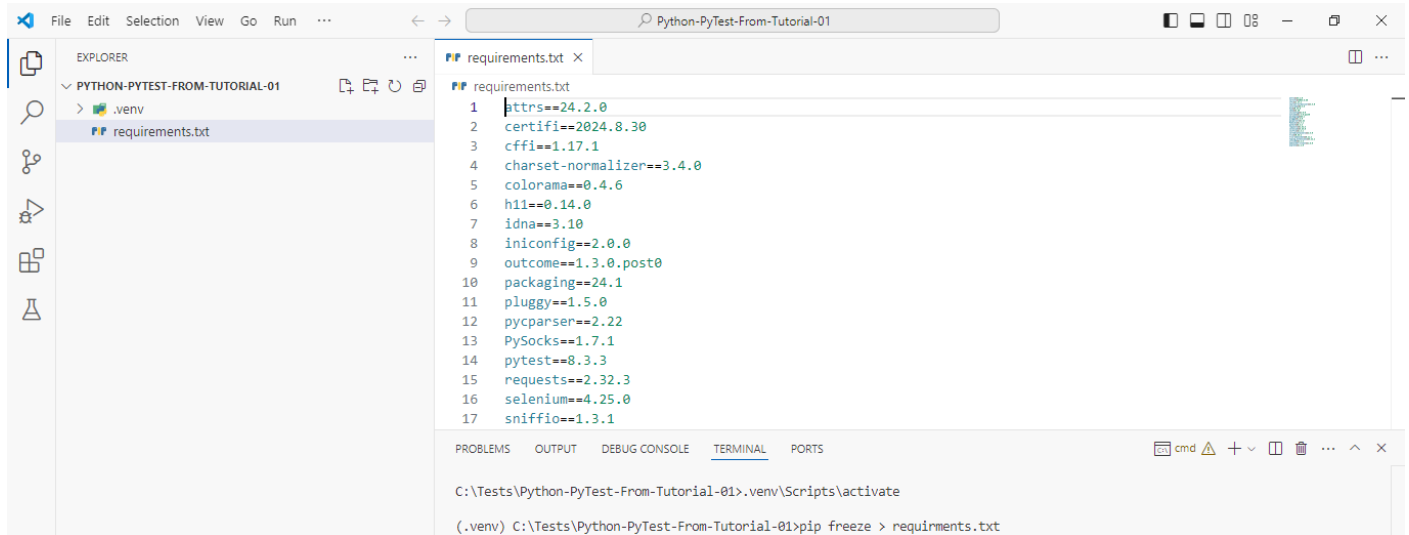


If We Want To Share Our Packages Of Our Projects With Other:

- Run: `pip freeze > requirements.txt`



The screenshot shows a Visual Studio Code editor window titled 'Python-PyTest-From-Tutorial-01'. The Explorer sidebar on the left shows a project folder with a subfolder '.venv' and a file 'requirements.txt'. The main editor area displays the contents of 'requirements.txt', which lists 17 dependencies: http==24.2.0, certifi==2024.8.30, cffi==1.17.1, charset-normalizer==3.4.0, colorama==0.4.6, h11==0.14.0, idna==3.10, ini==2.0.0, outcome==1.3.0.post0, packaging==24.1, plugy==1.5.0, pycparser==2.22, PySocks==1.7.1, pytest==8.3.3, requests==2.32.3, selenium==4.25.0, and sniffio==1.3.1. The bottom panel shows the TERMINAL with the command '(.venv) C:\Tests\Python-PyTest-From-Tutorial-01>pip freeze > requirements.txt'.

\*\*\*\*\*

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

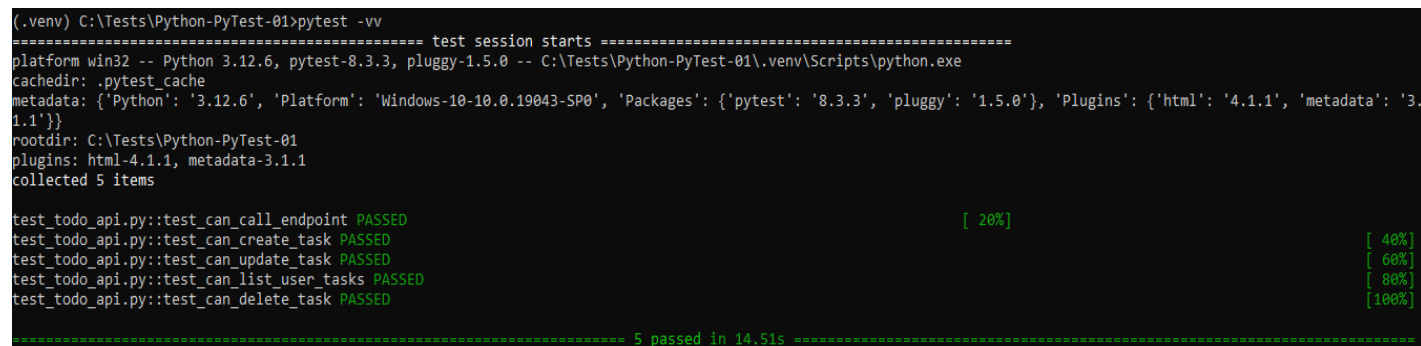
**Note 2:** We Must Name All Test Functions: `test_func_name_here`.

