```
pip install pytest
```

Running pytest without mentioning a filename will run all files of format **test_*.py** or ***_test.py** in the current directory and subdirectories. Pytest automatically identifies those files as test files. We **can** make pytest run other filenames by explicitly mentioning them

```
import math

def test_sqrt():
   num = 25
   assert math.sqrt(num) == 5

def testsquare():
   num = 7
   assert 7*7 == 40

def tesequality():
   assert 10 == 11
```

Run tests in a module

```
pytest test_mod.py
```

Run tests in a directory

```
pytest testing/
```

```
pytest test_compare.py -v      # increases the verbosity
```

To execute the tests containing a string in its name

```
pytest -k great -v
```

This will execute all the test names having the word **'great'** in its name. In this case, they are **test_greater()** and **test_greater_equal()**

Pytest allows us to use markers on test functions. Markers are used to set various features/attributes to test functions. Pytest provides many inbuilt markers such as xfail, skip and parametrize. Apart from that, users can create their own marker names. Markers are applied on the tests using the syntax given below −

```
@pytest.mark.<markername>
```

test_compare.py

```
import pytest
@pytest.mark.great
def test_greater():
   num = 100
   assert num > 100

@pytest.mark.great
def test_greater_equal():
   num = 100
   assert num >= 100

@pytest.mark.smoketestcase
def test_less():
   num = 100
   assert num < 200
```

test_square.py

```
import pytest
import math

@pytest.mark.square
def test_sqrt():
   num = 25
   assert math.sqrt(num) == 5

@pytest.mark.square
def testsquare():
   num = 7
   assert 7*7 == 40

@pytest.mark.smoketestcase
   def test_equality():
   assert 10 == 11
```

Now to run the tests marked as **others**, run the following command −

```
pytest -m smoketestcase -v
```

it will run test case marks with name smoketestcase in the entire folder

Calling pytest through python

```
\test_myPytest>python -m pytest test_compare.py -v
```

Run a single test from single class file

```
pytest test_square.py::test_sqrt
```

Run a single test from Multiple class file

```
pytest test_mod.py::TestClass::test_method
```

Detailed summary report

The -r flag can be used to display a “short test summary info” at the end of the test session, making it easy in large test suites to get a clear picture of all failures, skips, xfails, etc.

```
Pytest -ra test_example.py

test_example.py:14: AssertionError
========================== short test summary info ==========================
SKIPPED [1] $REGENDOC_TMPDIR/test_example.py:22: skipping this test
XFAIL test_example.py::test_xfail
  reason: xfailing this test
XPASS test_example.py::test_xpass always xfail
ERROR test_example.py::test_error - assert 0
FAILED test_example.py::test_fail - assert 0
== 1 failed, 1 passed, 1 skipped, 1 xfailed, 1 xpassed, 1 error in 0.12s ===
```

Fixtures are functions, which will run before each test function to which it is applied. Fixtures are used to feed some data to the tests such as database connections

```
import pytest

@pytest.fixture
def input_value():
   input = 39
   return input

def test_divisible_by_3(input_value):
   assert input_value % 3 == 0

def test_divisible_by_6(input_value):
   assert input_value % 6 == 0
```

We can define the fixture functions in this file to make them accessible across multiple test files.

Create a new file **conftest.py**

```
import pytest

@pytest.fixture
def input_value():
   input = 39
   return input
```

Edit the **test_div_by_3_6.py** to remove the fixture function −

```
import pytest

def test_divisible_by_3(input_value):
```

```
    assert input_value % 3 == 0

def test_divisible_by_6(input_value):
    assert input_value % 6 == 0
```

Create a new file **test_div_by_13.py** −

```
import pytest

def test_divisible_by_13(input_value):
    assert input_value % 13 == 0
```

Now, we have the files **test_div_by_3_6.py** and **test_div_by_13.py** making use of the fixture defined in **conftest.py**. divisible is keyword to run

Run the tests by executing the following command −

```
pytest -k divisible -v
```

Parameterizing of a test is done to run the test against multiple sets of inputs. We can do this by using the following marker −

```
@pytest.mark.parametrize
```

Copy the below code into a file called **test_multiplication.py** −

```
import pytest

@pytest.mark.parametrize("num, output",[(1,11),(2,22),(3,35),(4,44)])
def test_multiplication_11(num, output):
    assert 11*num == output
```

Here the test multiplies an input with 11 and compares the result with the expected output. The test has 4 sets of inputs, each has 2 values – one is the number to be multiplied with 11 and the other is the expected result.

