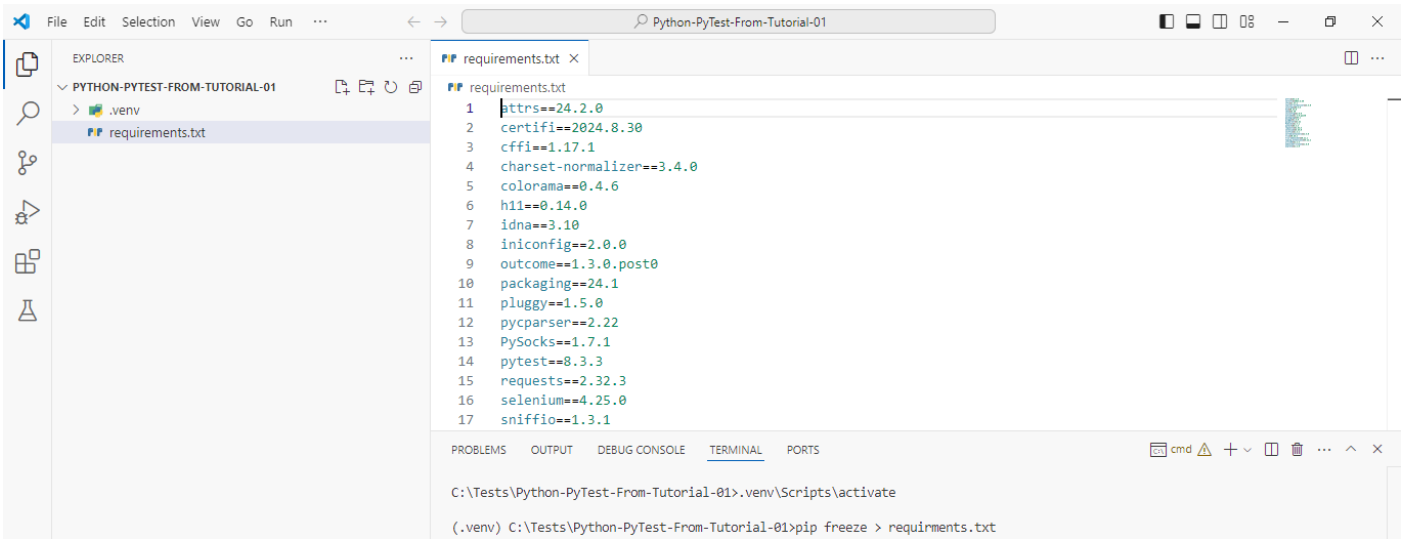
If We Want To Share Our Packages Of Our Projects With Other:

- Run: *pip freeze > requirements.txt*



****

Note 1: We Must Name All Test Modules: *test_name_here.py*.

Note 2: We Must Name All Test Functions: *test_func_name_here*.

Note 3: We Must Name All Test Classes: *Test<Name_Here>*.

****

To Show The Percentage Of Test Completeness:

- Run: pytest -vv

```
(.venv) C:\Tests\Python-PyTest-01>pytest -vv
============================================= test session starts =============================================
platform win32 -- Python 3.12.6, pytest-8.3.3, pluggy-1.5.0 -- C:\Tests\Python-PyTest-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'}, 'Plugins': {'html': '4.1.1', 'metadata': '3.
1.1'}}
rootdir: C:\Tests\Python-PyTest-01
plugins: html-4.1.1, metadata-3.1.1
collected 5 items

test_todo_api.py::test_can_call_endpoint PASSED                                                          [ 20%]
test_todo_api.py::test_can_create_task PASSED                                                            [ 40%]
test_todo_api.py::test_can_update_task PASSED                                                            [ 60%]
test_todo_api.py::test_can_list_user_tasks PASSED                                                        [ 80%]
test_todo_api.py::test_can_delete_task PASSED                                                            [100%]

============================================================================ 5 passed in 14.51s ============================================================================
```

****

Note 1: In PyTest, We Can Run The Fail Tests Using Arguments.

