Fixtures: Functions That Run Before (Some Of Them After) The Test Functions.

The Most Value Of Them: Using For DB Initialization, WebDriver Initialization, And …etc.

****

Note: If We Set The Fixture Inside Any File, It Then Will Run For That File Only.

****

Note: If We Set Fixtures Inside The conftest.py-File It Will Be Then Available To All Directories And Sub-Directory.

****

```python
import pytest;

@pytest.fixture()
def setup_names():
    print("=" * 15);
    print("\n ... Setup From Fixtures ...\n");
    print("=" * 15);

    return ["Jafar", "Loka", "Jafar-Loka-01"];

def test_len_of_names(setup_names):
    assert len(setup_names) == 3;
```

****

If We Use Fixtures With Markers, Then The Returned Value From Fixtures Can't Be Used.

We Can Use Fixtures Using Markers, By This Way:

```
@pytest.mark.usefixtures("setup_names")
def test_use_fixtures():
    assert 1 == 1;
```

```
14  @pytest.mark.usefixtures("setup_names")
15  def test_use_fixtures():
16      assert 1 == 1;
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS                                   cmd

========================================================= test session starts =========================================================

(.venv) C:\Tests\Python-PyTest-From-Tutorial-01>pytest -v -s -k "fixtures and use"
========================================================= test session starts =========================================================
platform win32 -- Python 3.12.6, pytest-8.3.3, pluggy-1.5.0 -- C:\Tests\Python-PyTest-From-Tutorial-01\.venv\Scripts\python.exe
cachedir: .pytest_cache
metadata: {'Python': '3.12.6', 'Platform': 'Windows-10-10.0.19043-SP0', 'Packages': {'pytest': '8.3.3', 'pluggy': '1.5.0'}, 'Plugi
ns': {'html': '4.1.1', 'metadata': '3.1.1'}}
rootdir: C:\Tests\Python-PyTest-From-Tutorial-01
configfile: pytest.ini
plugins: html-4.1.1, metadata-3.1.1
collected 24 items / 23 deselected / 1 selected

test_fixtures.py::test_use_fixtures ===============

 ... Setup From Fixtures ...

===============
PASSED

============================================== 1 passed, 23 deselected in 0.07s ===============================================
```

****

If We Want To Close The:

- Connection To DB.
- The Driver Page.
- The File.
- Any Thing That We Returned From Fixture

Then We Must Use Yield.

```
days1 = ["Sat", "Sun", "Mon"];
days2 = ["Tue", "Wed", "Thu"];

@pytest.fixture()
def setup_days1():
    wk1 = days1.copy();
    wk1.append('Fri');
    yield wk1;
    print("\n After Yield Wk1 \n");

    # wk1.clear();
    wk1.pop();

def test_wk1(setup_days1):
    assert len(setup_days1) == 4;
```

```
(.venv) C:\Tests\Python-PyTest-From-Tutorial-01>pytest -v -s -k fixtures_02
================================================= test session starts =================================================
platform win32 -- Python 3.12.6, pytest-8.3.3, pluggy-1.5.0 -- C:\Tests\Python-PyTest-From-Tutorial-01\.venv\Scripts\python.exe
cachedir: .pytest_cache
metadata: {'Python': '3.12.6', 'Platform': 'Windows-10-10.0.19043-SP0', 'Packages': {'pytest': '8.3.3', 'pluggy': '1.5.0'}, 'Plugi
ns': {'html': '4.1.1', 'metadata': '3.1.1'}}
rootdir: C:\Tests\Python-PyTest-From-Tutorial-01
configfile: pytest.ini
plugins: html-4.1.1, metadata-3.1.1
collected 25 items / 24 deselected / 1 selected

test_fixtures_02.py::test_wk1 PASSED
 After Yield Wk1



============================================= 1 passed, 24 deselected in 0.08s ==============================================
```

