PYTEST

Definition- Pytest is a python-based testing Framework which is used to write and execute the codes

Advantages:

* Very easy to start with because of its simple and easy syntax.
* Can run tests in parallel.
* Can run a specific test or a subset of tests.
* Dependency test.
* Grouping of test methods.
* Generates report.

Execution:

* pytest modulename
* pytest

Execution test methods:

* pytest packagename/module\_name.py:testmethod
* pytest
* pytest package name/modulename
* pytest module\_name::classname::methodname

Following are the conventions to be followed

* The module name should either start with test\_\*
* All the classes inside the module should start from Test\*
* All the test methods should start from test\_\*.

Eg: save the file as library.py

class TestUtility:

def test\_even(self):

assert is\_even(10) == True

def test\_odd(self):

assert is\_even(9) == False

* another file as test\_library.py

class TestUtility:

def test\_even(self):

assert is\_even(10) == True

def test\_odd(self):

assert is\_even(9) == False

Pytest dependency:

Install pytest- dependency

Definition: in order to make one test dependent on test result of another test method we use python dependency

* Both the test are decorated with “@pytest.mark.dependency()”

Eg: class TestLogin:

@pytest.mark.dependency()

def test\_login(self):

print('Running test\_login')

assert False

@pytest.mark.dependency(depends=["TestLogin::test\_login"])

def test\_logout(self):

print('Running test\_logout')

Pytest Ordering:

* To execute the test methods in specified order
* Install pytest- ordering
* @pytest.mark.run(order=value)

Eg:

class TestLibrary:

@pytest.mark.run(order=2)

def test\_even(self):

print('Running test\_even')

assert is\_even(10) == True

@pytest.mark.run(order=1)

def test\_odd(self):

print('Running test\_odd')

assert is\_even(9) == False

Pytest Markers:

Its is a decorator used to add metadata to the test

There are two types:

* inbuilt markers
* customized markers
* Inbuilt markers:
* @pytest.mark.skip(reason= “ “)
* @pytest.mark.skipif(condition= “ “)
* @pytest.mark.parameterize(arg\_name, arg\_value)
* @pytest.mark.usefixtures(name)
* Customized markers
* @pytest.mark.customized\_markername
* **pytest module\_name -vs -m name of the marker**

eg: class TestUtility:

@pytest.mark.smoke

def test\_even(self):

assert is\_even(10) == True

def test\_odd(self):

assert is\_even(9) == False

Generation of report:

1)HTML report generation :

* install pytest-html:
* pytest –html= reports.html

Pytestfixtures:

Definition: functions which run before each test functions to which they are specified

Syntax: @pytest.fixtures(attributes)

@pytest.mark.usefixture(name)

* attributes:
* scope: 1) sets the scope of fixtures
* by default function
* four types: function, class,module,session
* autouse:
* if true fixture is applied to all test
* if false fixture needs to be explicitly called
* params:
* takes list of parameters which are used to run fixture for multiple times

eg class TestUtility:

def test\_even(self, init):

assert is\_even(10) == True

def test\_odd(self, init):

assert is\_even(9) == False

class TestUtility:

def test\_even(self, init):

assert is\_even(10) == True

def test\_odd(self, init):

assert is\_even(9) == False

@pytest.mark.usefixtures("init")

class TestUtility:

def test\_even(self):

assert is\_even(10) == True

def test\_odd(self):

assert is\_even(9) == False