PyTest Markers

PyTest uses a decorator called "**PyTest.Mark**" to add the syntactical meta data to tests.

What is syntactical meta data?

- The shortest definition for the term meta data is "data about data" or "information about the information".
- Syntactical meta data means, the data about the data which has a pre-defined meaning.
- One of the most important uses for metadata is to locate a resource.

Note:

- Markers are useful to note the related tests and to select the group of tests to be run.

Markers are categorised into two types.

1. Built-in markers.
2. Custom markers.

In-built markers:

Markers which are provided by PyTest are called as built-in markers.

Following are the markers provided by PyTest.

1. @pytest.mark.skip (reason="")
2. @pytest.mark.skipif (condition, reason="")
3. @pytest.mark.parametrise (arg_names, arg_values)
4. @pytest.mark.usefixtures (fixuturename1, ….)
5. @pytest.mark.tryfirst
6. @pytest.mark.trylast

Skip Marker:

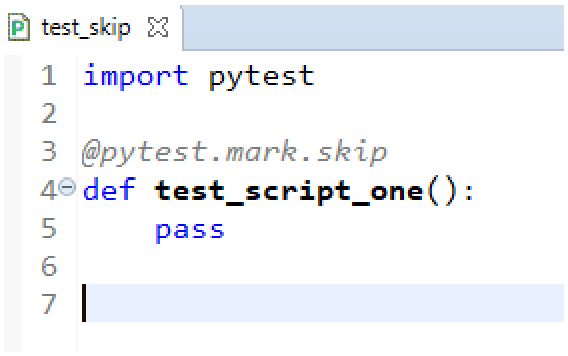
Skip marker is used to skip the test function with optional reason argument.

Note:

When you want to skip a test method,

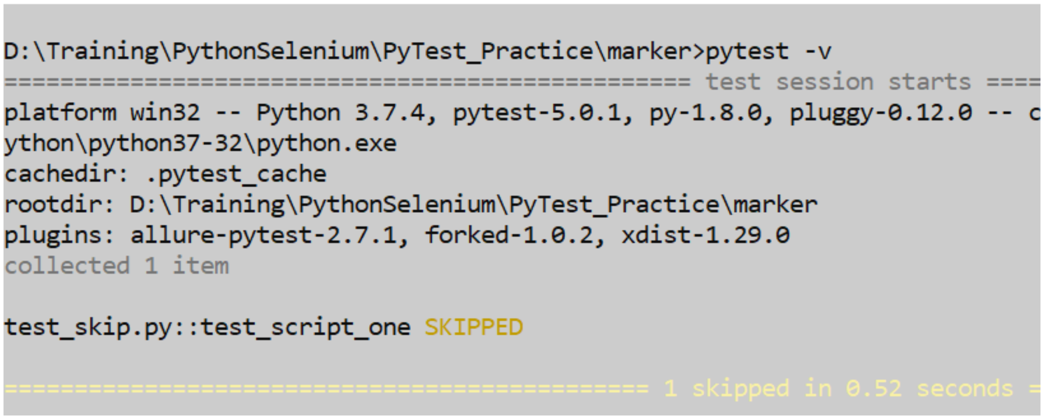
- When test development is not completed.
- When test has a defect, coz of that you don't want to run a test function. We can use skip marker to achieve.
- Reason is optional argument. But recommended to pass the reason, while skipping the test functions.

Program to skip the test function?



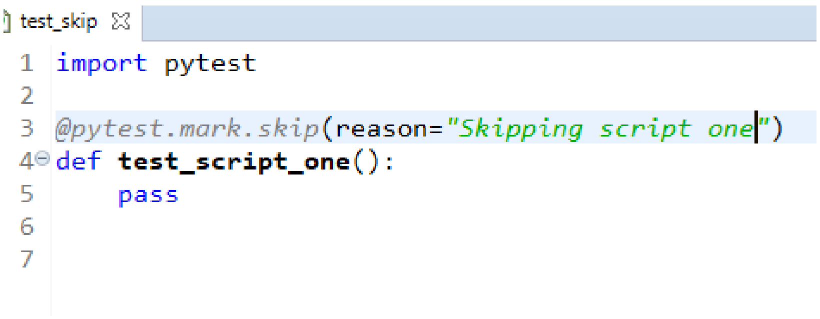
```
test_skip

import pytest

@pytest.mark.skip
def test_script_one():
    pass

```

Open the command prompt: run the script.



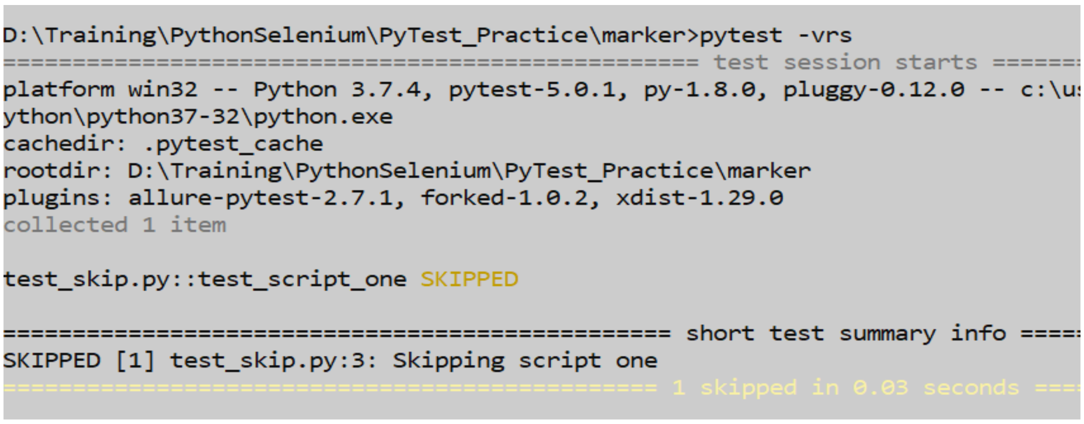
```
D:\Training\PythonSelenium\PyTest_Practice\marker>pytest -v
=================================================== test session starts ====
platform win32 -- Python 3.7.4, pytest-5.0.1, py-1.8.0, pluggy-0.12.0 -- c
ython\python37-32\python.exe
cachedir: .pytest_cache
rootdir: D:\Training\PythonSelenium\PyTest_Practice\marker
plugins: allure-pytest-2.7.1, forked-1.0.2, xdist-1.29.0
collected 1 item

test_skip.py::test_script_one SKIPPED

=============================================== 1 skipped in 0.52 seconds =
```

Test function is skipped since we have used the skip marker.



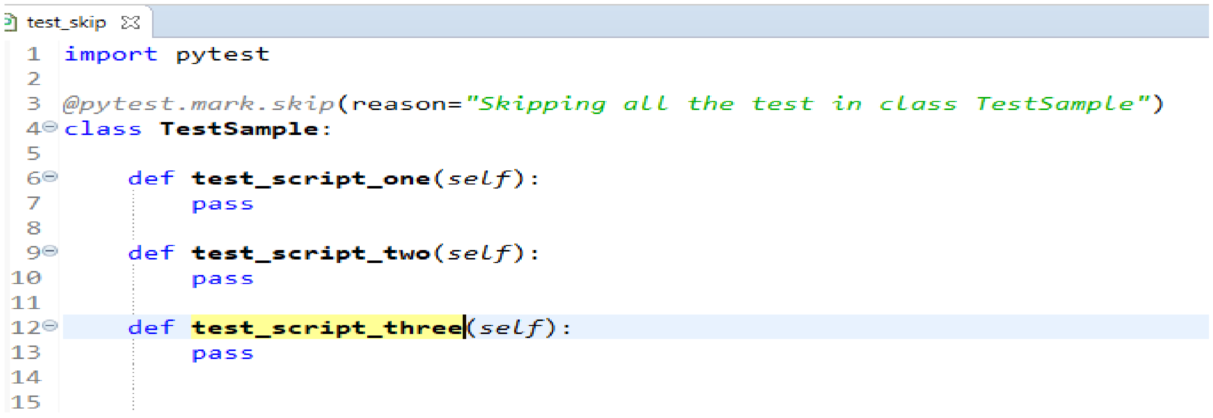
```
test_skip

import pytest

@pytest.mark.skip(reason="Skipping script one")
def test_script_one():
    pass