****

Understanding Test Outputs :

- --lf, --last-failed : only re-run the failures.
- --ff, --failed-first : to run the failures first and then the rest of the tests.
- Ref doc pytest_cache: https://docs.pytest.org/en/latest/cache.htm

****

When We Want To Make Test Cases Using Classes:

- The Name Of Class Must Start With *Test*
- The Class Must Not Contain Constructor: *__init__(self)*
- The Test Cases (Functions) Must Start With: *test_*

```
class TestClass:

    def test_t1(self):
        assert 1 == 1;
```

****

If We Want To Check The Exceptions Using The Test Cases Then We Can Use:

- Import: *import pytest*.
- Use raises: *with pytest.raises(Exception):*
- Set The Logic: *assert (1/0)*

Note 1: In This Way We Can Catch The Exceptions In Run-Time.

```
def test_t2(self):
    with pytest.raises(Exception):
        assert (1/0);
```

```
(.venv) C:\Tests\Python-PyTest-From-Tutorial-01>pytest -vvv
========================================================= 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
rootdir: C:\Tests\Python-PyTest-From-Tutorial-01
collected 2 items

test_class_01.py::TestClass::test_t1 PASSED                                                                                  [ 50%]
test_class_01.py::TestClass::test_t2 PASSED                                                                                  [100%]

========================================================== 2 passed in 0.06s ==========================================================
```

****

If We Want To Skip Some Of The Tests That We Don't Want To Run:

- Import: import pytest
- Annotate The Skipped Test With: @pytest.mark.skip(reason='Reason Will Be Here')

```
@pytest.mark.skip(reason="Skip Reason Will be Here")
def test_skip_test(self):
    assert type(1) == int;
```

****

Also, We Can Use Conditional Skipping:

Note 1: Here We Must Set The Reason.

```
@pytest.mark.skipif(sys.version_info.major>=3, reason="No Reason Here")
def test_t1(self):
        assert 1 == 1;
```

****

If We Want To Skip The Full Module, We Add This At The First Line:

```
pytestmark = pytest.mark.skipif(sys.platform == 'win32', reason="No Reason Here")
```

****

If We Want To Mark The Test Function:

- Import: *import pytest*
- Add Annotation With Specific Name: *@pytest.mark.name_will_be_here*
- Create New File: *pytest.ini*
- Add The Marker Name To It, As Displayed In The Picture.

Note 1: Marker Name Can Be Used Multiple Times In The Same File OR Other Files.

Note 2: Test Function Can Have Multiple Marker Names.

File Edit Selection View Go Run ... Python-PyTest-From-Tutorial-01

EXPLORER

PYTHON-PYTEST-FROM-TUTORIAL-01

- __pycache__
- .pytest_cache
- .venv
- pytest.ini
- requirements.txt
- test_class_01.py
- test_markers.py

requirements.txt | test_class_01.py | test_markers.py | pytest.ini

pytest.ini

```
[pytest]
markers=
    fail: For Only Failed Tests That We Want
    jafar_loka_01
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
1

(.venv) C:\Tests\Python-PyTest-From-Tutorial-01>pytest -v -m fail
========================================================= 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
rootdir: C:\Tests\Python-PyTest-From-Tutorial-01
configfile: pytest.ini
collected 8 items / 7 deselected / 1 selected

test_markers.py::test_t2 FAILED                                                                                                  [100%]

============================================================= FAILURES ==============================================================
_____________________________________________________________ test_t2 _____________________________________________________________

    @pytest.mark.jafar_loka_01
    @pytest.mark.fail
    def test_t2():
>       assert 1 > 2;
E       assert 1 > 2

test_markers.py:10: AssertionError
======================================================= short test summary info =======================================================
FAILED test_markers.py::test_t2 - assert 1 > 2
================================================= 1 failed, 7 deselected in 0.14s =================================================
```

requirements.txt | test_class_01.py | test_markers.py | pytest.ini

test_markers.py > test_t2

```
import pytest;

@pytest.mark.jafar_loka_01
def test_t1():
    assert 1 == 1;

@pytest.mark.jafar_loka_01
@pytest.mark.fail
def test_t2():
    assert 1 > 2;

def test_t3():
    assert 2 > 1;

@pytest.mark.jafar_loka_01
def test_t4():
    assert 2 < 3;
```

