in a Test Class.
* **Teardown Method** - Code that runs after each test method in a Test Class.

**Python Code Examples**

assert num \* num == square(num)

# Parametrized test function

@pytest.mark.parametrize("num", [1, 5, 10])

def test\_squared(num):

    def test\_divide\_two\_numbers(self):

        assert self.calculator.divide(10, 5) == 2

In this lesson, we explored more advanced techniques for writing effective tests in Python using pytest. We looked at leveraging plain Python asserts for simple yet descriptive test failures. We covered how to group tests together in classes, make use of setup/teardown methods, and parameterize tests.

**Top 3 Key Points**

* Plain asserts give easy yet descriptive test failures
* Test classes group tests and execute setup/teardown
* Parametrized tests run multiple times with different data

**Key Terms**

* **Test Failure Output** - Pytest results containing details on which tests failed and why.
* **PDB (Python Debugger)** - Tool to debug Python code by stepping through execution.
* **Pytest Fixtures** - Shared test data/state managed by Pytest.
* **Pytest Plugins** - Extensions that provide added Pytest functionality.
* **Pytest Options** - Command line flags that control Pytest test runner behavior.

# Using PDB to debug inside a test

import pdb

def test\_stuff():

    x = 5 \* 5

    pdb.set\_trace() # Launch debugger

    assert x == 10

# Pytest fixture providing temp directory

import pytest

@pytest.fixture

def temp\_dir():

    import tempfile

    dirpath = tempfile.mkdtemp()

    yield dirpath

    import shutil

    shutil.rmtree(dirpath)

# Pytest test function using fixture

def test\_using\_dir(temp\_dir):

    path = temp\_dir.join("test.txt")

    # Use temp\_dir path for test I/O

**For debugging** :- “pytest –pdb test\_abc.py”

Pytest –help

Pytest –collect-only

Pytest -x

Pip install pytest-xdist

Pytest -n 4