```

Note: To see the reason on console, we need to use the extra summery commands.



```
D:\Training\PythonSelenium\PyTest_Practice\marker>pytest -vrs
=================================================== test session starts =======
platform win32 -- Python 3.7.4, pytest-5.0.1, py-1.8.0, pluggy-0.12.0 -- c:\u
ython\python37-32\python.exe
cachedir: .pytest_cache
rootdir: D:\Training\PythonSelenium\PyTest_Practice\marker
plugins: allure-pytest-2.7.1, forked-1.0.2, xdist-1.29.0
collected 1 item

test_skip.py::test_script_one SKIPPED

================================================ short test summary info ====
SKIPPED [1] test_skip.py:3: Skipping script one
============================================== 1 skipped in 0.03 seconds ===
```

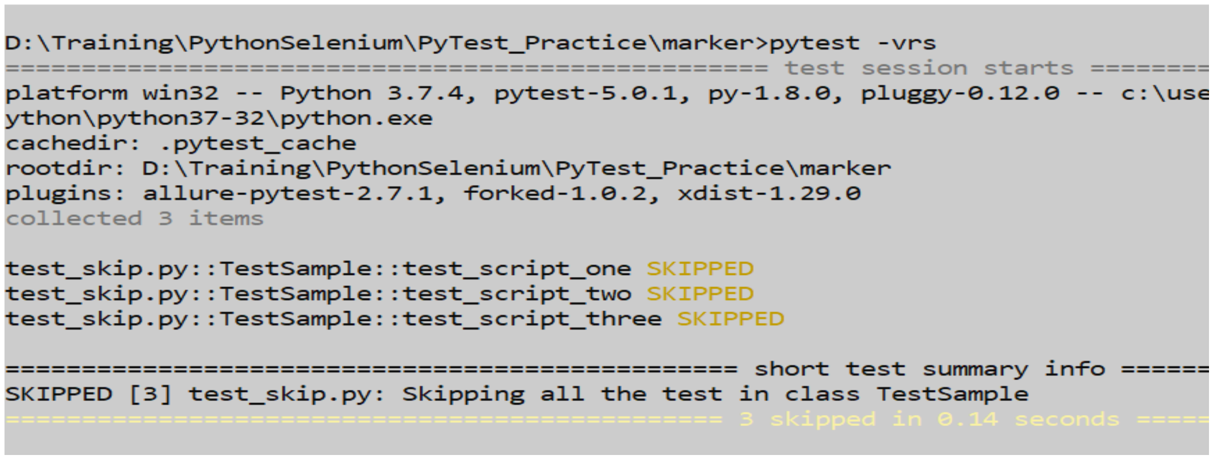
Skipping all the functions which is inside the class.



test_skip

```python
import pytest

@pytest.mark.skip(reason="Skipping all the test in class TestSample")
class TestSample:

    def test_script_one(self):
        pass

    def test_script_two(self):
        pass

    def test_script_three(self):
        pass


```

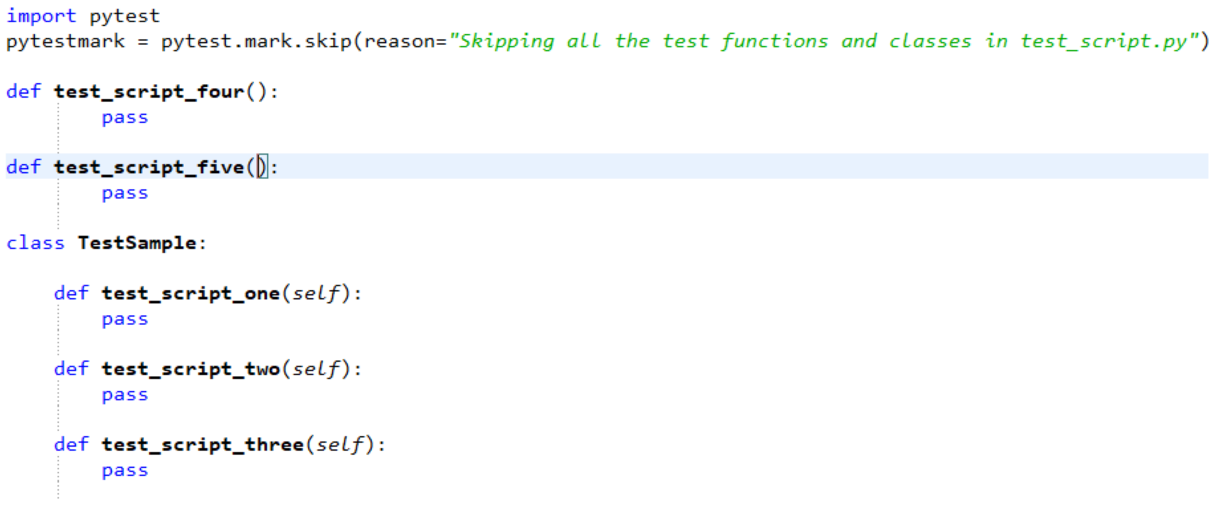
Output:



```
D:\Training\PythonSelenium\PyTest_Practice\marker>pytest -vrs
===================================================== test session starts ========
platform win32 -- Python 3.7.4, pytest-5.0.1, py-1.8.0, pluggy-0.12.0 -- c:\use
ython\python37-32\python.exe
cachedir: .pytest_cache
rootdir: D:\Training\PythonSelenium\PyTest_Practice\marker
plugins: allure-pytest-2.7.1, forked-1.0.2, xdist-1.29.0
collected 3 items

test_skip.py::TestSample::test_script_one SKIPPED
test_skip.py::TestSample::test_script_two SKIPPED
test_skip.py::TestSample::test_script_three SKIPPED

================================================ short test summary info ======
SKIPPED [3] test_skip.py: Skipping all the test in class TestSample
============================================== 3 skipped in 0.14 seconds =====
```

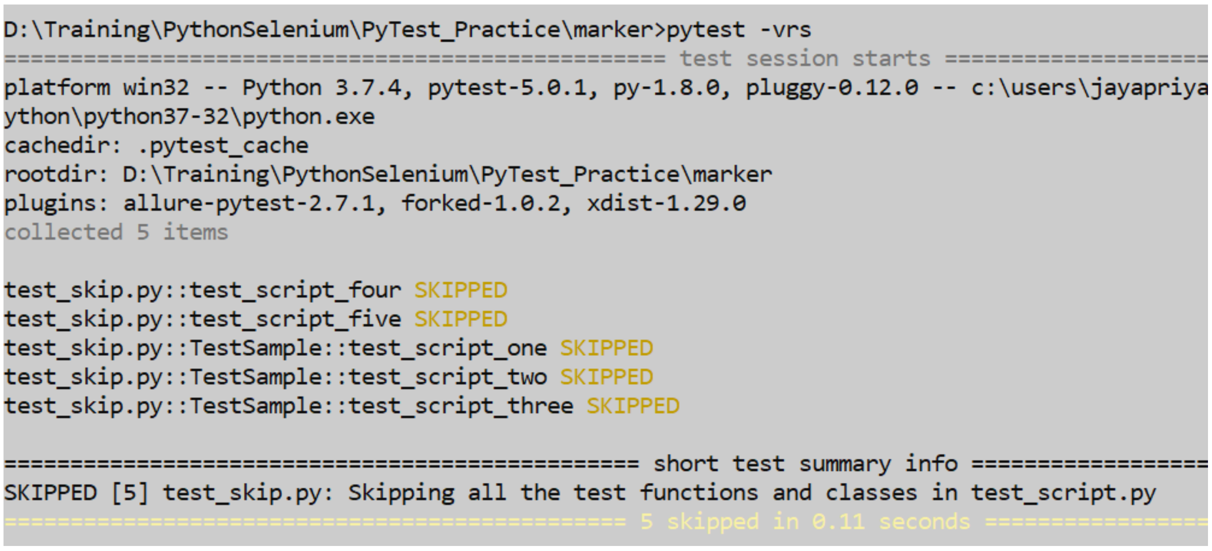
Skipping all the test function in a module

- In order to skip all the functions of any python module (file). We use “pytestmark” attribute below the last import statements.



```python
import pytest
pytestmark = pytest.mark.skip(reason="Skipping all the test functions and classes in test_script.py")

def test_script_four():
        pass

def test_script_five():
        pass

class TestSample:

    def test_script_one(self):
        pass

    def test_script_two(self):
        pass

    def test_script_three(self):
        pass
```