****

For Markers, We Can Run Tests Using Conditional Statements:

```
(.venv) C:\Tests\Python-PyTest-From-Tutorial-01>pytest -v -m "jafar_loka_01 and not fail"
==================================================== 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
rootdir: C:\Tests\Python-PyTest-From-Tutorial-01
configfile: pytest.ini
collected 8 items / 6 deselected / 2 selected

test_markers.py::test_t1 PASSED                                                                                            [ 50%]
test_markers.py::test_t4 PASSED                                                                                            [100%]

================================================= 2 passed, 6 deselected in 0.04s ==================================================
```

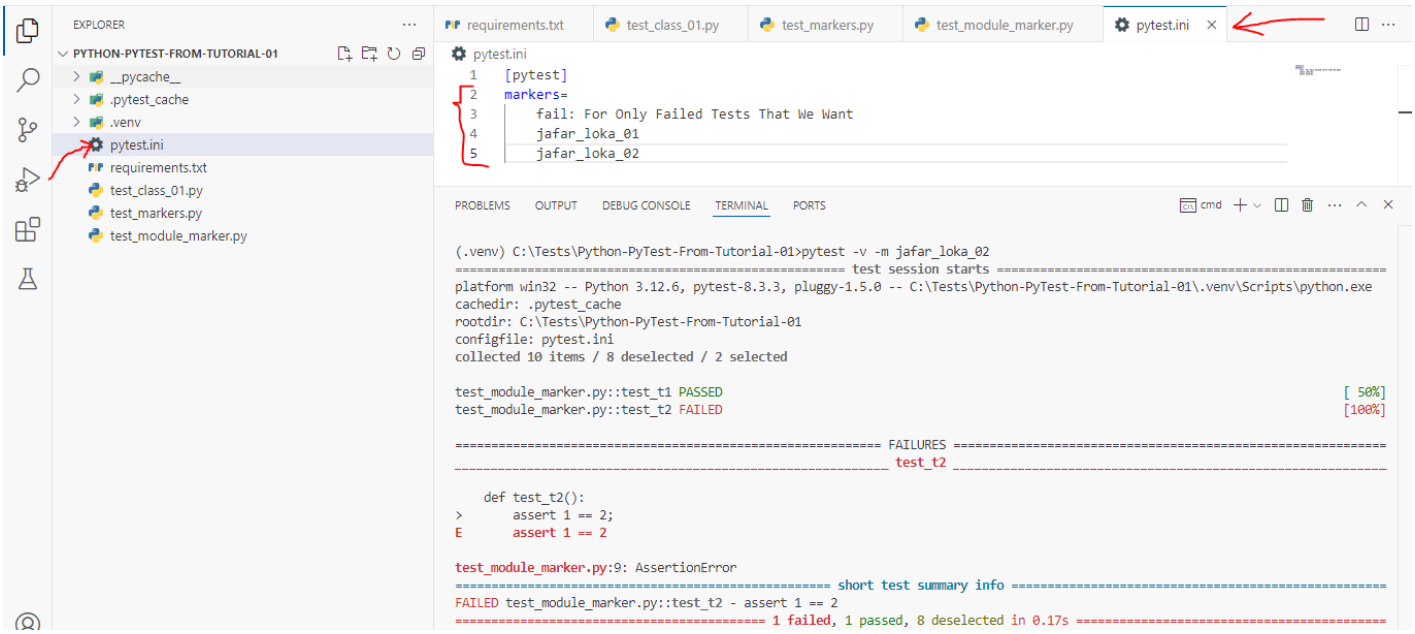
```
(.venv) C:\Tests\Python-PyTest-From-Tutorial-01>pytest -m "not fail"
==================================================== test session starts ====================================================
platform win32 -- Python 3.12.6, pytest-8.3.3, pluggy-1.5.0
rootdir: C:\Tests\Python-PyTest-From-Tutorial-01
configfile: pytest.ini
collected 8 items / 1 deselected / 7 selected

test_class_01.py sss                                                                                                       [ 42%]
test_markers.py ...F                                                                                                       [100%]

========================================================= FAILURES ==========================================================
_____________________________________________________________ test_t5 _____________________________________________________________

    def test_t5():
>       assert 3 < 2;
E       assert 3 < 2

test_markers.py:20: AssertionError
================================================= short test summary info ===================================================
FAILED test_markers.py::test_t5 - assert 3 < 2
===================================== 1 failed, 3 passed, 3 skipped, 1 deselected in 0.17s ======================================
```

