For More Visual Output, We Can Add ids-Attribute For Output Formatting Of The Fixture:

```
@pytest.fixture(params=[(3, 4), [3, 5]], ids=['tuple', 'list'])
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[tuple] PASSED                [ 12%]
PyTest_Fixtures_Module/test_fixtures_06.py::test_fixtures_with_params_01[list] PASSED                 [ 25%]
PyTest_Fixtures_Module/test_fixtures_06.py::test_fixtures_with_params_02[tuple] PASSED                [ 37%]
PyTest_Fixtures_Module/test_fixtures_06.py::test_fixtures_with_params_02[list] PASSED                 [ 50%]
PyTest_Fixtures_Module/test_fixtures_06.py::test_fixtures_with_params_03[tuple] PASSED                [ 62%]
PyTest_Fixtures_Module/test_fixtures_06.py::test_fixtures_with_params_03[list] PASSED                 [ 75%]
PyTest_Fixtures_Module/test_fixtures_06.py::test_fixtures_with_params_04[tuple] PASSED                [ 87%]
PyTest_Fixtures_Module/test_fixtures_06.py::test_fixtures_with_params_04[list] PASSED                 [100%]

============================================ 8 passed, 33 deselected in 0.17s =============================================
```

****

If We Want To Set Multiple Fixtures With Multiple Params (Like Dictionary):

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

@pytest.fixture(params=['access'])
def fixture02(request):
    return request.param;

def test_fixtures_with_multiple_params_05(fixture01, fixture02):
    if(fixture02 == 'access'):
        assert fixture01[0];
```

****

```
Fixture scopes:

Fixtures are created when first requested by a test, and are destroyed based on their scope:

function: the default scope, the fixture is destroyed at the end of the test.
class: the fixture is destroyed during teardown of the last test in the class.
module: the fixture is destroyed during teardown of the last test in the module.
package: the fixture is destroyed during teardown of the last test in the package.
session: the fixture is destroyed at the end of the test session.
```

****

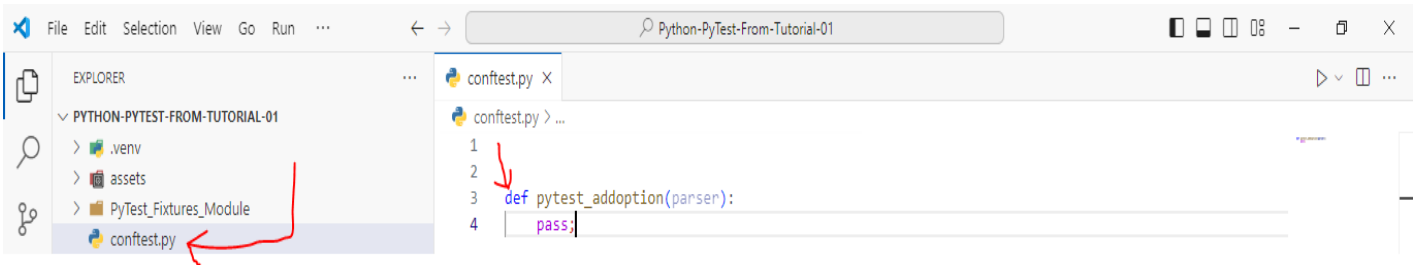
Note (From Me): The Most Benefit From Passing The Parameters To Test Cases By Command Line, Is That We Control The Test Of Each Environment That We Have.

****

Note (To Remember): If We Want To Share Fixtures, We Must Add New File, Called: conftest.py

****

To Add New Option To Our PyTest-Framework For Our Test Cases:



****

```python
def pytest_addoption(parser):
    parser.addoption("--cmdopt", defaults='QA');
```

****

Then To Get The Command Params:

```
@pytest.fixture()
def CmdOpt(pytestconfig):
    opt = pytestconfig.getoption('cmdopt');
    if opt == 'Prod':
        f = open(prod_config, 'r');
    else:
        f = open(QA_config, 'r')
    yield f;
    f.close();
**********************************************************************
```