Execute the test by running the following command −

```
Pytest -k multiplication -v
```

SKIP the test cases

1. *@pytest.mark.skip*
2. Skipif
3. Pytest.skip

```
@pytest.mark.skip
def test_less():
    print("skip")

def test_function():
    if not valid_config():
        pytest.skip("unsupported configuration")

import sys
import pytest
```

```
# It is also possible to skip the whole module using allow_module_level=True
if not sys.platform.startswith("lin"):
    pytest.skip("skipping windows-only tests", allow_module_level=True)


@pytest.mark.skipif(sys.version_info < (3, 6), reason="requires python3.6 or higher")
def test_function():
    pass
```

xfail marker to indicate that you expect a test to fail. This test will of course fail until you fix the bug. To avoid having a failing test you mark the test as xfail. Once the bug is fixed you remove the xfail marker and have a regression test which ensures that the bug will not reoccur.

```
@pytest.mark.xfail
def test_greater():
    num = 100
    assert num > 100

@pytest.mark.xfail
def test_greater_equal():
    num = 100
    assert num >= 100
```

Pytest - Stop Test Suite after N Test Failures

```
pytest --maxfail = <num>


pytest test_failure.py -v --maxfail = 1
```

Pytest - Run Tests in Parallel

Install pytest-xdist by running the following command −

```
pip install pytest-xdist
pytest -n 3
```

Test Execution Results in XML Format

```
pytest test_multiplication.py -v --junitxml="result.xml"
```

Custom fixture to each file

```
# content of conftest.py

import os
import shutil
import tempfile

import pytest
```

```
@pytest.fixture
def cleandir():
    old_cwd = os.getcwd()
    newpath = tempfile.mkdtemp()
    os.chdir(newpath)
    yield
    os.chdir(old_cwd)
    shutil.rmtree(newpath)
# content of test_setenv.py
import os
import pytest


@pytest.mark.usefixtures("cleandir")
class TestDirectoryInit:
    def test_cwd_starts_empty(self):
        assert os.listdir(os.getcwd()) == []
        with open("myfile", "w") as f:
            f.write("hello")

    def test_cwd_again_starts_empty(self):
        assert os.listdir(os.getcwd()) == []
```

You can specify multiple fixtures like this:

```
@pytest.mark.usefixtures("cleandir", "anotherfixture")
def test():
```

pytest-ordering: run your tests in order

```
@pytest.mark.run(order=-2)
@pytest.mark.run(order=1)
@pytest.mark.order2
@pytest.mark.order1
@pytest.mark.second_to_last
@pytest.mark.last
@pytest.mark.second
@pytest.mark.first
@pytest.mark.run('second-to-last')
@pytest.mark.run('last')
@pytest.mark.run('second')
@pytest.mark.run('first')

@pytest.mark.run(after='test_second')
def test_third():
    assert True

def test_second():
    assert True

@pytest.mark.run(before='test_second')
def test_first():
    assert True
```

Creating resultlog format files

To create plain-text machine-readable result files you can issue:

```
pytest --resultlog=path
```

How can I repeat each test multiple times in a py.test run?

`pip install pytest-repeat`

```
$ py.test --count=10 test_file.py
```

Each test collected by py.test will be run `count` times.

If you want to mark a test in your code to be repeated a number of times, you can use the `@pytest.mark.repeat(count)` decorator:

```
import pytest

@pytest.mark.repeat(3)

def test_repeat_decorator():

    pass
```

pytest Report Generation

`pip install pytest-html`

`pytest --html=pytest_selenium_test_report.html`