Framework

It is a standard set of rules, guidelines and best practices followed to efficiently convert manual test cases into automation scripts.

Note:

- One of the most challenging aspects in converting manual test cases into automation scripts problems is that they are highly unstructured. To develop the automation scripts to test the application better, you have to figure out what the problem or area of opportunity is. That's where the frameworks come in-they help you understand your problem in a new way.

Framework development involves following 3 major stages.

- ➔ **Framework Design.**
- ➔ **Framework Implementation.**
- ➔ **Framework Execution.**

Note:

- **Test frameworks** are an essential part of any successful automated testing process. They can reduce maintenance costs and testing efforts and will provide a higher return on investment (ROI) for QA teams looking to optimize their agile processes.
- A testing framework is a set of guidelines or rules used for creating and designing test cases.
- A framework is comprised of a combination of practices and tools that are designed to help QA professionals test more efficiently.
- These guidelines could include coding standards, test-data handling methods, object repositories, processes for storing test results, or information on how to access external resources.
- Framework is a way to organize your complex code logics in more meaningful way to make our life easier.
- Frameworks help to structure our code and make maintenance easy.

BENEFITS OF A TEST AUTOMATION FRAMEWORK

Utilizing a framework for automated testing will increase a team's test speed and efficiency, improve test accuracy, and will reduce test maintenance costs as well as lower risks. They are essential to an efficient automated testing process for a few key reasons:

- **Improved test efficiency**
- **Lower maintenance costs**
- **Minimal manual intervention**
- **Maximum test coverage**
- **Reusability of code**

- **Easy to understand the code**
- **Easy to debug the code**
- **Rapid development of code**
- **Less error prone code**

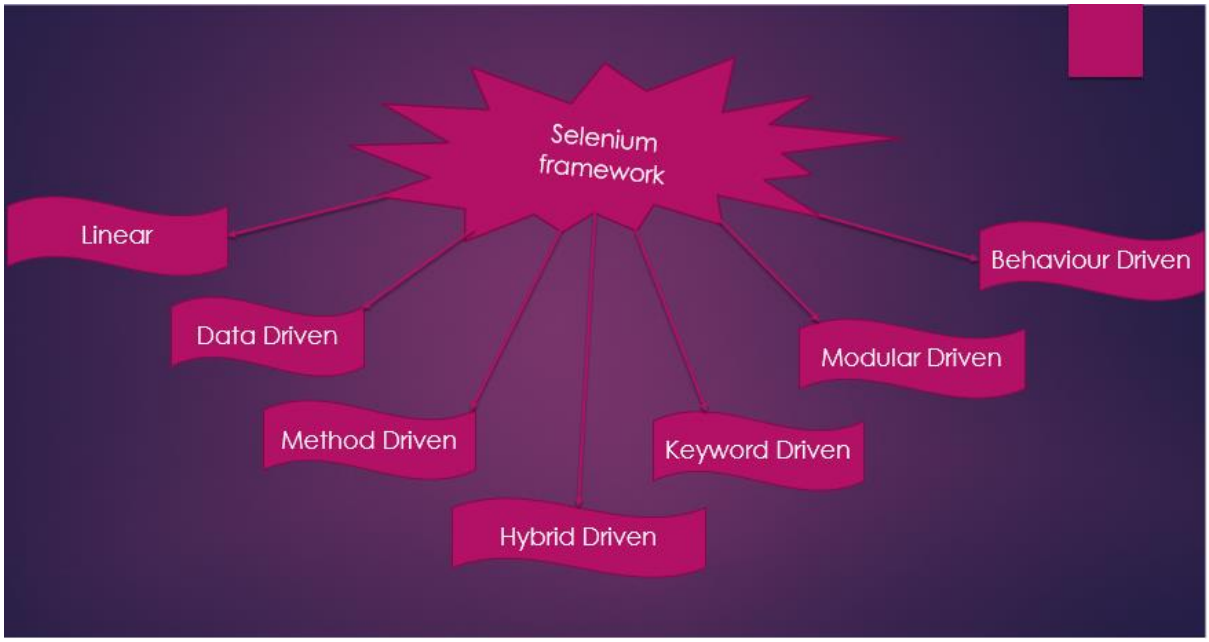
The above stages depend on type of the framework which is used in the automation.
We can broadly categorize the framework types as specified below irrespective of the automation tools.