Output:



```
D:\Training\PythonSelenium\PyTest_Practice\marker>pytest -vrs
================================================== test session starts ====================
platform win32 -- Python 3.7.4, pytest-5.0.1, py-1.8.0, pluggy-0.12.0 -- c:\users\jayapriya
ython\python37-32\python.exe
cachedir: .pytest_cache
rootdir: D:\Training\PythonSelenium\PyTest_Practice\marker
plugins: allure-pytest-2.7.1, forked-1.0.2, xdist-1.29.0
collected 5 items

test_skip.py::test_script_four SKIPPED
test_skip.py::test_script_five SKIPPED
test_skip.py::TestSample::test_script_one SKIPPED
test_skip.py::TestSample::test_script_two SKIPPED
test_skip.py::TestSample::test_script_three SKIPPED

================================================ short test summary info ==================
SKIPPED [5] test_skip.py: Skipping all the test functions and classes in test_script.py
=============================================== 5 skipped in 0.11 seconds =================
```

http://doc.pytest.org/en/latest/example/pythoncollection.html#customizing-test-collection

http://doc.pytest.org/en/latest/skipping.html

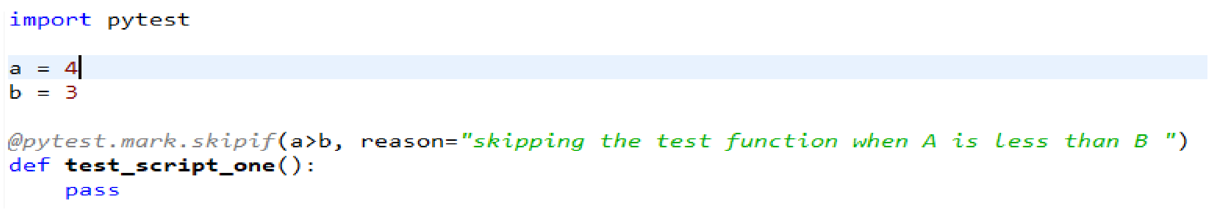
Skip-If marker

- In order to skip test functions based on some condition. Use skip-if marker.
- Skip-if marker skip the test function if the condition results in true.

Note:

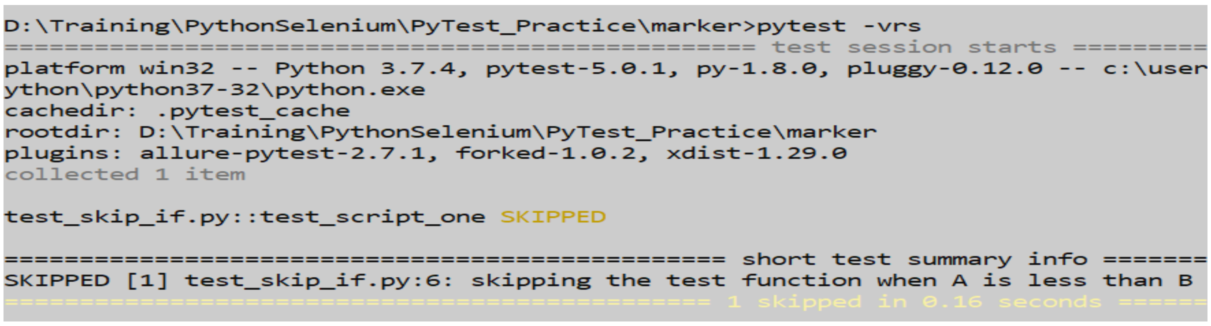
- If the condition returns true: skips the test function.
- If the condition returns false: executes the test function.

Program to skip the test function based on condition.



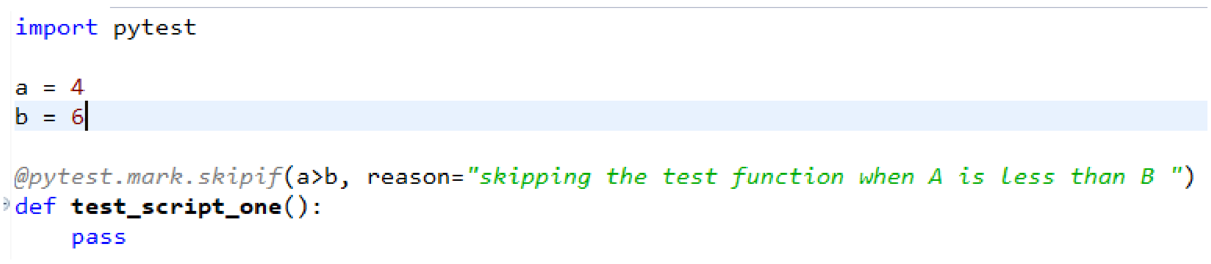
```
import pytest

a = 4
b = 3

@pytest.mark.skipif(a>b, reason="skipping the test function when A is less than B ")
def test_script_one():
    pass
```

Output:



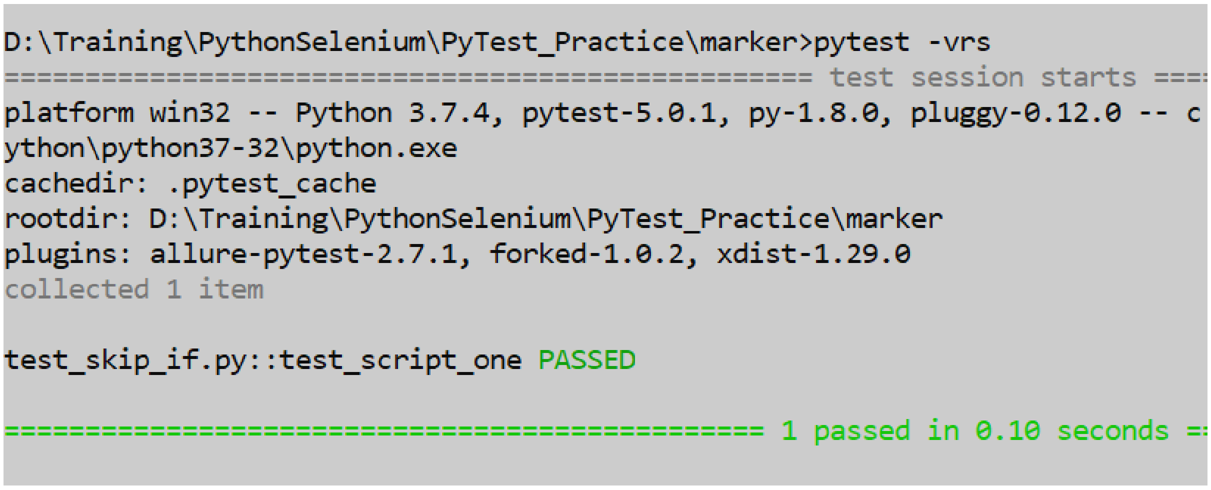
```
D:\Training\PythonSelenium\PyTest_Practice\marker>pytest -vrs
==================================================== test session starts =========
platform win32 -- Python 3.7.4, pytest-5.0.1, py-1.8.0, pluggy-0.12.0 -- c:\user
ython\python37-32\python.exe
cachedir: .pytest_cache
rootdir: D:\Training\PythonSelenium\PyTest_Practice\marker
plugins: allure-pytest-2.7.1, forked-1.0.2, xdist-1.29.0
collected 1 item

test_skip_if.py::test_script_one SKIPPED

================================================ short test summary info =======
SKIPPED [1] test_skip_if.py:6: skipping the test function when A is less than B
=============================================== 1 skipped in 0.16 seconds ======
```

If the value of B becomes greater than A, than test script will execute, coz the condition fails.



```python
import pytest

a = 4
b = 6

@pytest.mark.skipif(a>b, reason="skipping the test function when A is less than B ")
def test_script_one():
    pass
```

Output:



```
D:\Training\PythonSelenium\PyTest_Practice\marker>pytest -vrs
==================================================== test session starts ====
platform win32 -- Python 3.7.4, pytest-5.0.1, py-1.8.0, pluggy-0.12.0 -- c
ython\python37-32\python.exe
cachedir: .pytest_cache
rootdir: D:\Training\PythonSelenium\PyTest_Practice\marker
plugins: allure-pytest-2.7.1, forked-1.0.2, xdist-1.29.0
collected 1 item

test_skip_if.py::test_script_one PASSED

============================================== 1 passed in 0.10 seconds ==
```

Note:

1. To skip all the functions of a class, then decorate the class with the marker. Then all the functions of that class will be skipped if the conditions return true.
2. To skip all the functions of python files, then use the ‘pytestmark’ attribute below the last import statement and assign the decorator to pytestmark. If condition return true then all the functions of that python file will be skipped else all test functions will execute.

Parameterization of test functions

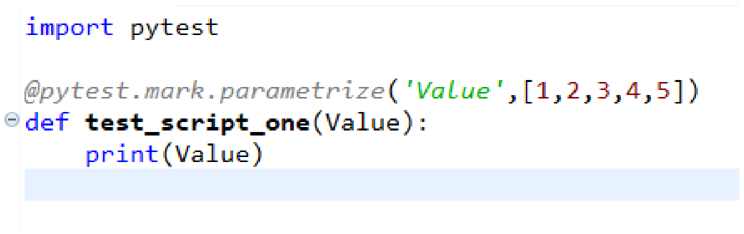
The built-in “**pytest.mark.parametrize**” decorator enables parametrization of arguments for a test function.