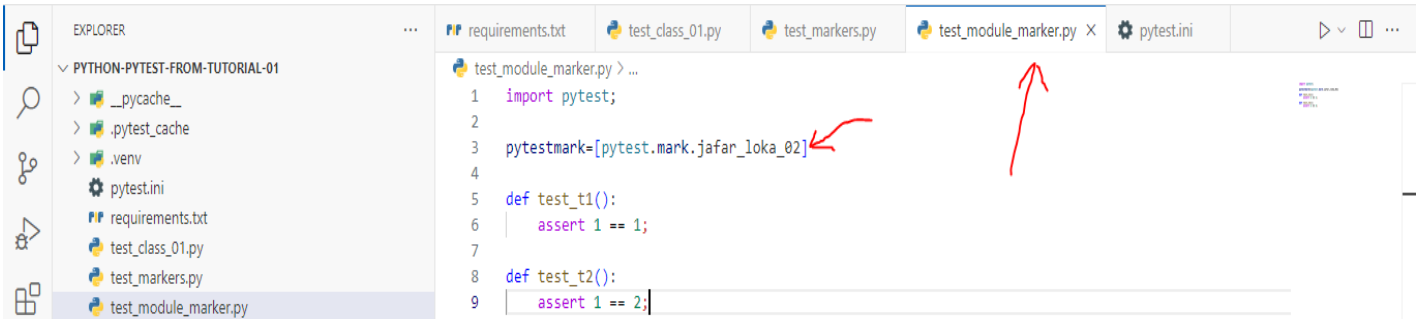
```
(.venv) C:\Tests\Python-PyTest-From-Tutorial-01>pytest -vv -m "jafar_loka_01 and fail"
==================================================== test session starts ====================================================
platform win32 -- Python 3.12.6, pytest-8.3.3, pluggy-1.5.0 -- C:\Tests\Python-PyTest-From-Tutorial-01\.venv\Scripts\python.exe
cachedir: .pytest_cache
rootdir: C:\Tests\Python-PyTest-From-Tutorial-01
configfile: pytest.ini
collected 8 items / 7 deselected / 1 selected

test_markers.py::test_t2 FAILED                                                                                            [100%]

========================================================= FAILURES ==========================================================
_____________________________________________________________ test_t2 _____________________________________________________________

    @pytest.mark.jafar_loka_01
    @pytest.mark.fail
    def test_t2():
>       assert 1 > 2;
E       assert 1 > 2

test_markers.py:10: AssertionError
================================================= short test summary info ===================================================
FAILED test_markers.py::test_t2 - assert 1 > 2
================================================= 1 failed, 7 deselected in 0.13s ===================================================
```

****

If We Want To Mark The Complete Module With Specific Mark:

- Import: *import pytest*.
- Add The Line: pytestmark=[…Marks Will Be Here…]
- Create New File: pytest.ini
- Add The Conf. To The *pytest.ini-File*
- Run The Tests.





