**PYTEST TESTING FRAMEWORK**

1. **Pytest:**

* Pytest is python based testing Framework. Which is used to write and execute test codes.
* Pytest mainly used for API testing even though we can use pytest to write simple to complex testcases i.e we can write code to test API, database,UI etc.
* The Pytest framework makes it easy to write small, readable tests, and can scale to support complex functional testing for applications and libraries.

**2.Advantages of Pytest:**

* Pytest can run multiple test in parellel, which reducess the execution time of the test suite.
* Pytest has its own way to detect the test file and test functions automatically, if not mentioned explicitly.
* Pytest allows us to skip a subset of the tests during execution.
* Pytest allows us to run a subset of the entire test suite.

**3.Naming Conventions & Test Discovery:**

* File names should start or end with “test” , as in **test\_example.py** or **example\_test.py.**
* Test method names or function names should start with “test\_” , as in **test\_example.**
* If tests are defined as methods on class name should be startwith “ Test”, as in **TestExample.** The class should not have \_\_init\_\_ method.
* For running the testcases cmd : **Pytest[Filename/options] ,Pytest[Filename/options] -v**
* Calling **Pytest –collects-only** is useful way to see what tests pyetst will discover without running the tests. It will gives the how many test cases have in your file.

**4.Various Ways of running Tests in Pytest:**

* **Finding all the possible tests:**

**pytest –-collect-only**

* **Running all the tests:**

**pytest**

* **Running all the tests with verbous and print statement:**

**pytest -vs**

* **Running the tests including print statment:**

**pytest -s test\_demo.py**

* **Running Specific module:**

**pytest -vs test\_pytest\_demo.py**

* **Running specific method under module:**

**pytest -vs test\_pytest\_demo.py :: method\_name(test\_addition)**

* **Running Specific method under class inside module:**

**pytest -vs test\_pytest\_demo.py :: class\_name :: method\_name(test\_addition)**

* **Run Set of tests which has specific keyword method name:**

**pytest -vs -k keyword\_name**

* **Run Set of tests which comes under specific Marking(custom- @pytest.mark.regression):**

**pytest -vs -m regression**

* **Skipping the test in execution(test has skip marking- @ pytest.mark.skip):**

**pytest -vs test\_pytest\_run\_demo.py**

* **Stop test After N test failure (if maxfail test occure to stop the execution we use maxfail cmd):**

**pytest -vs filename –maxfail (num)**

(**num**: here we can give number i.e 1,2,3 for after how many test cases fail then you want to stop the execution.)

**5.Assert Statement:**

* pytest aseertations are **Chesks that the return either True or False Status.**
* In pytest , if an asseration fails in a test method , then that method execution is stopped there .
* The remaining code in that test method is not executed , and Pytest asseration will coontinue with the next test method.

**6.Markers in Pytest:**

* Pytest allows us to use markers on test functions .
* Markers are used to set of various fixture to test function.
* Pytest provide many inbuilt markers such as **xfail**, **skip**, and **parametrize**.
* Aprat from that , user can create their own marker names.
* Markers are applied on the test using the syntax given below:

**@pytest.mark.<markername>**

* To use marker we have to first import pytest module in test file. To run the marked tests , we can use the following syntax

**pytest – m <markername> -v**

where -m <markername> represent the marker name of the test to be executed

**7. Pytest - Fixture:**

* Fixtures are functions, which will run before each method/class/module test function to which it is applied.
* Fixtures will be used to execute tear down activities after each method/class/module(using yeild)
* Fixtures are used to feed some data to the tests such as database connections, URLs to test and some sort of input data.
* Therefore, instead of running the same code for every test, we can attach fixture function to the tests and it will run and return the data to the test before executing each test.
* A function is marked as a fixture by −

**@pytest.Fixture**

* Fixture can be Centralized via conftest.py.

### **Fixture scopes:**

Fixtures are created when first requested by a test, and are destroyed based on their **scope**:

* **Functions**: The default scope, the fixture is destroyed at the end of the test.
* **Class** : The fixture is destroyed during teardown of the last test in the class.
* **Module**: The fixture is destroyed during teardown of the last test in the module.
* **Packages**: the fixture is destroyed during teardown of the last test in the package.
* **Session** : The fixture is destroyed at the end of the test session.