- Call a test function multiple time with different arguments intern.

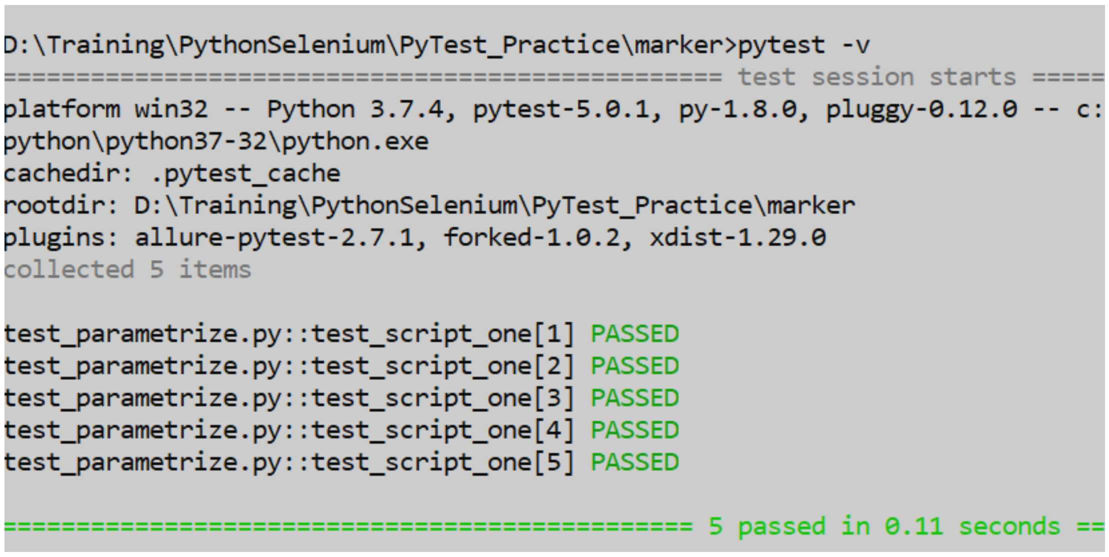
- Parametrize takes two arguments, arg_names & arg_values.
- Arg_values generally need to be a list of values if arg_names specify only one name or a list of tuples of values if arg_names specifies multiple names.

Example: @parametrize ('arg1', [1, 2]) would lead to two calls of the decorated test function, one with arg1 = 1 and another arg1=2.

Note: Based on the number of arguments passed to parametrize decorator, that many numbers of times the test function will be executed by the PyTest.



```python
import pytest

@pytest.mark.parametrize('Value',[1,2,3,4,5])
def test_script_one(Value):
    print(Value)
```

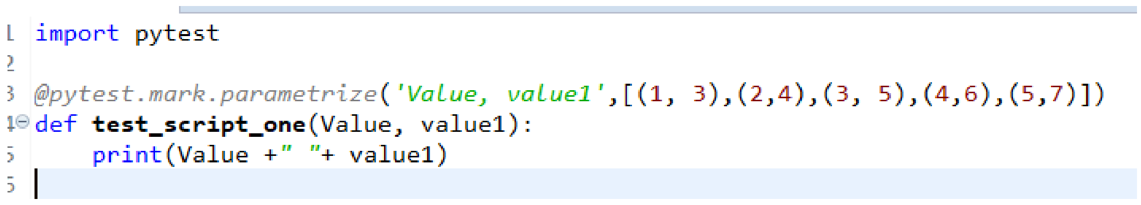


```
D:\Training\PythonSelenium\PyTest_Practice\marker>pytest -v
============================================= test session starts =====
platform win32 -- Python 3.7.4, pytest-5.0.1, py-1.8.0, pluggy-0.12.0 -- c:
python\python37-32\python.exe
cachedir: .pytest_cache
rootdir: D:\Training\PythonSelenium\PyTest_Practice\marker
plugins: allure-pytest-2.7.1, forked-1.0.2, xdist-1.29.0
collected 5 items

test_parametrize.py::test_script_one[1] PASSED
test_parametrize.py::test_script_one[2] PASSED
test_parametrize.py::test_script_one[3] PASSED
test_parametrize.py::test_script_one[4] PASSED
test_parametrize.py::test_script_one[5] PASSED

============================================ 5 passed in 0.11 seconds ==
```

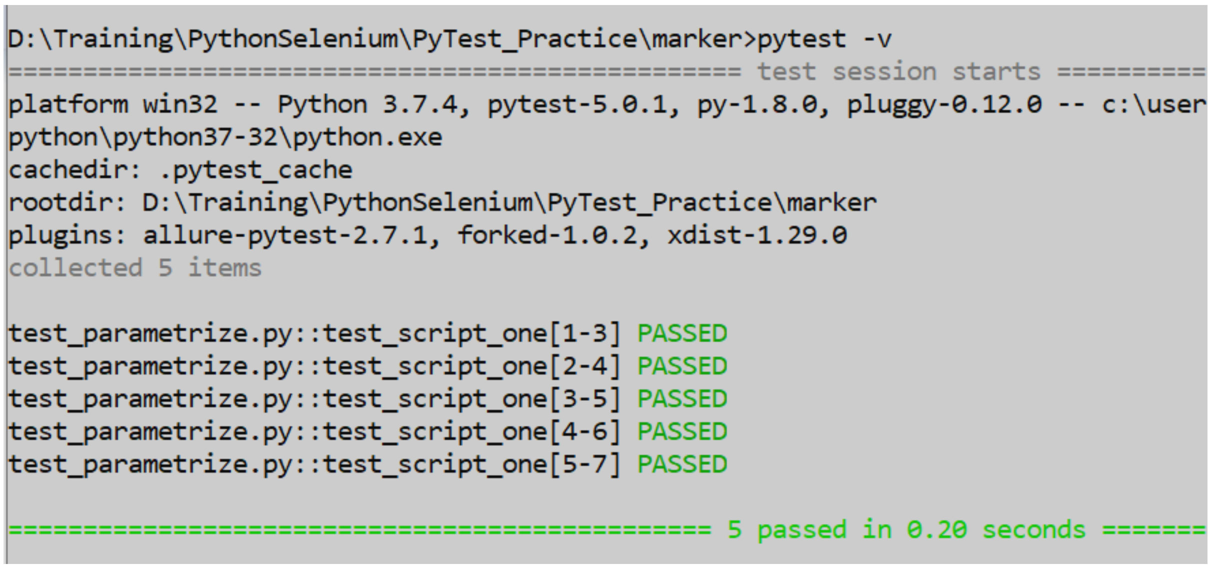
IMP: parameters of test functions should be same as **arg_names**

The above test function will be called 5 times, since we are passing the 5 arg_names.

Program to pass multiple arg_names.



```python
import pytest

@pytest.mark.parametrize('Value, value1',[(1, 3),(2,4),(3, 5),(4,6),(5,7)])
def test_script_one(Value, value1):
    print(Value +" "+ value1)

```



```
D:\Training\PythonSelenium\PyTest_Practice\marker>pytest -v
================================================== test session starts ==========
platform win32 -- Python 3.7.4, pytest-5.0.1, py-1.8.0, pluggy-0.12.0 -- c:\user
python\python37-32\python.exe
cachedir: .pytest_cache
rootdir: D:\Training\PythonSelenium\PyTest_Practice\marker
plugins: allure-pytest-2.7.1, forked-1.0.2, xdist-1.29.0
collected 5 items

test_parametrize.py::test_script_one[1-3] PASSED
test_parametrize.py::test_script_one[2-4] PASSED
test_parametrize.py::test_script_one[3-5] PASSED
test_parametrize.py::test_script_one[4-6] PASSED
test_parametrize.py::test_script_one[5-7] PASSED

================================================ 5 passed in 0.20 seconds =======
```

Use fixture

@pytest.mark.usefixtures(fixturename1, fixturename2…..)

- Mark tests as needing all the specified fixtures.

Try first:

- Mark a hook implementation function, such that the plugin will try to call it first as early as possible.