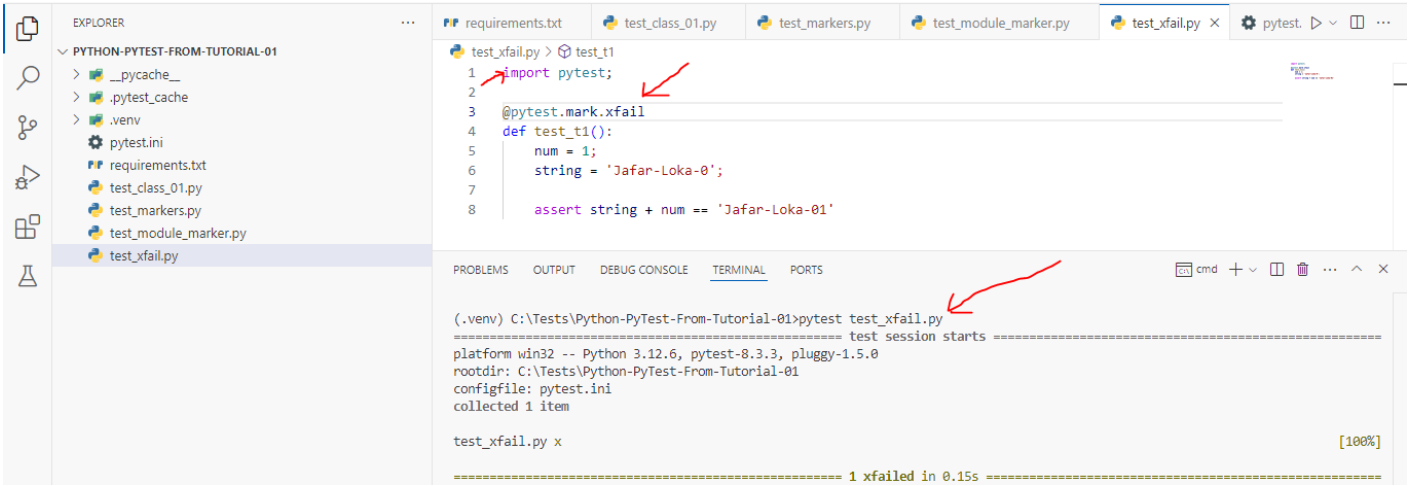
****

Note: We Must Define The pytest.ini-File At The Root Of The Folder, Else It Will Not Worked.

****

If We Want To Test Any Function That We Know It Will Fail, Then We Use *xfail*.

Note: We Use Here The Same Steps.




```
(.venv) C:\Tests\Python-PyTest-From-Tutorial-01>pytest -v test_xfail.py
================================================= 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
rootdir: C:\Tests\Python-PyTest-From-Tutorial-01
configfile: pytest.ini
collected 1 item

test_xfail.py::test_t1 XFAIL                                                                                         [100%]

================================================= 1 xfailed in 0.13s ==================================================
```

****

Note: If Any Test Function Marked With xfail, And The Result is Passed, Then The Final Result Of Testing is *XPass*, Not *XFAIL*.

