1. Pytest file should start with test\_ or end with \_test
2. Pytest method should start with test
3. To run all the scripts -> navigate into particular part and give py.test in comment prompt
4. Py.test – v (verbose) , py.test –v –s ( all the console logs)
5. Py.test **test\_login1** –v –s -----🡪 To run specific files
6. To run some specific testcase -> 1 ( same case) used for multiple files at the time we can use below comment -> py.test –k CreditCard –v –s ( here k stands for regular expression, s -> log ouput, s -> metadata of the file and CC is name used in file)
7. To run particular smoke test case, at the time use @pytest.mark.smoke, In comment prompt we have to use -> py.test –m (mark) smoke –v –s
8. To skip particular test -> **@pytest.mark.skip**
9. @Pytest.mark.xfail -> it is running but not shown in report
10. **Fixtures** -> prerequisite before testing ( base concepts) -> **@pytest.fixture() and** also help to load the data
11. **Yield -> post execution script** -> after yield keyword, -> finally it has run
12. Fixture are used to setup and tear down method
13. **Conftest ->** file used to generalize the common testcase
14. **When declare the class -> self is mandatory key word to pass**
15. **@pytest.mark.usefixtures("setup") – Used for pass setup method from conftest file**
16. **Conftest file -> @pytest.fixture(scope="class") -> when we are using “scope = “class” (executed only in before the test and after the test)**
17. **Fixture** – helps to load the data using return method in conftest file
18. **W**hen we are returning value from the conftest fixture file, we need to pass the fixture method name in (dataLoad) method

**def test\_editProfile(self, dataLoad):**

**print(dataLoad)**

1. If we want to print data individually which is in conftest fixture file use below method

**def test\_editProfile(self, dataLoad):**

**print(dataLoad[0])**

**print(dataLoad[1])**

1. When we send the parameter in fixture like **@pytest.fixture(params=["chrome", "IE"])**

**We need to pass the instance called “request” in method like below**

**@pytest.fixture(params=["chrome", "IE"])**

**def crossBrowser(request):**

**the return value should be as below @pytest.fixture(params=["chrome", "IE"])**

**def crossBrowser(request):**

**return request.param**

**What is framework?**

1. organized way of maintaining automation files,
2. in the framework all the files will communicate each other to perform certain task

**Objective/Goals**

1. Re-usability ( same component used wherever is required – Utility files (Excel)
2. Maintainability (add new testcase in framework and maintain with existing tc)

**Type of framework**

1. Built-in-framework (pytest, robotframework, unittest etc) -Predefined
2. Customized/user defined framework ( data driven, keyword driven, hybrid driven framework)

Data driven -> Excel based data driver, we can read and pass the data through the excel

Data driven is part of keyword and hybrid because without data driven (data).

Key word driver -> using keywords to call the data

Hybrid -> combination of both data and keyword driven

**Phases:**

1. **Analysis**

analysis the application,

technology and skill set of the team

**choose the test case** – to select what are the testcase want to automate ( Overcoming retest and regression)

1st select re-testcase (data driven testcase)

2nd -> regression testing -> execute same testcase multiple time

TC’s can be automatable

100 % automation is not possible ( reports, screen capture, security related testcase )

1. Design and implementation of framework
2. Execution ( run the tc with pytest or IDE)
3. Maintenance ( version control system) -> Tools like SVN, GIT

**Ecommerce Application…..**

1. Frontend -> <https://demo.nopcommerce.com/>
2. Back end -> <https://admin-demo.nopcommerce.com/>

**Back end ->** [**https://admin-demo.nopcommerce.com/**](https://admin-demo.nopcommerce.com/)

**Selenium Hybrid Framework**

**(Python, Selenium, PyTest, Page Object Model and HTML Reports)**

**Step 1: Create new project and Install required package and plugins**

* **Selenium: Selenium Libraries**
* **Pytest: Python Unit Test framework**
* **Pytest – html : PyTest HTML Reports**
* **Pytest – xdist : Run Test parallel**
* **Openpyxl: MS Excel Support**
* **Allure-pytest: To generate allure reports**

**Step 2: Create Folder Structure**

**Project Name**

|

Page Object (package)

|

Testcase (package)

|

Utilities (package)

|

Test Data (Folder)

|

Configuration (Folder)

|

Logs (Folder)

|

Screenshot (Folder)

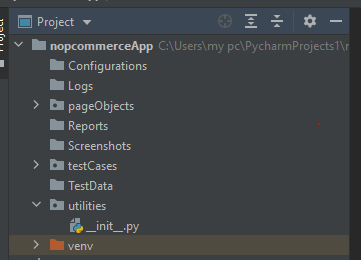
|

Reports (Folder)

|

Run.bat

**Folder Structure**



**Step3: Automating login Testcase**

3.1: Create Login page object class under “Page Objects”

3.2: Create LoginTest under “testcase”

3.3: Create conftest.py under “testcases”

**Step4: capture screenshot on failures**

4.1: Update Login test with Screenshot under “Testcase”

**Step 5: Read common file from INI file (initialization file)**

5.1: ADD “config.ini” file in configuration folder

5.2: Create “readProperties.py” utility file under the utility package to read common folder

5.3: Replace hardcoded values in Login testcase [INI utility TC ]

**Step 6: Adding logs to the testcase**

6.1: Add customLogger.py under utility package

6.2: Add logs to the login testcase

**Step 7: Run Tests on Desired Browser/CrossBrowser/parallel**

7.1: update conftest.py with required fixutures which accept comment line argument (browser name)

7.2: Pass browser name as argument in command line