Try last:

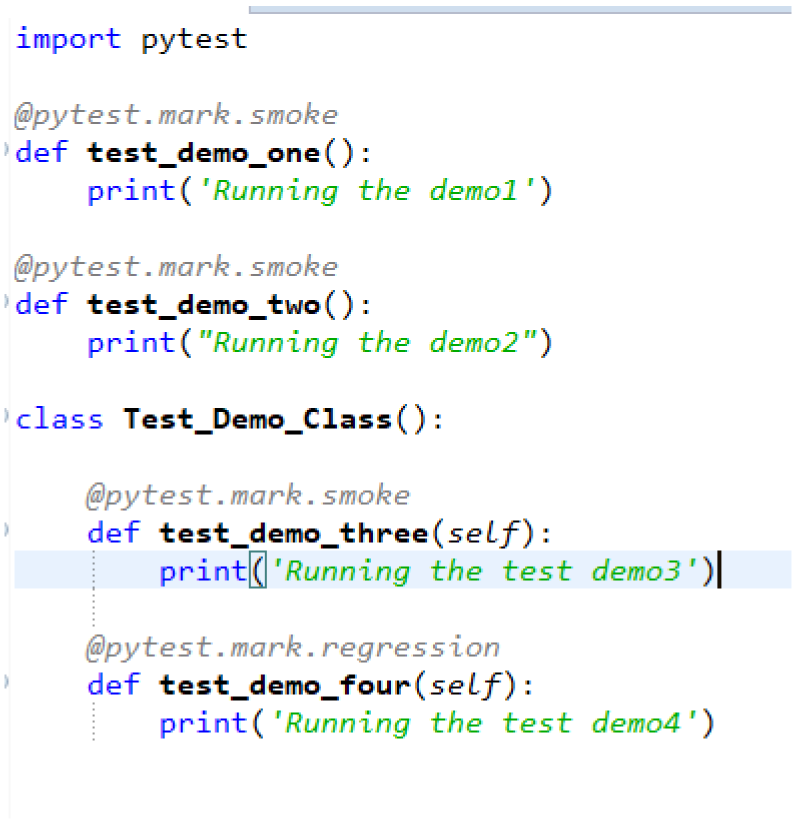
- Mark a hook implementation function such that plugin will try to call it as late as possible.

User defined markers or Custom Markers

PyTest allows to group tests using markers. PyTest.Mark decorator is used to mark a test function with custom metadata like @pytest.mark.smoke.

This is very handy when we want to run only a subset of tests like "smoke tests" to quickly verify if the changes made by the developer not breaking any major functionalities.

You can mark a test function with custom metadata like below:



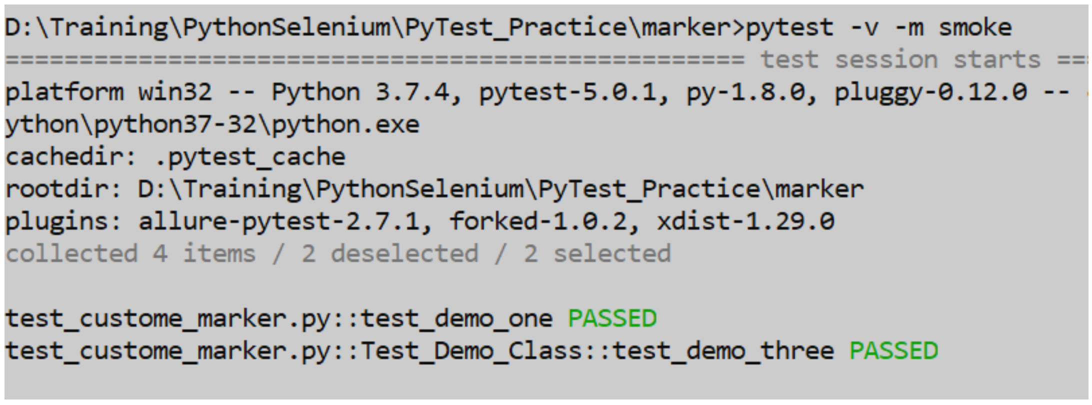
```python
import pytest

@pytest.mark.smoke
def test_demo_one():
    print('Running the demo1')

@pytest.mark.smoke
def test_demo_two():
    print("Running the demo2")

class Test_Demo_Class():

    @pytest.mark.smoke
    def test_demo_three(self):
        print('Running the test demo3')

    @pytest.mark.regression
    def test_demo_four(self):
        print('Running the test demo4')
```

Now, to run the specific marker. For example: running only tests which are marked with smoke.

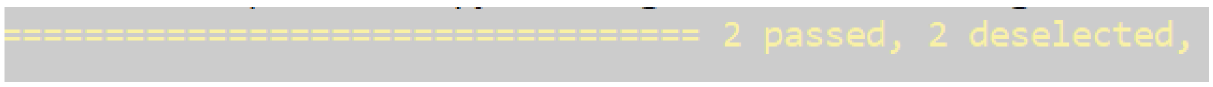
Use the below command:

“ **PyTest -m ‘marker name’** ”

To run only smoke tests:



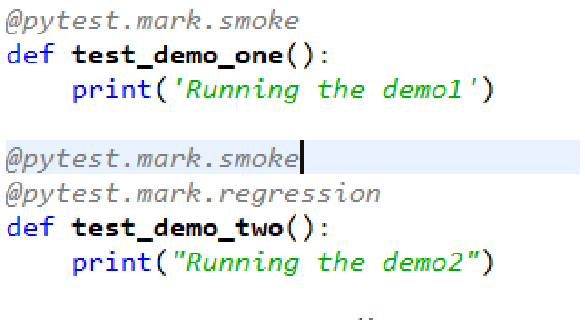
```
D:\Training\PythonSelenium\PyTest_Practice\marker>pytest -v -m smoke
================================================== test session starts ==
platform win32 -- Python 3.7.4, pytest-5.0.1, py-1.8.0, pluggy-0.12.0 --
ython\python37-32\python.exe
cachedir: .pytest_cache
rootdir: D:\Training\PythonSelenium\PyTest_Practice\marker
plugins: allure-pytest-2.7.1, forked-1.0.2, xdist-1.29.0
collected 4 items / 2 deselected / 2 selected

test_custome_marker.py::test_demo_one PASSED
test_custome_marker.py::Test_Demo_Class::test_demo_three PASSED
```



```
=================================== 2 passed, 2 deselected,
```

Two tests were deselected coz, which are not smoke tests.

Note: We can also decorate the tests with multiple marker.

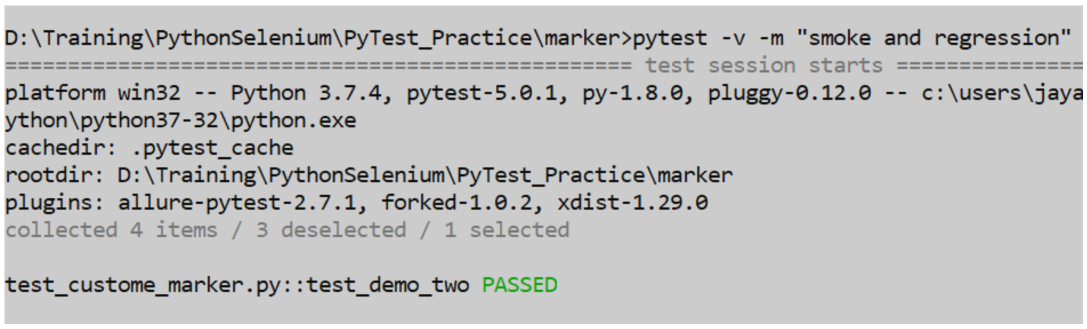


```
@pytest.mark.smoke
def test_demo_one():
    print('Running the demo1')

@pytest.mark.smoke
@pytest.mark.regression
def test_demo_two():
    print("Running the demo2")
```

Now, to run the tests which has multiple marker. Or to run all the tests by ignoring the one group or to run all the tests which either belong to any markers. We use logical operators.

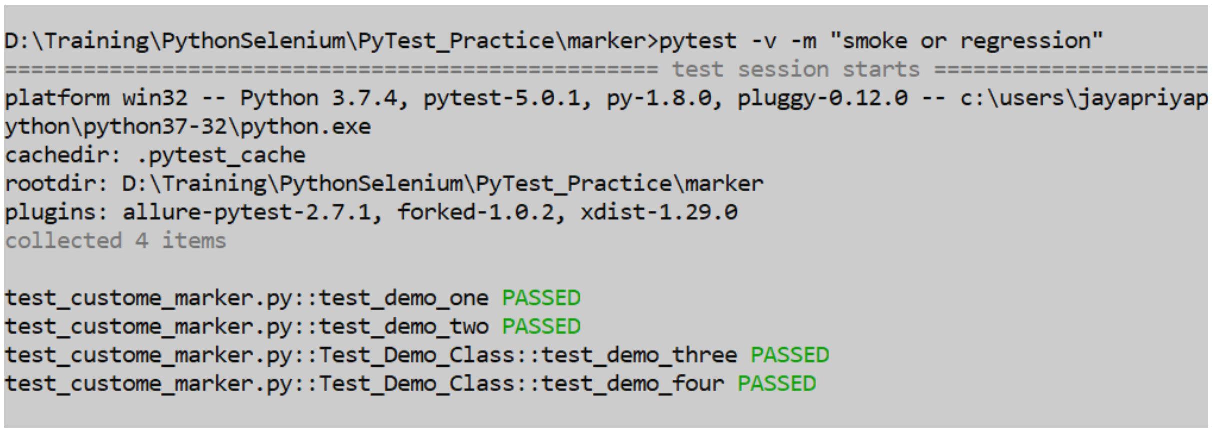
Below are the example programs to us the logical conditions to choose tests to run.