****

We Can Also, Use Multiple Fixtures For The Same Test:

```
def test_wk1_wk2(setup_days1, setup_days2):
    assert len(setup_days1) + len(setup_days2) == 7;
```

****

We Can Use This Way To Handle DB Connection Open, File Open, …etc.

```
@pytest.fixture()
def setup_days1():
    wk1 = days1.copy();
    wk1.append('Fri');
    yield wk1;
    wk1.pop();

@pytest.fixture()
def setup_days2():
    wk2 = days2.copy();
    yield wk2;
    wk2.clear();
```

```
(.venv) C:\Tests\Python-PyTest-From-Tutorial-01>pytest -v -s -k fixtures_02
================================================== test session starts ==================================================
platform win32 -- Python 3.12.6, pytest-8.3.3, pluggy-1.5.0 -- C:\Tests\Python-PyTest-From-Tutorial-01\.venv\Scripts\python.exe
cachedir: .pytest_cache
metadata: {'Python': '3.12.6', 'Platform': 'Windows-10-10.0.19043-SP0', 'Packages': {'pytest': '8.3.3', 'pluggy': '1.5.0'}, 'Plugi
ns': {'html': '4.1.1', 'metadata': '3.1.1'}}
rootdir: C:\Tests\Python-PyTest-From-Tutorial-01
configfile: pytest.ini
plugins: html-4.1.1, metadata-3.1.1
collected 26 items / 24 deselected / 2 selected

test_fixtures_02.py::test_wk1 PASSED
 After Yield Wk1


test_fixtures_02.py::test_wk1_wk2 PASSED
 After Yield Wk1



============================================ 2 passed, 24 deselected in 0.10s ============================================
```

****

The conftest.py File Used To Share The Fixtures Across The Test Files.

We Can Set Multiple conftest.py Files In All Subdirectories.

It Shouldn't Import The conftest.py-Files From The Test Files.

The conftest.py File Is Python Module.

****

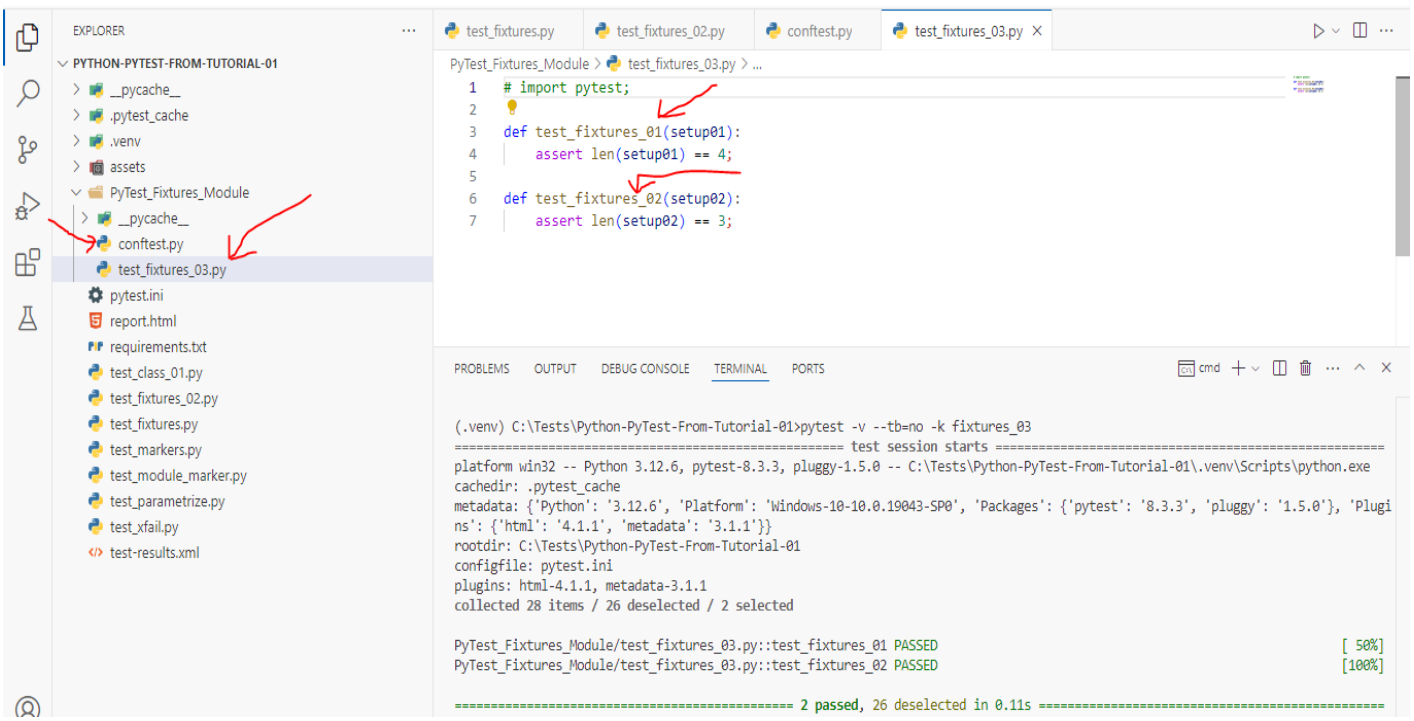
Note: If We Used Files Using Fixtures And Generators (Yield), We Must Close Them Before Remove Them From Directories.