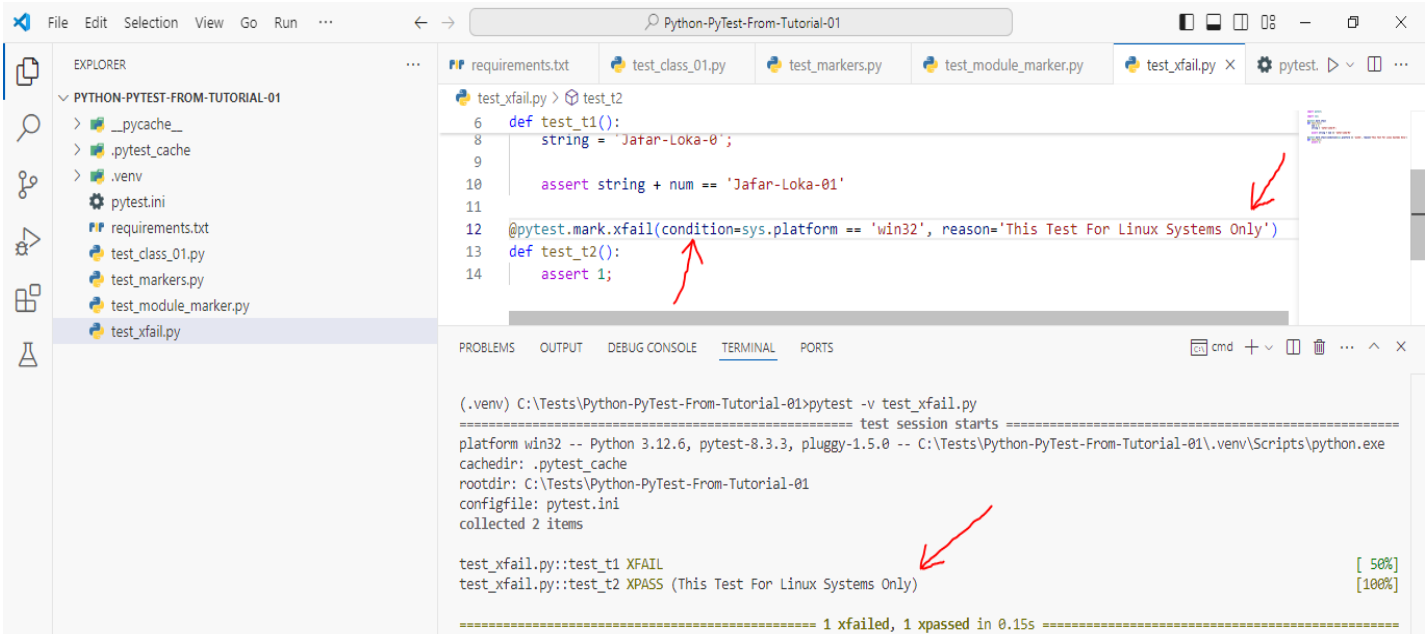
****

Note: We Can Add Reason To xfail-Annotation.

****

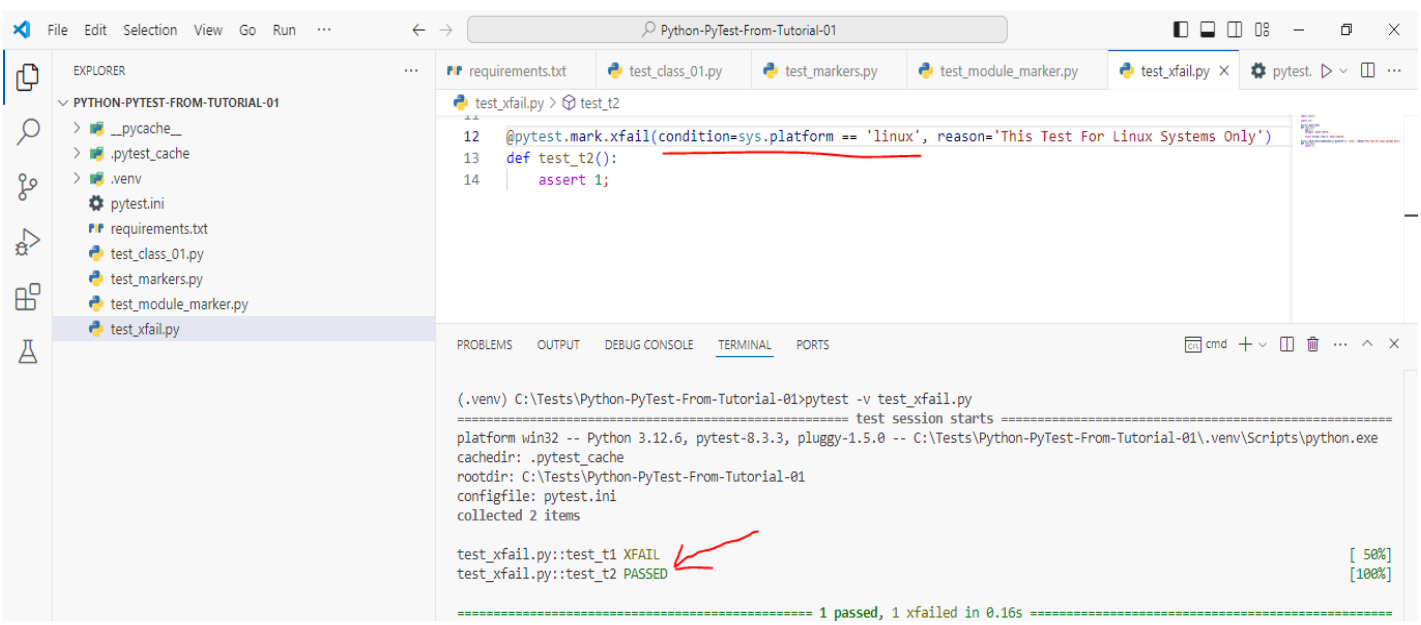
Note: We Can Also Use Conditional Statement With XFail.