- ➔ Linear automation framework
- ➔ Method driven automation framework. It is also called as function driven or action driver automation framework
- ➔ Modular driven automation framework
- ➔ Data driven automation framework
- ➔ Keyword driven automation framework
- ➔ Hybrid driven automation framework
- ➔ Behaviour driven automation framework

Note:

- With respect to selenium there are popular frameworks which are implemented using any one of the above types.

Ex: Page object module (POM), Robot framework, Behave framework.



LINEAR AUTOMATION FRAMEWORK

With a linear test automation framework, also referred to as a record-and-playback framework, testers don't need to write code to create functions and the steps are written in a sequential order. In this process, the tester records each step such as navigation, user input, or checkpoints, and then plays the script back automatically to conduct the test.

Advantages of a linear framework:

- There is no need to write custom code, so expertise in test automation is not necessary.
- This is one of the fastest ways to generate test scripts since they can be easily recorded in a minimal amount of time.
- The test workflow is easier to understand for any party involved in testing since the scripts are laid out in a sequential manner.
- This is also the easiest way to get up and running with automated testing, especially with a new tool. Most automated testing tools today will provide record-and-playback features, so you also won't need to plan extensively with this framework.

Disadvantages:

- The scripts developed using this framework aren't reusable. The data is hardcoded into the test script, meaning the test cases cannot be re-run with multiple sets and will need to be modified if the data is altered.
- Maintenance is considered a hassle because any changes to the application will require a lot of rework. This model is not particularly scalable as the scope of testing expands.

MODULAR BASED TESTING FRAMEWORK

Implementing a modular framework will require testers to divide the application under test into separate units, functions, or sections, each of which will be tested in isolation. After breaking down the application into individual modules, a test script is created for each part and then combined to build larger tests in a hierarchical fashion. These larger sets of tests will begin to represent various test cases.
A key strategy in using the modular framework is to build an abstraction layer, so that any changes made in individual sections won't affect the overarching module.

Advantages of a Modular Framework:

- If any changes are made to the application, only the module and its associated individual test script will needs to be fixed, meaning you won't have to tinker with the rest of the application and can leave it untouched.
- Creating test cases takes less effort because test scripts for different modules can be reused.

Disadvantages of a Modular Framework:

- Data is still hard-coded into the test script since the tests are executed separately, so you can't use multiple data sets.
- Programming knowledge is required to set up the framework.

Method Driven or Action Driven Framework

The Method architecture framework for automated testing is based on the modular framework, but has some additional benefits. Instead of dividing the application under test into the various scripts that need to be run, similar tasks within the scripts are identified and later grouped by function, so the application is ultimately broken down by common objectives. These functions are kept in a library which can be called upon by the test scripts whenever needed.

Advantages of a Method driven Testing Framework:

- Similar to the modular framework, utilizing this architecture will lead to a high level of modularization, which makes test maintenance and scalability easier and more cost effective.
- This framework has a higher degree of reusability because there is a library of common functions that can be used by multiple test scripts.

Disadvantages:

- Test data is still hard coded into the script. Therefore, any changes to the data will require changes to the scripts.
- Technical expertise is needed to write and analyse the common functions within the test scripts.
- Test scripts take more time to develop.

DATA-DRIVEN FRAMEWORK

Using a data-driven framework separates the test data from script logic, meaning testers can store data externally. Very frequently, testers find themselves in a situation where they need to test the same feature or function of an application multiple times with different sets of data. In these instances, it's critical that the test data not be hard-coded in the script itself, which is what happens with a Linear or Modular-based testing framework.

Setting up a data-driven test framework will allow the tester to store and pass the input/ output parameters to test scripts from an external data source, such as Excel Spreadsheets, Text Files, CSV files, SQL Tables, or DataBase repositories.
The test scripts are connected to the external data source and told to read and populate the necessary data when needed.

Advantages of a Data-Driven Framework:

- Tests can be executed with multiple data sets.
- Multiple scenarios can be tested quickly by varying the data, thereby reducing the number of scripts needed.
- Hard-coding data can be avoided so any changes to the test scripts do not affect the data being used and vice versa.
- You'll save time by executing more tests faster.

Disadvantages

- You'll need a highly-experienced tester who is proficient in various programming languages to properly utilize this framework design. They will need to identify and format the external data sources and to write code (create functions) that connect the tests to those external data sources seamlessly.
- Setting up a data-driven framework takes a significant amount of time.