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

\*\*\*\*\*

To Show The Percentage Of Test Completeness:

- Run: `pytest -vv`



The screenshot shows a terminal window with the command '(.venv) C:\Tests\Python-PyTest-01>pytest -vv'. The output displays test session information, including platform (win32), Python version (3.12.6), and pytest version (8.3.3). It then lists 5 collected items and shows the progress of each test: 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%], and test\_todo\_api.py::test\_can\_delete\_task PASSED [100%]. The final summary shows '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.

The screenshot shows the VS Code interface with the file explorer on the left and the editor on the right. The file explorer shows the project structure for 'PYTHON-PYTEST-FROM-TUTORIAL-01', including files like `__pycache__`, `.pytest_cache`, `.venv`, `pytest.ini`, `requirements.txt`, `test_class_01.py`, and `test_markers.py`. The `pytest.ini` file is selected in the editor, showing the following content:

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

The terminal window at the bottom shows the output of the command `pytest -v -m fail` executed in the `.venv` environment. The output indicates that the test session started successfully, collected 8 items, and 7 were deselected, leaving 1 selected. The test `test_t2` in `test_markers.py` failed with an `AssertionError`. The failure details show that the test function `test_t2` contains the assertion `assert 1 > 2`, which is false. The terminal output also includes a short test summary info showing 1 failed, 7 deselected in 0.14s.

The screenshot shows the VS Code interface with the file explorer on the left and the editor on the right. The file explorer shows the project structure for 'PYTHON-PYTEST-FROM-TUTORIAL-01', including files like `__pycache__`, `.pytest_cache`, `.venv`, `pytest.ini`, `requirements.txt`, `test_class_01.py`, and `test_markers.py`. The `test_markers.py` file is selected in the editor, showing the following content:

```
1 import pytest;
2
3 @pytest.mark.jafar_loka_01
4 def test_t1():
5     assert 1 == 1;
6
7 @pytest.mark.jafar_loka_01
8 @pytest.mark.fail
9 def test_t2():
10     assert 1 > 2;
11
12 def test_t3():
13     assert 2 > 1;
14
15 @pytest.mark.jafar_loka_01
16 def test_t4():
17     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 -----

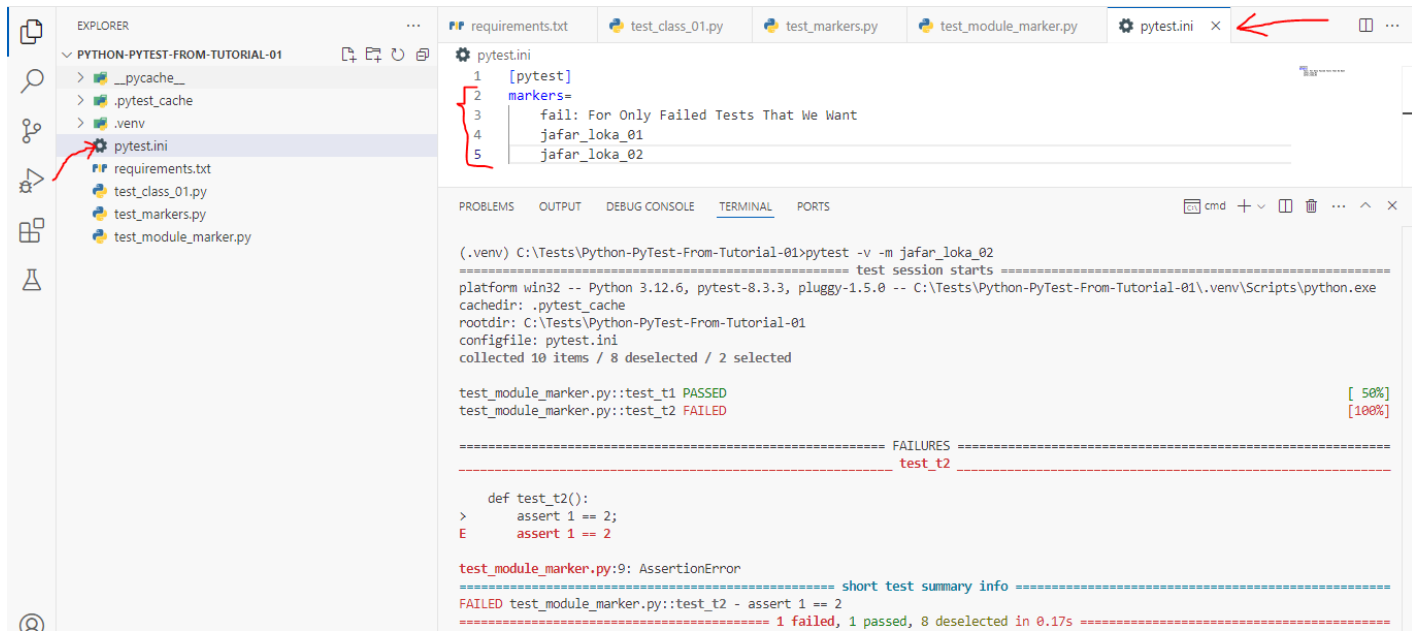
    @pytest.mark.jafar_loka_01
    @pytest.mark.fail
    def test_t2():
>     assert 1 > 2;
E       assert 1 > 2

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.



The screenshot shows the VS Code interface. In the Explorer pane, the file `pytest.ini` is selected. The main editor shows the content of `pytest.ini`:

```
1 [pytest]
2 markers=
3     fail: For Only Failed Tests That We Want
4     jafar_loka_01
5     jafar_loka_02
```

The terminal pane shows the output of running `pytest -v -m jafar_loka_02`:

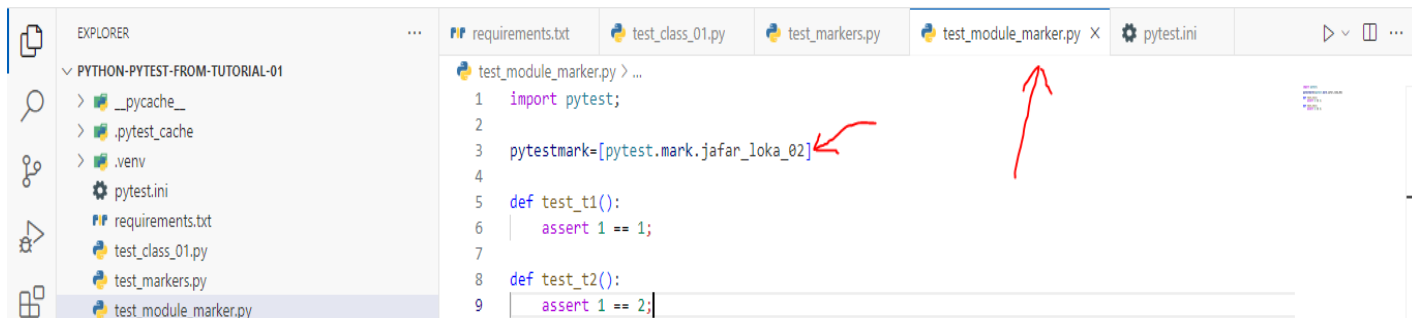
```
(.venv) C:\Tests\Python-PyTest-From-Tutorial-01>pytest -v -m jafar_loka_02
===== test session starts =====
platform win32 -- Python 3.12.6, pytest-8.3.3, pluggy-1.5.0 -- C:\Tests\Python-PyTest-From-Tutorial-01\.venv\Scripts\python.exe
cachedir: .pytest_cache
rootdir: C:\Tests\Python-PyTest-From-Tutorial-01
configfile: pytest.ini
collected 10 items / 8 deselected / 2 selected

test_module_marker.py::test_t1 PASSED [ 50%]
test_module_marker.py::test_t2 FAILED [100%]

===== FAILURES =====
test_t2

def test_t2():
> assert 1 == 2;
E assert 1 == 2

test_module_marker.py:9: AssertionError
===== short test summary info =====
FAILED test_module_marker.py::test_t2 - assert 1 == 2
===== 1 failed, 1 passed, 8 deselected in 0.17s =====
```



The screenshot shows the VS Code interface with the file `test_module_marker.py` selected in the Explorer pane. The main editor shows the content of `test_module_marker.py`:

```
1 import pytest;
2
3 pytestmark=[pytest.mark.jafar_loka_02]
4
5 def test_t1():
6     assert 1 == 1;
7
8 def test_t2():
9     assert 1 == 2;
```

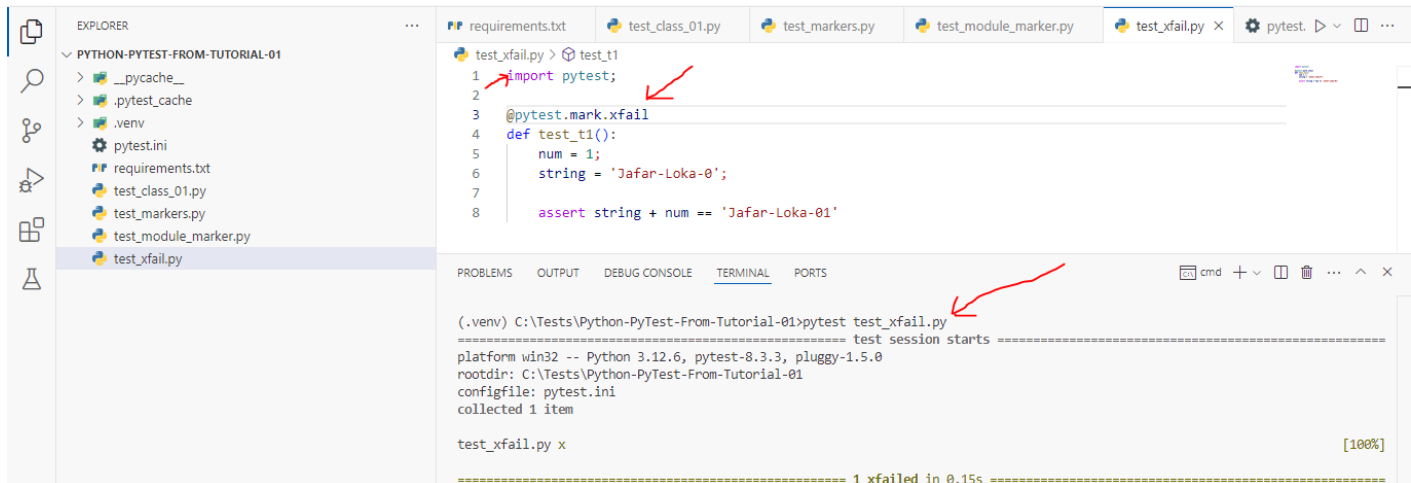
\*\*\*\*\*

**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.



```
test_xfail.py > test_t1
1 import pytest;
2
3 @pytest.mark.xfail
4 def test_t1():
5     num = 1;
6     string = 'Jafar-Loka-0';
7
8     assert string + num == 'Jafar-Loka-01'
```

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

test_xfail.py x

===== 1 xfailed in 0.15s =====
```

```
(.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

===== 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.

\*\*\*\*\*



```
test_xfail.py > test_t2
6 def test_t1():
8     string = 'Jafar-Loka-0';
9
10     assert string + num == 'Jafar-Loka-01'
11
12 @pytest.mark.xfail(condition=sys.platform == 'win32', reason='This Test For Linux Systems Only')
13 def test_t2():
14     assert 1;
```

```
(.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
test_xfail.py::test_t2 XPASS (This Test For Linux Systems Only)

===== 1 xfailed, 1 xpassed in 0.15s =====
```

\*\*\*\*\*

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

```
test_xfail.py > test_t2
12 @pytest.mark.xfail(condition=sys.platform == 'linux', reason='This Test For Linux Systems Only')
13 def test_t2():
14     assert 1;
```

```
(.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
test_xfail.py::test_t2 PASSED

===== 1 passed, 1 xfailed in 0.16s =====
```

\*\*\*\*\*

**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

test\_xfail.py

```
1 import pytest;
2
3 import sys;
4
5 @pytest.mark.xfail(raises=TypeError)
6 def test_t1():
7     num = 1;
8     string = 'Jafar-Loka-0';
9
10     assert string + num == 'Jafar-Loka-01'
```

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

test\_xfail.py

```
1 import pytest;
2
3 import sys;
4
5 @pytest.mark.xfail(raises=ValueError)
6 def test_t1():
7     num = 1;
8     string = 'Jafar-Loka-0';
9
10     assert string + num == 'Jafar-Loka-01'
```

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 =====

\*\*\*\*\*