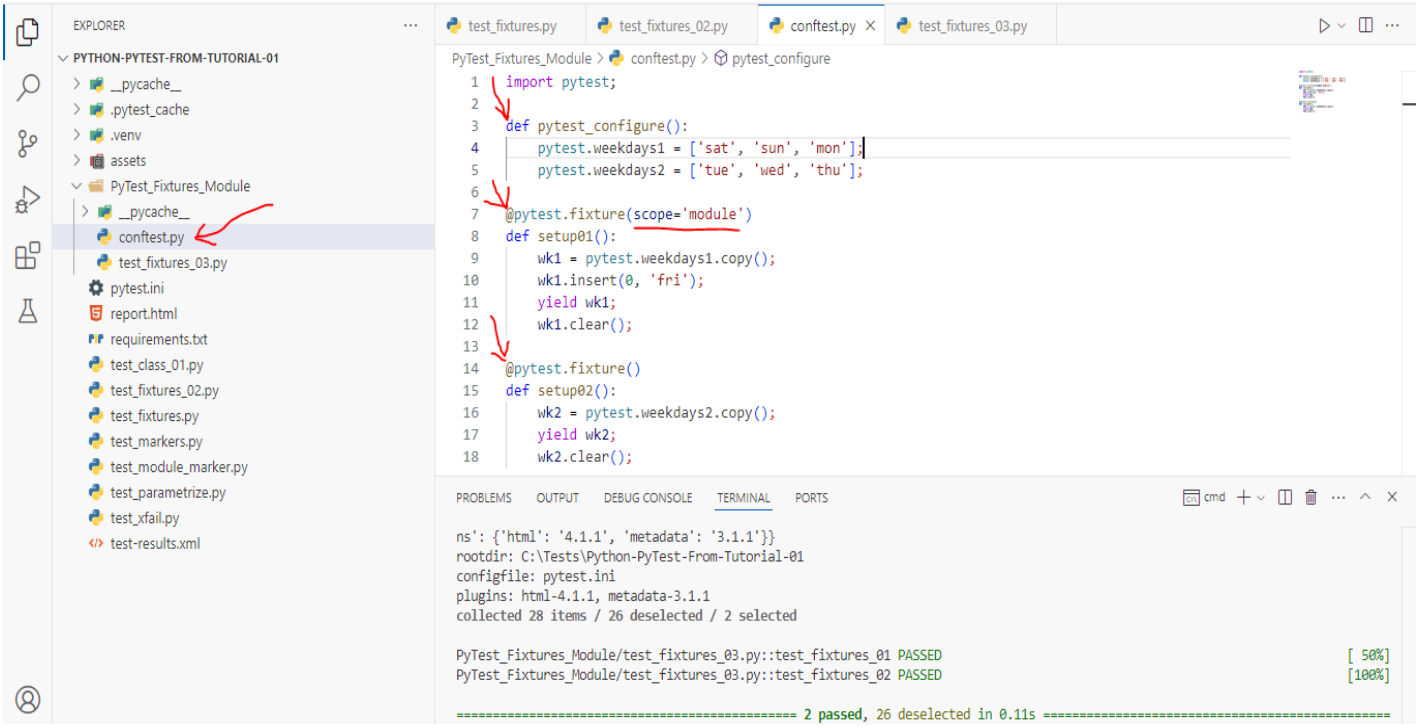
****

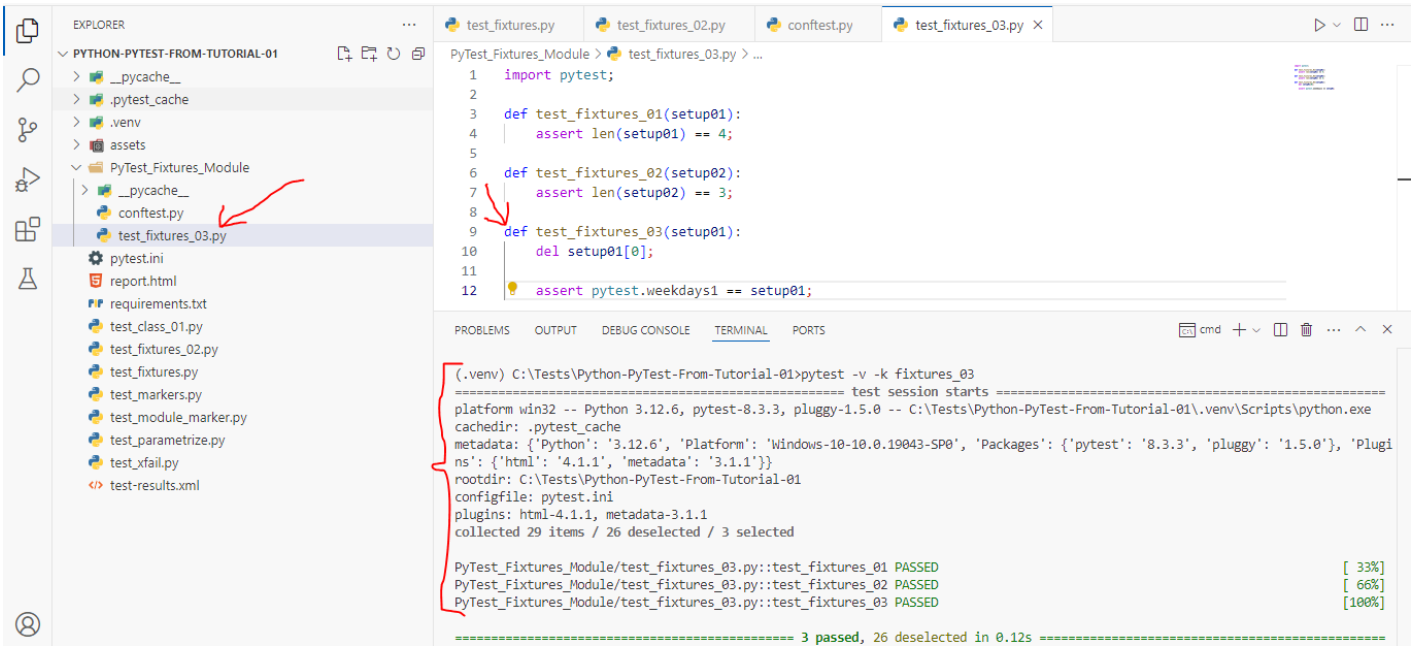
To Create Global Variables For Fixtures, To Use Them We Can Use: *pytest_configure.py*:

```
def pytest_configure():
    pytest.weekdays1 = ['sat', 'sun', 'mon'];
    pytest.weekdays2 = ['tue', 'wed', 'thu'];
```

****







****

Note: Fixtures Can Be Overridden From Test Module Level.