KEYWORD-DRIVEN FRAMEWORK

In a keyword-driven framework, each function of the application under test is laid out in a table with a series of instructions in consecutive order for each test that needs to be run. In a similar fashion to the data-driven framework, the test data and script logic are separated in a keyword-driven framework, but this approach takes it a step further.

Advantages of Keyword-Driven Frameworks:

- Minimal scripting knowledge is needed.
- A single keyword can be used across multiple test scripts, so the code is reusable.
- Test scripts can be built independent of the application under test.

Disadvantages:

- The initial cost of setting up the framework is high. It is time-consuming and complex. The keywords need to be defined and the object repositories / libraries need to be set up.
- You need an employee with good test automation skills.

- Keywords can be a hassle to maintain when scaling a test operation. You will need to continue building out the repositories and keyword tables.

Hybrid Framework

A hybrid framework is a combination of any of the previously mentioned basic frameworks set up to leverage the advantages of some and mitigate the weaknesses of others.

Every application is different, and so should the processes used to test them. As more teams move to an agile model, setting up a flexible framework for automated testing is crucial. A hybrid framework can be more easily adapted to get the best test results.

Advantages:

The Hybrid framework is built with a number of reusable modules / function libraries that are developed with the following features in mind:

- ➔ **Maintainability** – Hybrid framework significantly reduces maintenance effort.
- ➔ **Re-usability** – It allows to reuse test cases and library functions.
- ➔ **Manageability** - effective test design, execution, and traceability.
- ➔ **Accessibility** – easy to design, develop, modify and debug test cases while executing.
- ➔ **Availability** – Allows to schedule automation execution.
- ➔ **Reliability** – due to advanced error handling and scenario recovery.
- ➔ **Flexibility** – framework independent of system or environment under test.
- ➔ **Measurability** – customizable reporting of test results ensures the quality output.

Behaviour Driven

Behaviour Driven Development (BDD) is a software development process that originally emerged from Test Driven Development (TDD).

BDD focuses on:

- Where to start in the process
- What to test and what not to test
- How much to test in one go
- What to call the tests
- How to understand why a test fails

At the heart of BDD is a rethinking of the approach to the Unit Testing and Acceptance Testing that naturally arise with these issues. For example, BDD suggests that Unit Test names be whole

sentences starting with a conditional verb ("should" in English for example) and should be written in order of business value.

Acceptance Tests should be written using the standard agile framework of a user story: "As a [role] I want [feature] so that [benefit]". Acceptance criteria should be written in terms of scenarios and implemented as classes: Given [initial context], When [event occurs], Then [ensure some outcomes].

✓ shifting from thinking in “tests” to thinking in “behaviour”.

✓ Collaboration between Business stakeholders, Business Analysts, QA Team and developers.

✓ Extends Test Driven Development (TDD) by utilizing natural language that non-technical stakeholders can understand.

✓ BDD frameworks such as Cucumber or Behave are an enabler, acting a “bridge” between Business & Technical Language.

Framework Design

A **framework design** is a simple visual structure that helps organize the information and ideas of a problem so you can work on it more effectively.

A **framework** is often composed of a relevant list of categories. These categories are developed from initial research that should be a part of every new project.

Note: Framework design solve several problems, including:

- Eliminating inconsistencies in the script.
- Increasing quality, efficiency, and effectiveness.
- Making design updates later in the design process less frustrating.

Initially automation lead or Senior automation engineer with their past experience will design the automation framework.

- Before you can have great ideas to prototype, test and launch, you have to understand and frame your problem in an approachable way.
    - What frameworks are and why we need them.
    - Conduct research to gather the content for your framework.
    - Identify the categories of information as the basis of your framework.
    - Organise the categories visually into a diagram.
    - Use the framework to describe the problem and focus areas for ideation.
- Now, start creating framework design to improve your team's ability to automate complex manual test cases effectively.

- **Designing of the framework can be categorised into following steps:**
    1. Required software and files.
    2. Standard python naming conventions.
    3. Standard folder structure.
    4. Set standard programming rules and guidelines.
    5. Develop reusable libraries.
        I. Generic Components.
        II. Selenium Core Components.
    6. Choose reporting mechanism.
    7. Decide how to integrate your framework with version control systems.
    8. Decide how to implement CI/CD.

9. Integrate your framework with other tools (Example: Integrating to Test Management Tools).

List out all the software which are required for the automation and then specify the folder structure and their purpose.

