

Installation

```
pip install pytest
```

Running pytest

<code>pytest test_mod.py</code>	Run tests in module
<code>pytest testing/</code>	Run multiple modules
<code>pytest test_mod.py::test_func</code>	Run a specific test within a module.
<code>pytest test_mod.py::TestClass::test_method</code>	Run a specific method of a class.

Assertions

pytest allows you to use the standard Python assert for verifying expectations and values in Python tests. For example, you can write the following:

<code>assert a % 2 == 0</code>	assert that you have a certain value.
<code>assert a % 2 == 0, "value was odd, should be even"</code>	specify a message with the assertion like this

Grouping in tests

<code>@pytest.mark.<markername></code>	mark it in tests
<code>pytest tests/ -m "sanitytests"</code>	run using -m flag

Disable tests

```
@pytest.mark.skip(
    reason="no way of currently testing this")
```

Assertion of expected exceptions

Test if the exception is raised

```
def test_zero_division():
    with pytest.raises(ZeroDivisionError):
        1 / 0
```

Test for specific exception info

```
def test_recursion_depth():
    with pytest.raises(RuntimeError) as excinfo:
        assert "maximu m" in str(excinfo.value)
```

MultiThreading

<code>pip install pytest-xdist</code>	Pytest does not run in parallel by default, therefor we need to enable this behaviour through a plugin.
<code>pytest tests/ -n 3</code>	

Fixtures

In testing, a fixture provides a defined, reliable and consistent context for the tests. This could include environment (for example a database configured with known parameters) or content (such as a dataset).

<code>@pytest.fixture</code>	Create fixture in confest
<code>def input_value():</code>	
<code> input = 10</code>	
<code> return input</code>	

<code>def test_divisible_by_3(input_value):</code>	Use it in tests
<code> assert input_value % 3 == 0</code>	

<code>@pytest.fixture</code>	using yield to setup and teardown test environment
<code>def sending_user(mail_admin):</code>	
<code> user = mail_admin.create_user()</code>	
<code> yield user</code>	
<code> mail_admin.delete_user(user)</code>	

<code>@pytest.fixture(scope="function")</code>	Pytest fixtures have scope. This scope controls how often the fixture is executed or, in other words, how often the setup and teardown of the fixture is performed.
--	---

`@pytest.fixture(scope="module")`

`@pytest.fixture(scope="class")`

`@pytest.fixture(scope="session")`

Data parameterization

The builtin `pytest.mark.parametrize` decorator enables parametrization of arguments for a test function

Define parametrized tests

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

Running specific tests of a parametrized test.

`test_mod.py::test_multiplication_11[2-22]`