****

If We Want To Show The Fixtures That We Use We Can Use: --setup-show With Pytest:

```
(.venv) C:\Tests\Python-PyTest-From-Tutorial-01>pytest -v -k fixtures_03 --tb=no --setup-show
================================================== test session starts ==================================================
platform win32 -- Python 3.12.6, pytest-8.3.3, pluggy-1.5.0 -- C:\Tests\Python-PyTest-From-Tutorial-01\.venv\Scripts\python.exe
cachedir: .pytest_cache
metadata: {'Python': '3.12.6', 'Platform': 'Windows-10-10.0.19043-SP0', 'Packages': {'pytest': '8.3.3', 'pluggy': '1.5.0'}, 'Plug
ns': {'html': '4.1.1', 'metadata': '3.1.1'}}
rootdir: C:\Tests\Python-PyTest-From-Tutorial-01
configfile: pytest.ini
plugins: html-4.1.1, metadata-3.1.1
collected 29 items / 26 deselected / 3 selected

PyTest_Fixtures_Module/test_fixtures_03.py::test_fixtures_01
    SETUP    M setup01
        PyTest_Fixtures_Module/test_fixtures_03.py::test_fixtures_01 (fixtures used: setup01)PASSED
PyTest_Fixtures_Module/test_fixtures_03.py::test_fixtures_02
        SETUP    F setup02
        PyTest_Fixtures_Module/test_fixtures_03.py::test_fixtures_02 (fixtures used: setup02)PASSED
        TEARDOWN F setup02
PyTest_Fixtures_Module/test_fixtures_03.py::test_fixtures_03
        PyTest_Fixtures_Module/test_fixtures_03.py::test_fixtures_03 (fixtures used: setup01)PASSED
    TEARDOWN M setup01

============================================== 3 passed, 26 deselected in 0.10s ==============================================
```

****

We Can Use Fixtures To Intercept The Test Function Calling:

Note: Here The Months Are Inside The Test Function That Use This Fixture.

```python
@pytest.fixture()
def setup04(request):
    months = getattr(request.module, "months");
    print("\n The Setup04 Fixture");
    print("\n Fixture Scope: ", str(request.scope));
    print("\n Calling Function: ", request.function.__name__);
    print("\n Calling Module: ", request.module.__name__);
    print("\n The Months Of Function Are: ", months);
    months.append("April");
    yield months;
    months.pop();
```

```
(.venv) C:\Tests\Python-PyTest-From-Tutorial-01>pytest -v -k fixtures_04 -s --setup-show
========================================================= test session starts =========================================================
platform win32 -- Python 3.12.6, pytest-8.3.3, pluggy-1.5.0 -- C:\Tests\Python-PyTest-From-Tutorial-01\.venv\Scripts\python.exe
cachedir: .pytest_cache
metadata: {'Python': '3.12.6', 'Platform': 'Windows-10-10.0.19043-SP0', 'Packages': {'pytest': '8.3.3', 'pluggy': '1.5.0'}, 'Plugi
ns': {'html': '4.1.1', 'metadata': '3.1.1'}}
rootdir: C:\Tests\Python-PyTest-From-Tutorial-01
configfile: pytest.ini
plugins: html-4.1.1, metadata-3.1.1
collected 30 items / 29 deselected / 1 selected

PyTest_Fixtures_Module/test_fixtures_04.py::test_request_fixture_01
 The Setup04 Fixture

 Fixture Scope:  function

 Calling Function:  test_request_fixture_01

 Calling Module:  PyTest_Fixtures_Module.test_fixtures_04

 The Months Of Function Are:  ['Jan', 'Feb', 'Mar']

        SETUP    F setup04
        PyTest_Fixtures_Module/test_fixtures_04.py::test_request_fixture_01 (fixtures used: request, setup04)
 The Function That Use The Setup04
PASSED
        TEARDOWN F setup04

============================================= 1 passed, 29 deselected in 0.15s ==============================================
```

```python
months = ["Jan", "Feb", "Mar"];

def test_request_fixture_01(setup04):
    print("\n The Function That Use The Setup04");
    assert len(setup04) == 4;
```