****



****

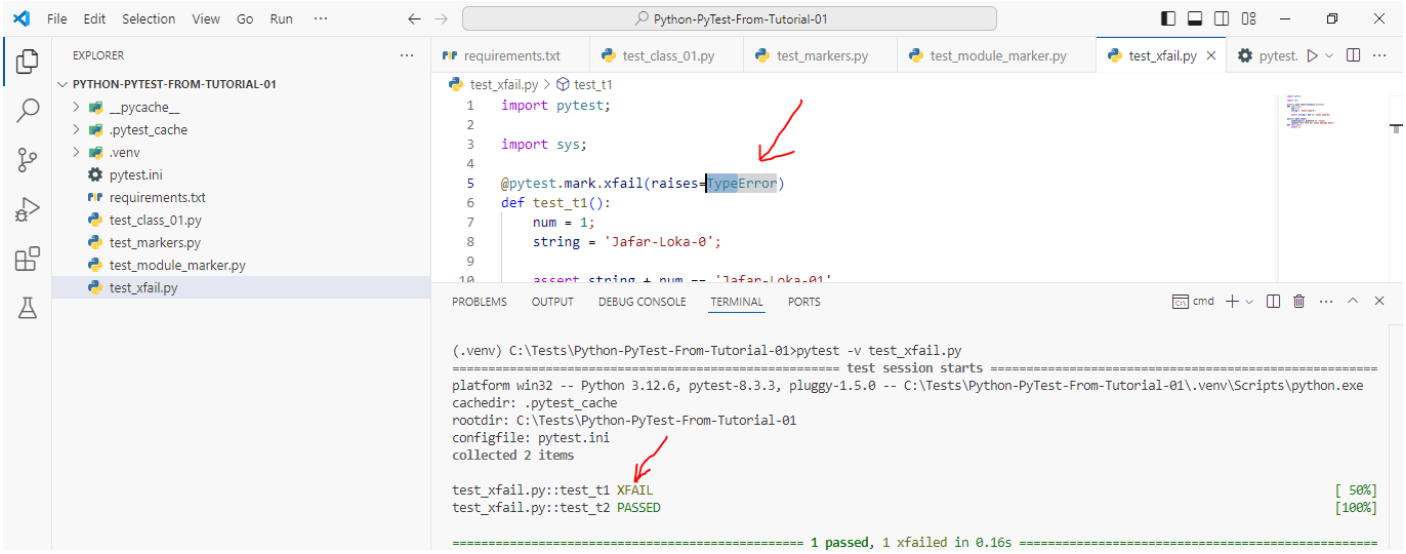
Note: If The Condition Is Not True, Then The Test Will Run As Any Normal Test.