1. Pre-Requisite or Required software and files.

- List out all the software's which are required to automate the application.

Example:

1. Programming language -> **Python**.
2. Automation tool -> **Selenium.**
3. Integrated Development Environment -> **PyCharm**.
4. All required browsers -> **Chrome, Firefox.**
5. All driver executables -> **chromedriver, gecko driver.**
6. Test data management software's -> **MS-Office (Excel), Properties files, JSON**.
7. Unit Testing Tool -> **Pytest**.
8. Data mining plugins -> **Pandas, PyJavaProperties.**
9. Handling non-html -> **Pyauto GUI, AutoIT.**
10. Image processing -> **Pillow**.
11. Reporting tool -> **Allure**.
12. Version control system -> **GIT.**
13. CI/CD tool **-> Jenkins.**
14. Cloud web testing platform -> **Sauce Labs**.

Note: All the above-mentioned software's and plugins should be installed and configured properly.

2. Standard python naming conventions.

Following are the python naming conventions or rules need to be followed while creating a project, python files, python classes and functions.

1. **General**
   - Avoid using names that are too general or too wordy.
2. **Project name**
   - Project name should contain the name of the application we are automating.
3. **Packages**
   - Package names should be all lower case.
   - When multiple words are needed, an underscore should be separated them.
   - It is usually preferable to stick to 1-word names.
4. **Modules**

- Module names should be all lower cases.
- When multiple words are needed, an underscore should be separated them.
- It is usually preferable to stick to 1-word names.

5. **Classes**
   - Class names should follow the upper-case camel case convention.
6. **Global (module-level) variables**
   - Global variables should be all lower cases.
   - Words in a global variable name should be separated by an underscore.
7. **Instance variable**
   - Instance variables should be all lower cases.
   - Words in an instance variable name should be separated by an underscore.
8. **Methods**
   - Method names should be all lower case.
   - Words in a method name should be separated by an underscore.
9. **Method arguments**
   - Instance methods should have their first argument named "self".
   - Class methods should have their first argument named "cls".
10. **Functions**
    - Functions names should be all lower case.
    - Words in a function name should be separated by an underscore.

Note: Name of Function/Method name should be action specific.

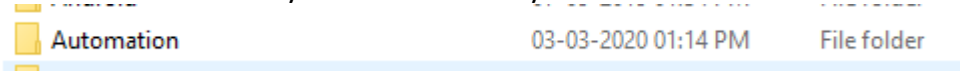
11. **Constants**
    - Constant names must be fully capitalized.
    - Words in a constant name should be separated by an underscore.
12. **Doc String**
    - Make sure you are adding doc string for all the functions (Libraries), which helps in understanding. (This is the rule we need to follow while coding).

3. Create standard folder structure

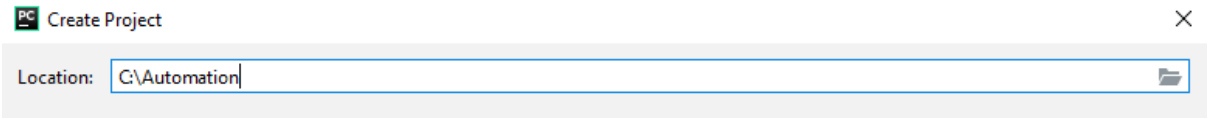
- We should ensure that we are following a standard folder structure, so that automation scripts will execute in other machines also.
- Now, it's time to take a look at our framework's architecture. Before we write a single line of code, we need to decide how our framework will be structured. So that our framework is a sustainable, maintainable, and scalable architecture.
- Our main goal is to keep it simple, easy to read and easy to maintain so based on those considerations.
- We highly recommend that you follow this architecture or at least the core principles behind it.

Steps to create the standard folder structure:

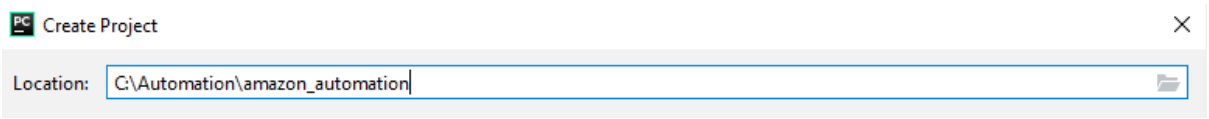
Step 1: Create a folder namely "Automation" in your "C" drive.