And condition:



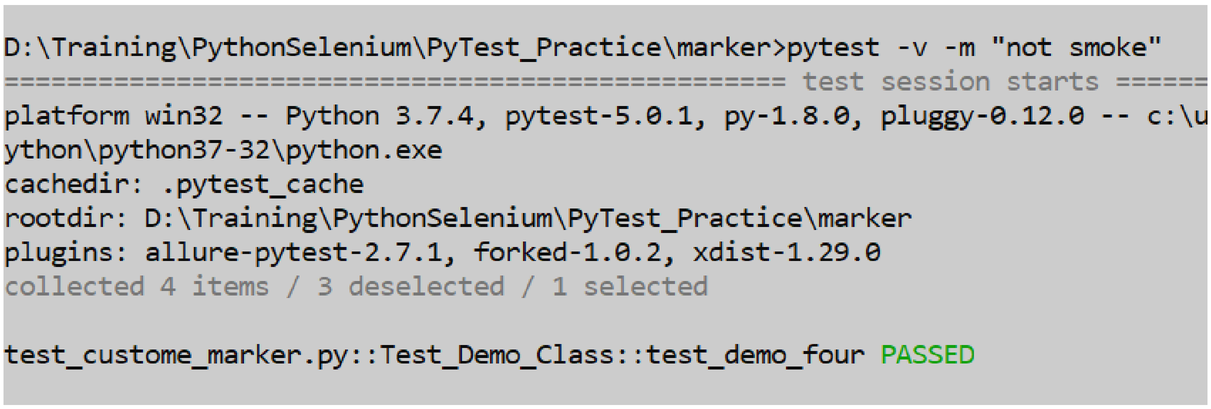
```
D:\Training\PythonSelenium\PyTest_Practice\marker>pytest -v -m "smoke and regression"
================================================== test session starts ===============
platform win32 -- Python 3.7.4, pytest-5.0.1, py-1.8.0, pluggy-0.12.0 -- c:\users\jaya
ython\python37-32\python.exe
cachedir: .pytest_cache
rootdir: D:\Training\PythonSelenium\PyTest_Practice\marker
plugins: allure-pytest-2.7.1, forked-1.0.2, xdist-1.29.0
collected 4 items / 3 deselected / 1 selected

test_custome_marker.py::test_demo_two PASSED
```

Or Condition:



```
D:\Training\PythonSelenium\PyTest_Practice\marker>pytest -v -m "smoke or regression"
==================================================== test session starts ====================
platform win32 -- Python 3.7.4, pytest-5.0.1, py-1.8.0, pluggy-0.12.0 -- c:\users\jayapriyap
ython\python37-32\python.exe
cachedir: .pytest_cache
rootdir: D:\Training\PythonSelenium\PyTest_Practice\marker
plugins: allure-pytest-2.7.1, forked-1.0.2, xdist-1.29.0
collected 4 items

test_custome_marker.py::test_demo_one PASSED
test_custome_marker.py::test_demo_two PASSED
test_custome_marker.py::Test_Demo_Class::test_demo_three PASSED
test_custome_marker.py::Test_Demo_Class::test_demo_four PASSED
```

Not condition:

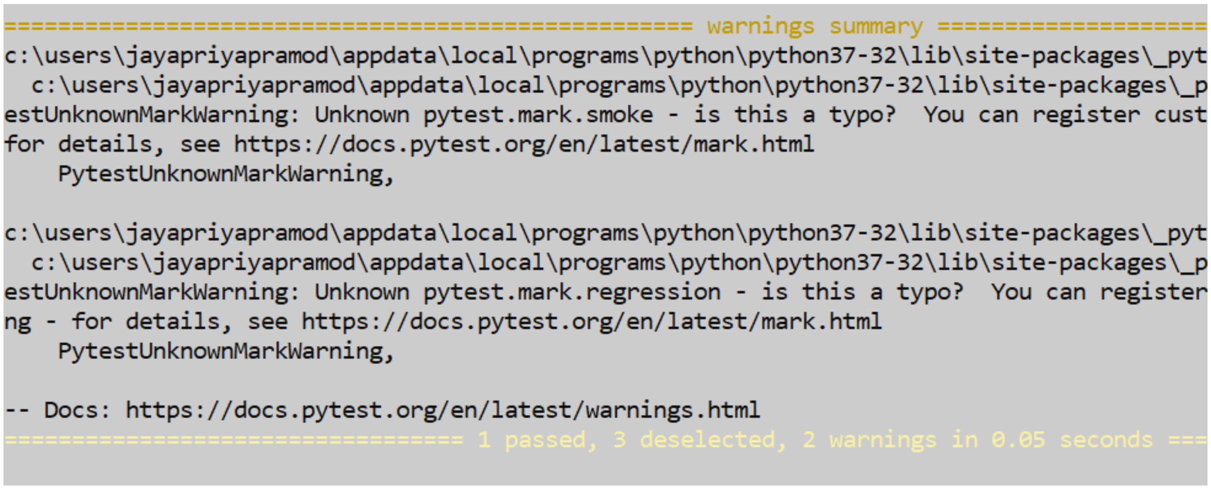


```
D:\Training\PythonSelenium\PyTest_Practice\marker>pytest -v -m "not smoke"
================================================== test session starts ======
platform win32 -- Python 3.7.4, pytest-5.0.1, py-1.8.0, pluggy-0.12.0 -- c:\u
ython\python37-32\python.exe
cachedir: .pytest_cache
rootdir: D:\Training\PythonSelenium\PyTest_Practice\marker
plugins: allure-pytest-2.7.1, forked-1.0.2, xdist-1.29.0
collected 4 items / 3 deselected / 1 selected

test_custome_marker.py::Test_Demo_Class::test_demo_four PASSED
```

Important:

When running the scripts by using the custom markers.

PyTest will through the warning messages as the custom markers are not registered.



```
=================================================== warnings summary ====================
c:\users\jayapriyapramod\appdata\local\programs\python\python37-32\lib\site-packages\_pyt
  c:\users\jayapriyapramod\appdata\local\programs\python\python37-32\lib\site-packages\_p
estUnknownMarkWarning: Unknown pytest.mark.smoke - is this a typo?  You can register cust
for details, see https://docs.pytest.org/en/latest/mark.html
    PytestUnknownMarkWarning,

c:\users\jayapriyapramod\appdata\local\programs\python\python37-32\lib\site-packages\_pyt
  c:\users\jayapriyapramod\appdata\local\programs\python\python37-32\lib\site-packages\_p
estUnknownMarkWarning: Unknown pytest.mark.regression - is this a typo?  You can register
ng - for details, see https://docs.pytest.org/en/latest/mark.html
    PytestUnknownMarkWarning,

-- Docs: https://docs.pytest.org/en/latest/warnings.html
================================= 1 passed, 3 deselected, 2 warnings in 0.05 seconds ===
```

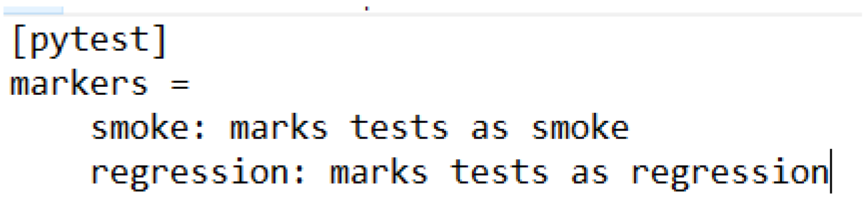
We always need to register the custom markers before decorating on test functions.