****

Factories as fixtures

The "factory as fixture" pattern can help in situations where the result of a fixture is needed multiple times in a single test. Instead of returning data directly, the fixture instead returns a function which generates the data. This function can then be called multiple times in the test.

****

In This Way We Can Define Fixture As Factory:

```
@pytest.fixture()
def setup05():
    def get_structure(name):
        if name == 'list':
            return [1, 2, 3];
        elif name == 'tuple':
            return (1, 3, 4);
    return get_structure;
```

And In This Way We Can Call Our Fixture That Used Factory:

```
def test_fixtures_list_05(setup05):
    list = setup05('list');
    assert 'list' in str(type(list));

def test_fixtures_tuple_05(setup05):
    tuple = setup05('tuple');
    assert 'tuple' in str(type(tuple))
```

****

Note (From Me): We Can Use Factory As Fixture To Generate The List Of Data That Represent The Test Parametrization Data.

****

Note: This Function Will Be Called 2-Times, For Each Test Cases:

```
import pytest;

@pytest.fixture(params=[(3, 4), [3, 5]])
def fixture01(request):
    return request.param;
```

****

```
(.venv) C:\Tests\Python-PyTest-From-Tutorial-01>pytest -v -k fixtures_06 --tb=no
==================================================== test session starts ====================================================
platform win32 -- Python 3.12.6, pytest-8.3.3, pluggy-1.5.0 -- C:\Tests\Python-PyTest-From-Tutorial-01\.venv\Scripts\python.exe
cachedir: .pytest_cache
metadata: {'Python': '3.12.6', 'Platform': 'Windows-10-10.0.19043-SP0', 'Packages': {'pytest': '8.3.3', 'pluggy': '1.5.0'}, 'Plugi
ns': {'html': '4.1.1', 'metadata': '3.1.1'}}
rootdir: C:\Tests\Python-PyTest-From-Tutorial-01
configfile: pytest.ini
plugins: html-4.1.1, metadata-3.1.1
collected 41 items / 33 deselected / 8 selected

PyTest_Fixtures_Module/test_fixtures_06.py::test_fixtures_with_params_01[fixture010] PASSED                              [ 12%]
PyTest_Fixtures_Module/test_fixtures_06.py::test_fixtures_with_params_01[fixture011] PASSED                              [ 25%]
PyTest_Fixtures_Module/test_fixtures_06.py::test_fixtures_with_params_02[fixture010] PASSED                              [ 37%]
PyTest_Fixtures_Module/test_fixtures_06.py::test_fixtures_with_params_02[fixture011] PASSED                              [ 50%]
PyTest_Fixtures_Module/test_fixtures_06.py::test_fixtures_with_params_03[fixture010] PASSED                              [ 62%]
PyTest_Fixtures_Module/test_fixtures_06.py::test_fixtures_with_params_03[fixture011] PASSED                              [ 75%]
PyTest_Fixtures_Module/test_fixtures_06.py::test_fixtures_with_params_04[fixture010] PASSED                              [ 87%]
PyTest_Fixtures_Module/test_fixtures_06.py::test_fixtures_with_params_04[fixture011] PASSED                              [100%]

============================================= 8 passed, 33 deselected in 0.16s ==============================================
```

****