# PYTEST IN PYTHON

**What is Pytest?** **Pytest** is a powerful and easy-to-use testing framework for Python. It allows you to write simple unit tests as well as complex functional testing. Pytest is widely used because of its simplicity, scalability, and rich set of plugins.

**Installation:**

To install pytest, simply use pip: pip install pytest

To verify the installation: pytest --version

**Directory Structure (Typical)**

```
project/
│
├── app.py
├── test_app.py
└── requirements.txt
```

Test files usually start with test_ and are stored in the same directory or in a /tests folder.

## Writing Your First Test:

**Code to Test:** **calculator.py**

```
# calculator.py

def add(x, y):
    return x + y

def subtract(x, y):
    return x - y

def multiply(x, y):
    return x * y

def divide(x, y):
    if y == 0:
        raise ValueError("Cannot divide by zero.")
    return x / y
```

**Test Code: test_calculator.py**

```
# test_calculator.py

import calculator

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

def test_subtract():
    assert calculator.subtract(5, 2) == 3

def test_multiply():
    assert calculator.multiply(4, 3) == 12

def test_divide():
    assert calculator.divide(10, 2) == 5

def test_divide_by_zero():
    import pytest
    with pytest.raises(ValueError):
        calculator.divide(5, 0)
```

**Running Tests:**

To run all tests in the current directory: pytest

To run a specific file: pytest test_calculator.py

To see detailed output: pytest -v

## Pytest Features Explained:

### Assertions:

No need to use self.assertEqual() like in unittest. You can directly use Python's built-in assert statement.

```
assert func(5) == 25
```

## Fixtures:

Fixtures are used to set up conditions before a test runs.

```
import pytest

@pytest.fixture
def sample_data():
    return {"username": "john", "password": "1234"}

def test_user(sample_data):
    assert sample_data["username"] == "john"
```

## Parametrize:

Run the same test function with different data sets.

```
import pytest

@pytest.mark.parametrize("a,b,result", [(2, 3, 5), (10, 5, 15)])
def test_add(a, b, result):
    assert a + b == result
```

## Markers:

Use markers to categorize or skip tests.

```
import pytest

@pytest.mark.slow
def test_slow_function():
    import time
    time.sleep(5)
    assert True

@pytest.mark.skip(reason="Feature not implemented")
def test_future_feature():
    assert False
```

## Pytest CLI Options:

| Command | Description |
|---|---|
| **-v** | Verbose output |
| **-k "keyword"** | Run tests matching keyword |
| **-m "marker"** | Run tests with a specific marker |
| **--maxfail=2** | Stop after 2 failures |
| **--tb=short** | Short traceback |

## Advanced: **Test Coverage:**

You can use pytest-cov for checking test coverage: pip install pytest-cov

pytest --cov=calculator test_calculator.py

## Useful Plugins:

- pytest-django: For Django testing
- pytest-mock: For mocking support
- pytest-xdist: Run tests in parallel
- pytest-html: Generate HTML reports

## Best Practices:

- Name test files as test_*.py
- Keep test functions simple
- Use fixtures for reusable setup
- Use parameterization for multiple test cases
- Keep test coverage above 80

## Final Notes:

- Pytest is beginner-friendly but powerful enough for complex test suites.
- It integrates well with CI/CD tools like GitHub Actions, Jenkins, etc.
- It encourages **TDD** (Test-Driven Development) and clean code practices.

**Presented By:**

Narmeen Asghar