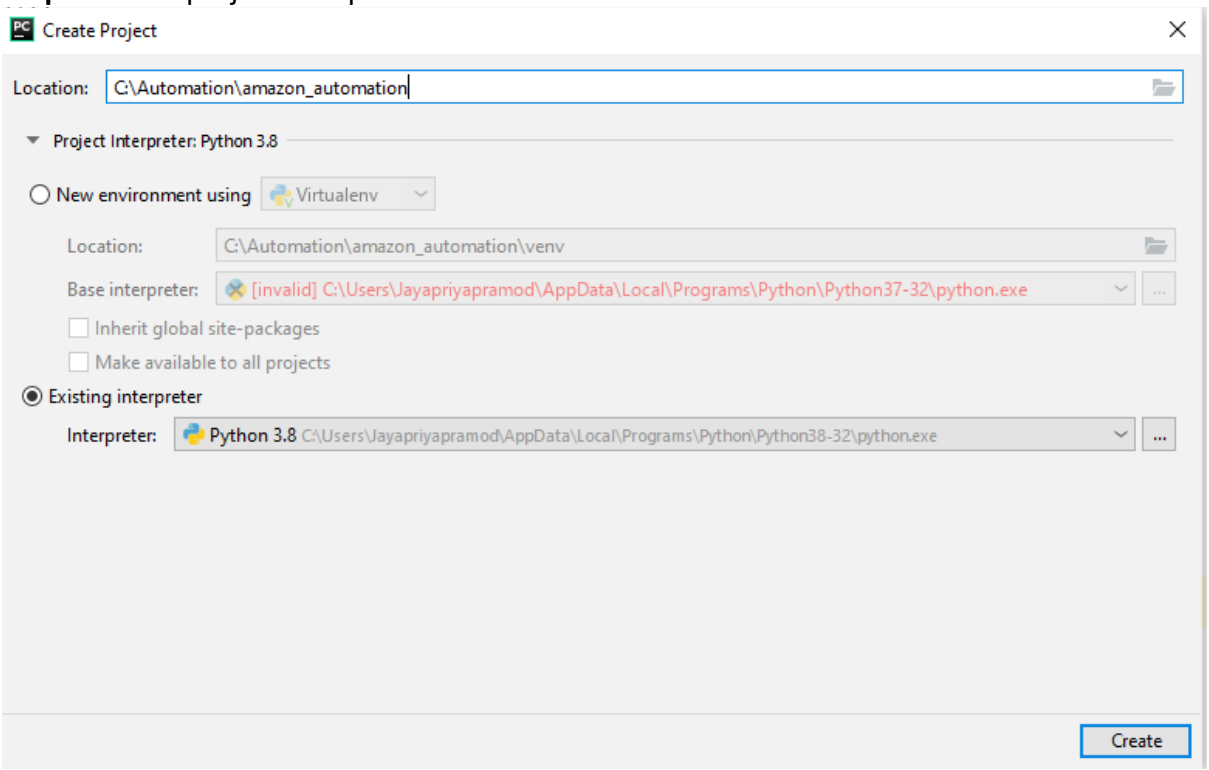
Step 2: Open PyCharm, and navigate to “Automation” folder.



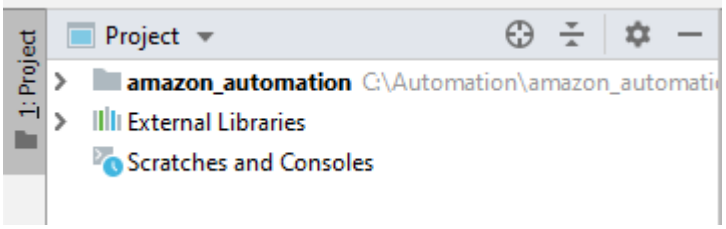
Step 3: Mention the project name as “**amazon_automation**”.



Step 4: Select project interpreter.



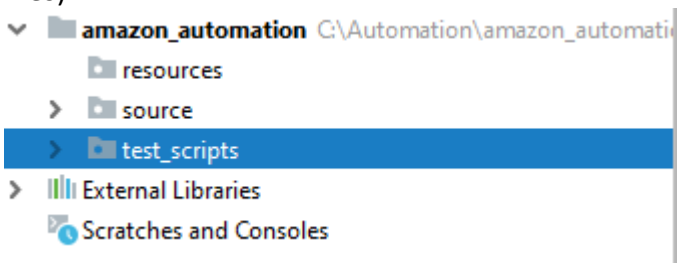
Step 5: Click on “Create” button.
Step 6: An empty project will be created.