Registering marks

You can register custom marks in your **"pytest.ini"** file like this:

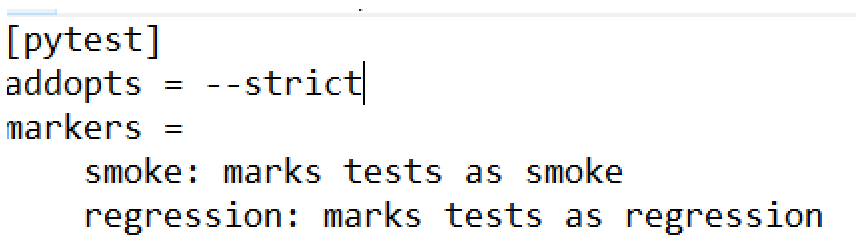
Right click on the project -> new -> file -> and type "pytest.ini".

Write the below code to register the custom marker.



```
[pytest]
markers =
    smoke: marks tests as smoke
    regression: marks tests as regression
```

Above statements will register the marker. And in order to force all the engineers to use the same markers, we need to use the "strict"



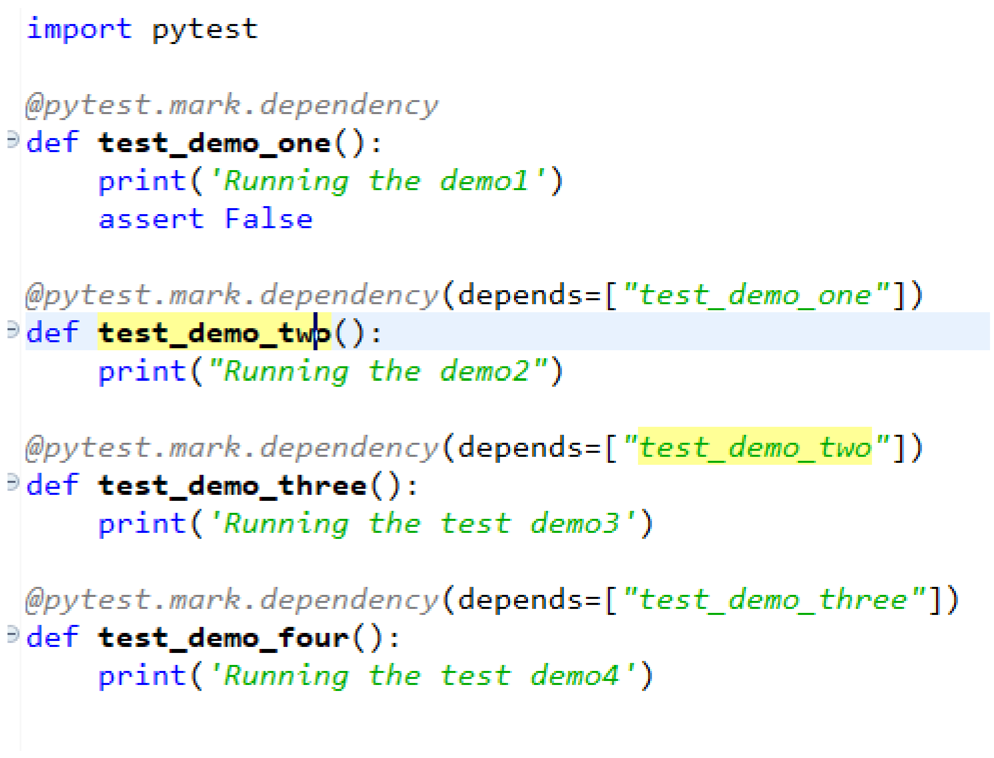
```
[pytest]
addopts = --strict
markers =
    smoke: marks tests as smoke
    regression: marks tests as regression
```

PyTest Dependency:

This module is a plugin for PyTest. It manages dependencies of tests. You may mark some tests as dependent from other tests. These tests will then be skipped if any of the dependencies did fail or has been skipped.

Installation

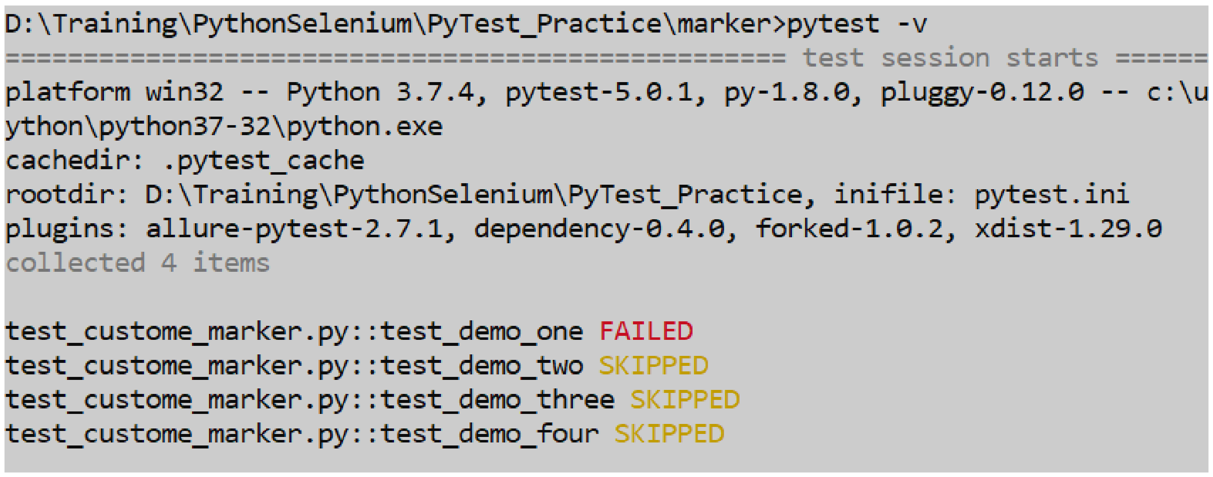
```
pip install pytest-dependency
```



```
import pytest

@pytest.mark.dependency
def test_demo_one():
    print('Running the demo1')
    assert False

@pytest.mark.dependency(depends=["test_demo_one"])
def test_demo_two():
    print("Running the demo2")

@pytest.mark.dependency(depends=["test_demo_two"])
def test_demo_three():
    print('Running the test demo3')

@pytest.mark.dependency(depends=["test_demo_three"])
def test_demo_four():
    print('Running the test demo4')
```

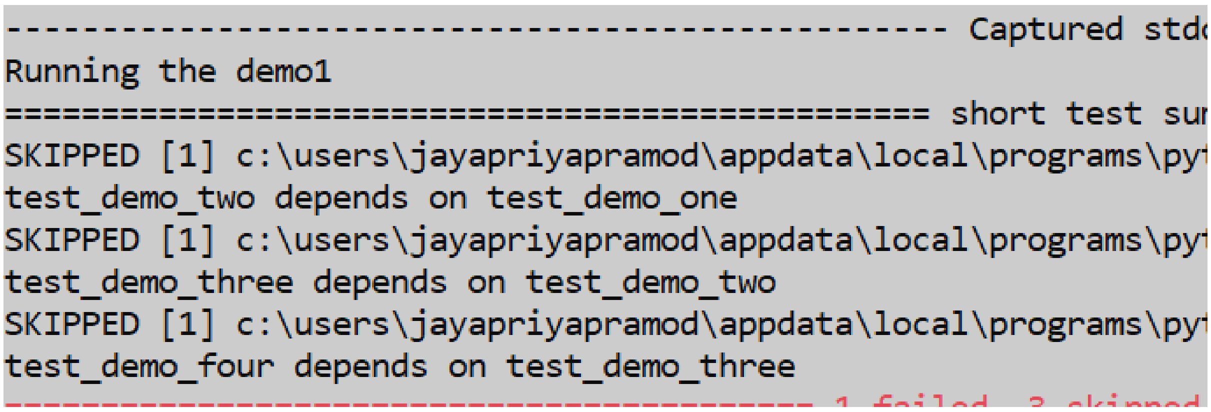
In the above program test_one is the independent test, test two, three & four are depending on each other.