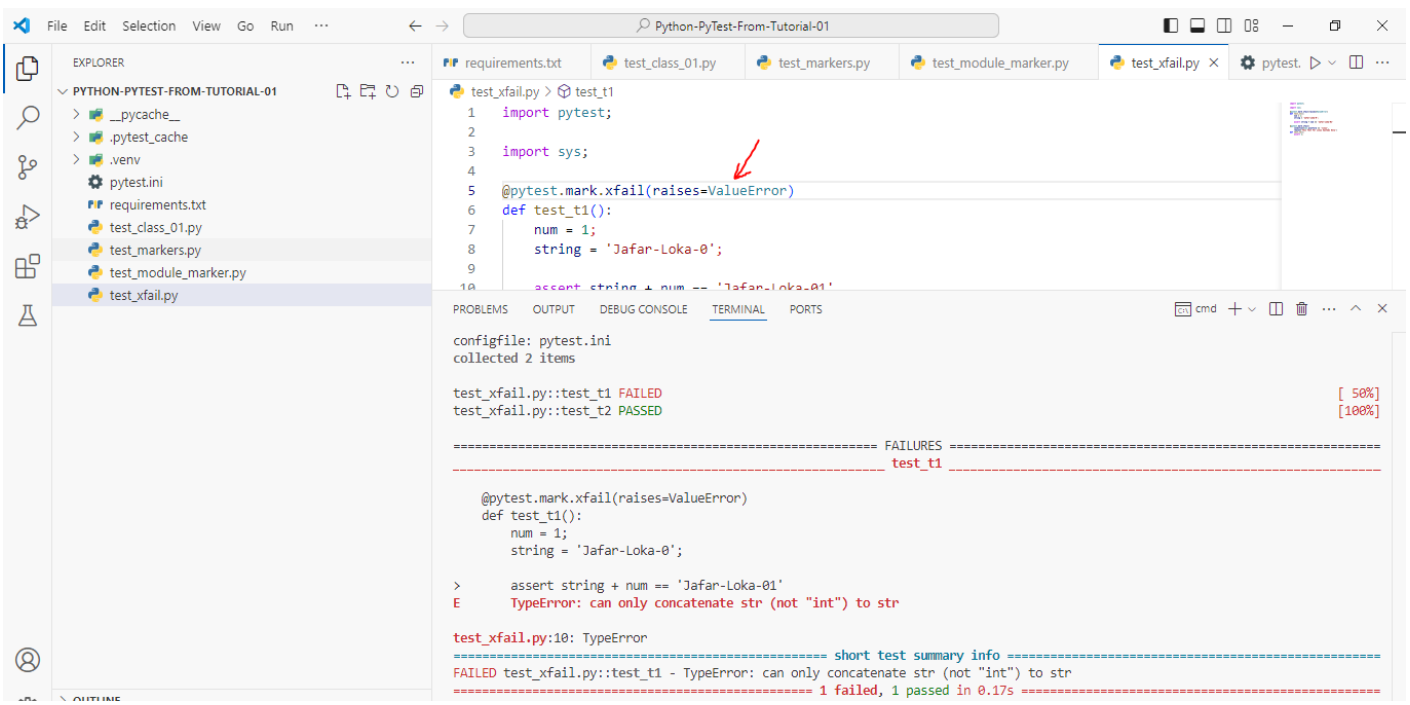
****

Note 1: We Can Also Add raises To Xfail-Marker Annotation, And it Work As Same As Condition.

****


File Edit Selection View Go Run
Python-PyTest-From-Tutorial-01
EXPLORER
PYTHON-PYTEST-FROM-TUTORIAL-01
__pycache__
.pytest_cache
.venv
pytest.ini
requirements.txt
test_class_01.py
test_markers.py
test_module_marker.py
test_xfail.py
requirements.txt
test_class_01.py
test_markers.py
test_module_marker.py
test_xfail.py
test_xfail.py > test_t1
import pytest;
import sys;
@pytest.mark.xfail(raises=TypeError)
def test_t1():
num = 1;
string = 'Jafar-Loka-0';
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
(.venv) C:\Tests\Python-PyTest-From-Tutorial-01>pytest -v test_xfail.py
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
rootdir: C:\Tests\Python-PyTest-From-Tutorial-01
configfile: pytest.ini
collected 2 items
test_xfail.py::test_t1 XFAIL [50%]
test_xfail.py::test_t2 PASSED [100%]
1 passed, 1 xfailed in 0.16s

****


File Edit Selection View Go Run
Python-PyTest-From-Tutorial-01
EXPLORER
PYTHON-PYTEST-FROM-TUTORIAL-01
__pycache__
.pytest_cache
.venv
pytest.ini
requirements.txt
test_class_01.py
test_markers.py
test_module_marker.py
test_xfail.py
requirements.txt
test_class_01.py
test_markers.py
test_module_marker.py
test_xfail.py
test_xfail.py > test_t1
import pytest;
import sys;
@pytest.mark.xfail(raises=ValueError)
def test_t1():
num = 1;
string = 'Jafar-Loka-0';
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
configfile: pytest.ini
collected 2 items
test_xfail.py::test_t1 FAILED [50%]
test_xfail.py::test_t2 PASSED [100%]
FAILURES
test_t1
@pytest.mark.xfail(raises=ValueError)
def test_t1():
num = 1;
string = 'Jafar-Loka-0';
> assert string + num == 'Jafar-Loka-01'
E TypeError: can only concatenate str (not "int") to str
test_xfail.py:10: TypeError
short test summary info
FAILED test_xfail.py::test_t1 - TypeError: can only concatenate str (not "int") to str
1 failed, 1 passed in 0.17s

****