Step 7: After creating the project, create following packages and directories under project “amazon_automation”

- **source** package: to keep all the libraries (project specific & generic code).
- **test_scripts** package: to keep all the instructions to test the application under test based on the manual test cases.

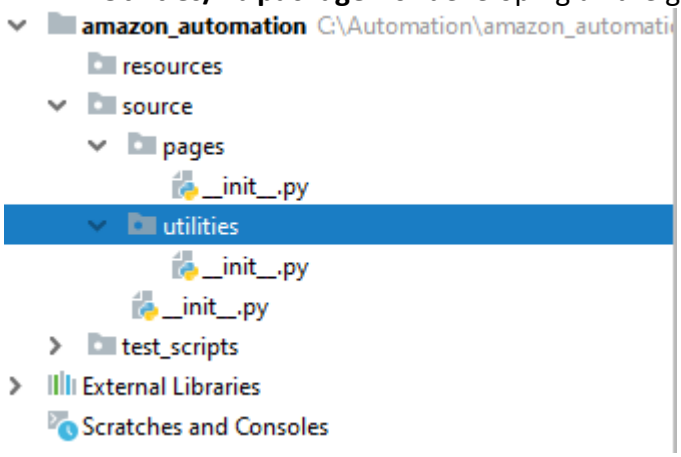
- **resources** directory: to keep all the data files (excel, json & properties files), driver executables (chrome driver, gecko driver executables) and reporting files (html, xml files).



Step 8: Create the following packages under "source" package.

➔ pages
➔ utilities/lib

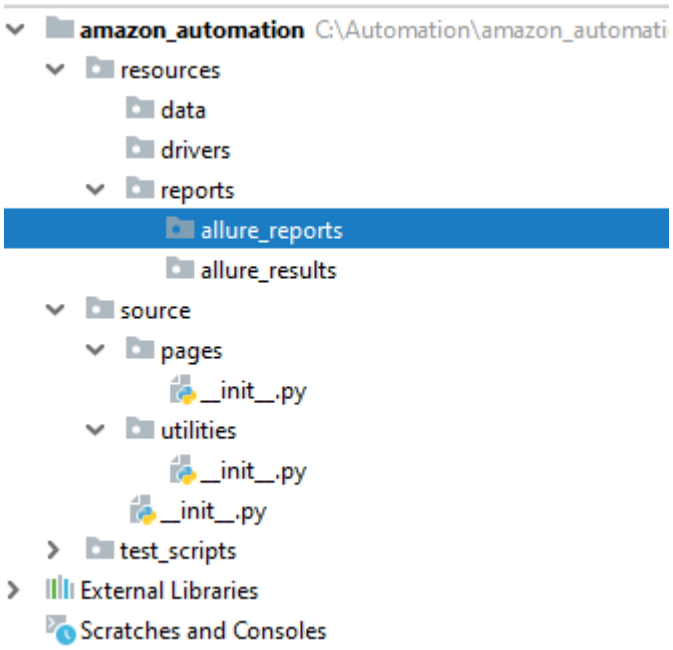
- **pages package:** for developing all the page classes (project specific reusable function).
- **Utilities/lib package:** for developing all the general specific reusable functions.



Step 9: Create following directories under resources directory.

➔ drivers
➔ reports
➔ data

- **drivers**: to keep all the browser specific executable files.
- **reports**: to keep all the execution results as a html file.
- **data**: to keep all the test data files (excel, properties, xml or json files)



Step 6: Create the following folders under "Resources"

- ➔ Drivers
- ➔ Data

Step 7:

Test Scripts:

- This is the folder is used to write all the instructions to test the application under test based on the test cases.

Very Important: Rules to develop the test script package.

13. Test package name should start with "test_"
    Example: **"test_scripts"**
14. Test module name should start with "test_"
    Example: "**test_login_validation**"
15. Test class name should start with "Test_"
    Example: "**Test_login**"
16. Test function/method name should start with "test_"
    Example: "**test_add_product_to_cart**"

Note: If you don't follow the above structure, then no test is going to execute. Because "PyTest" discovers the test package, test module, test class and test function/method only if it has "test_" as a prefix or else it's not going to discover any tests.