If test one executes without any error than test two will also executes, if test one fails than test two will be skipped since test two is depending on the test one.



```
D:\Training\PythonSelenium\PyTest_Practice\marker>pytest -v
================================================== test session starts ======
platform win32 -- Python 3.7.4, pytest-5.0.1, py-1.8.0, pluggy-0.12.0 -- c:\u
ython\python37-32\python.exe
cachedir: .pytest_cache
rootdir: D:\Training\PythonSelenium\PyTest_Practice, inifile: pytest.ini
plugins: allure-pytest-2.7.1, dependency-0.4.0, forked-1.0.2, xdist-1.29.0
collected 4 items

test_custome_marker.py::test_demo_one FAILED
test_custome_marker.py::test_demo_two SKIPPED
test_custome_marker.py::test_demo_three SKIPPED
test_custome_marker.py::test_demo_four SKIPPED
```

To check the dependency between the test use “-rsx” command.



```
------------------------------------------------- Captured std
Running the demo1
================================================ short test su
SKIPPED [1] c:\users\jayapriyapramod\appdata\local\programs\py
test_demo_two depends on test_demo_one
SKIPPED [1] c:\users\jayapriyapramod\appdata\local\programs\py
test_demo_three depends on test_demo_two
SKIPPED [1] c:\users\jayapriyapramod\appdata\local\programs\py
test_demo_four depends on test_demo_three
------------------------------------------ 1 failed, 3 skipped
```

Pytest-Ordering: run test in order

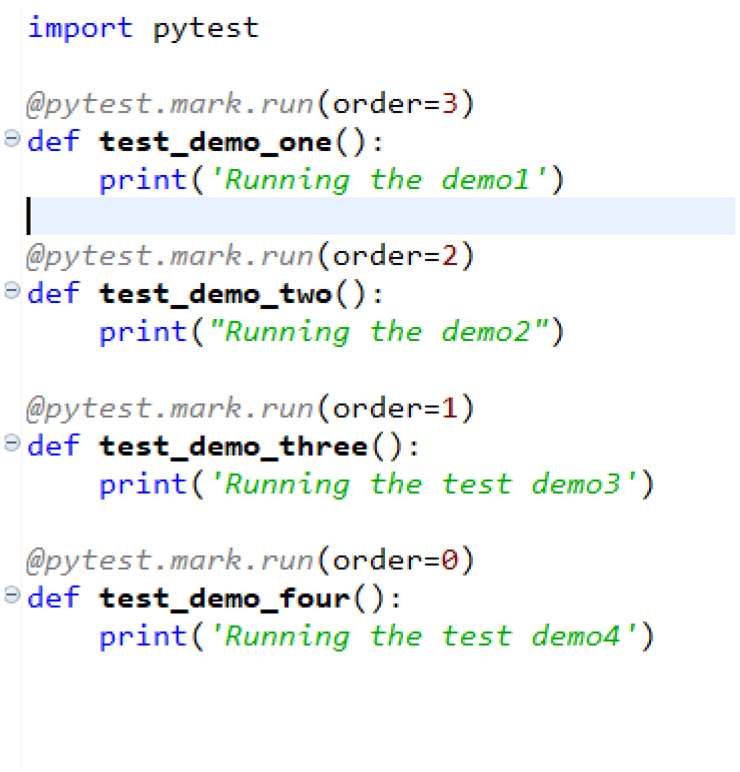
PyTest-ordering is a pytest plugin to run your tests in any order that you specify. It provides custom markers that say when your tests should run in relation to each other.

Installation:

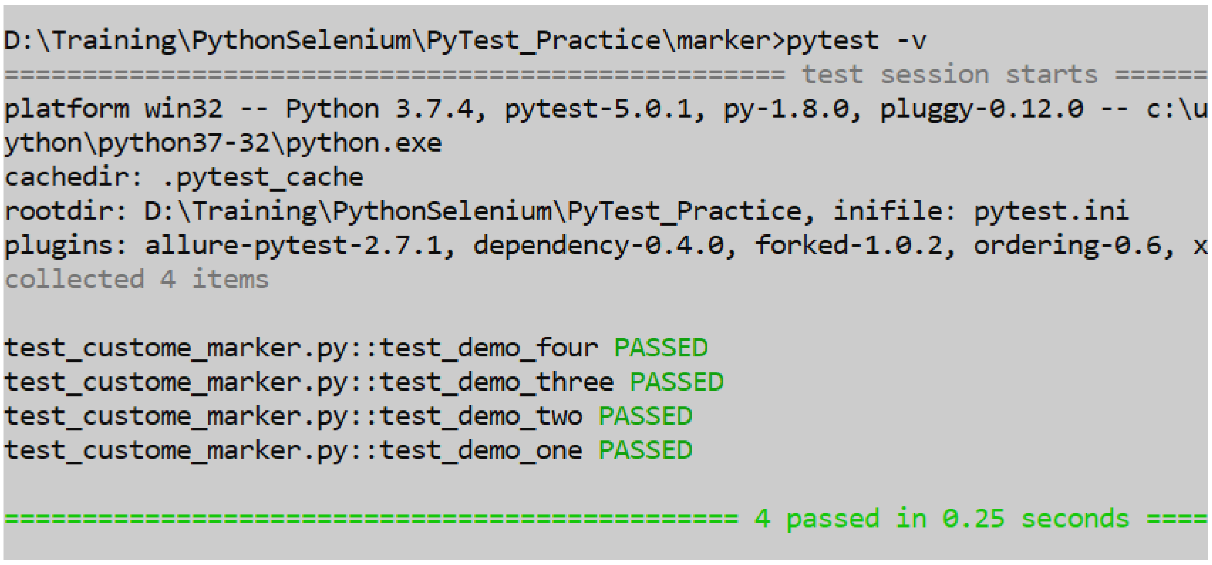
```
pip install pytest-ordering
```

Note:

- Ordinarily pytest will run tests in the order that they appear in a module
- Lowest the number highest the priority.



```python
import pytest

@pytest.mark.run(order=3)
def test_demo_one():
    print('Running the demo1')

@pytest.mark.run(order=2)
def test_demo_two():
    print("Running the demo2")

@pytest.mark.run(order=1)
def test_demo_three():
    print('Running the test demo3')

@pytest.mark.run(order=0)
def test_demo_four():
    print('Running the test demo4')
```


```
D:\Training\PythonSelenium\PyTest_Practice\marker>pytest -v
==================================================== test session starts ======
platform win32 -- Python 3.7.4, pytest-5.0.1, py-1.8.0, pluggy-0.12.0 -- c:\u
ython\python37-32\python.exe
cachedir: .pytest_cache
rootdir: D:\Training\PythonSelenium\PyTest_Practice, inifile: pytest.ini
plugins: allure-pytest-2.7.1, dependency-0.4.0, forked-1.0.2, ordering-0.6, x
collected 4 items

test_custome_marker.py::test_demo_four PASSED
test_custome_marker.py::test_demo_three PASSED
test_custome_marker.py::test_demo_two PASSED
test_custome_marker.py::test_demo_one PASSED

=============================================== 4 passed in 0.25 seconds ====
```

Automatically adding markers based on test names

If you a test suite where test function names indicate a certain type of test, you can implement a hook that automatically defines markers so that you can use the `-m` option with it. Let's look at this test module:

```
# content of test_module.py


def test_interface_simple():
    assert 0


def test_interface_complex():
    assert 0


def test_event_simple():
    assert 0


def test_something_else():
    assert 0
```

We want to dynamically define two markers and can do it in a `conftest.py` plugin:

```
# content of conftest.py

import pytest


def pytest_collection_modifyitems(items):
    for item in items:
        if "interface" in item.nodeid:
            item.add_marker(pytest.mark.interface)
        elif "event" in item.nodeid:
            item.add_marker(pytest.mark.event)
```

We can now use the `-m option` to select one set:

```
$ pytest -m interface --tb=short
=========================== test session starts
============================
platform linux -- Python 3.x.y, pytest-5.x.y, py-1.x.y, pluggy-0.x.y
cachedir: $PYTHON_PREFIX/.pytest_cache
rootdir: $REGENDOC_TMPDIR
collected 4 items / 2 deselected / 2 selected

test_module.py FF
[100%]

================================= FAILURES
=================================
__________________________ test_interface_simple
___________________________
test_module.py:4: in test_interface_simple
    assert 0
E   assert 0
__________________________ test_interface_complex
__________________________
test_module.py:8: in test_interface_complex
    assert 0
E   assert 0
===================== 2 failed, 2 deselected in 0.12s
======================
```

or to select both “event” and “interface” tests:

```
$ pytest -m "interface or event" --tb=short
```

```
=========================== test session starts
============================
platform linux -- Python 3.x.y, pytest-5.x.y, py-1.x.y, pluggy-0.x.y
cachedir: $PYTHON_PREFIX/.pytest_cache
rootdir: $REGENDOC_TMPDIR
collected 4 items / 1 deselected / 3 selected

test_module.py FFF
[100%]

================================= FAILURES
=================================
__________________________ test_interface_simple
___________________________
test_module.py:4: in test_interface_simple
    assert 0
E   assert 0
__________________________ test_interface_complex
__________________________
test_module.py:8: in test_interface_complex
    assert 0
E   assert 0
____________________________ test_event_simple
_____________________________
test_module.py:12: in test_event_simple
    assert 0
E   assert 0
```