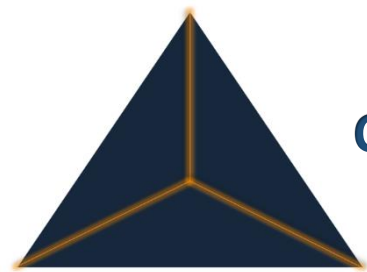


Credence IT Professional Training Institute

Python Framework Notes

Table of Contents

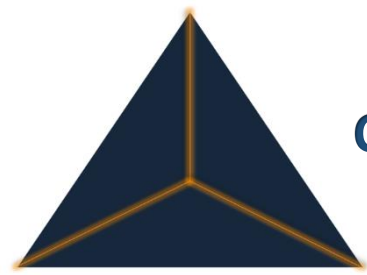
1. Step 1: Create a New Project & Install Required Packages	3
I. Set Up a New Project:	3
II. Install Required Libraries:.....	3
III. Explanation of Packages:.....	3
2. Step 2: Create Folder Structure	4
3. Step 3: Automate the Login Test Case	5
I. Create a Page Object Class:	5
II. Create a Test Case:.....	5
III. Setup conftest.py:.....	5
4. Step 4: Capture Screenshots on Failures.....	5
5. Step 5: Use an INI File for Common Values	6
I. Add config.ini:.....	6
II. Create a Reader for Config Values:.....	6
6. Step 6: Add Logs to Test Cases	7
I. Add Logger Class:	7
II. Add Logs to Test Cases:	7
7. Step 7: Run Tests on Desired Browser / Parallel Browsers	7
I. Command-Line Browser Selection:.....	7
II. Run Commands:.....	8
8. Step 8: Generate HTML Reports	8
I. Use pytest hooks in conftest.py to integrate HTML reports.	8
II. Command to Generate Report:	8
9. Step 9: Automate Data-Driven Test Cases (Updated).....	9
I. Prepare Test Data:	9
II. Read Excel Data:	9



Credence IT Professional Training Institute

Python Framework Notes

III.	Generate Fake Data:	9
10.	Step 10: Adding New Test Cases.....	9
11.	Step 11: Group Tests.....	9
I.	Add markers to test cases	9
II.	Add markers to pytest.ini.....	10
12.	Step 12: Retry Failed Tests	10
13.	Step 13: Test Execution Output Enhancements	10
I.	Improve Test Output with Pytest-sugar:.....	10
14.	Step 14: Run Tests in Jenkins.....	10
15.	Step 15: Push Build to GitHub	10



Credence IT Professional Training Institute

Python Framework Notes

1. Step 1: Create a New Project & Install Required Packages

I. Set Up a New Project:

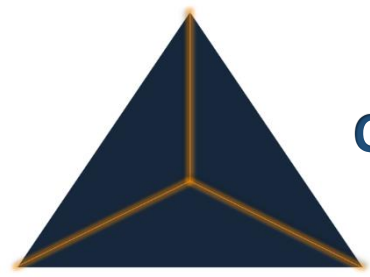
- Create a new Python project for your test automation.

II. Install Required Libraries:

- Open your terminal or command prompt and run the following commands:
 - `pip install selenium`
 - `pip install pytest`
 - `pip install pytest-html`
 - `pip install pytest-xdist`
 - `pip install openpyxl`
 - `pip install allure-pytest`
 - `pip install pytest-rerunfailures`
 - `pip install pytest-sugar`
 - `pip install pytest-faker`

III. Explanation of Packages:

- **Selenium:** Enables browser automation.
- **Pytest:** A framework to run and manage test cases.
- **Pytest-html:** Generates HTML test reports.
- **Pytest-xdist:** Allows parallel execution of tests.
- **Openpyxl:** Helps to read/write Excel files for data-driven testing.
- **Allure-pytest:** Generates advanced test reports in Allure format.
- **Pytest-rerunfailures:** Automatically reruns failed tests.
- **Pytest-sugar:** Enhances test output in the terminal with a cleaner, more visually appealing format.
- **Pytest-faker:** Provides utilities for generating fake test data using the Faker library.



Credence IT Professional Training Institute

Python Framework Notes

2. Step 2: Create Folder Structure

Organize your project with the following structure:

Project/

├─ Configurations/

| └─ config.ini

├─ Logs/

| └─ log_file.log

├─ Reports/

| └─ report.html

├─ Screenshots/

| └─ (Screenshots will be saved here)

├─ TestData/

| └─ test_data.xlsx

├─ pageObjects/

| └─ LoginPage.py

├─ testCases/

| └─ conftest.py

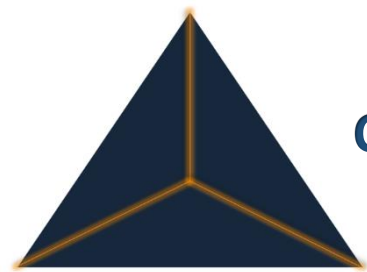
| └─ test_Login.py

├─ utilities/

| └─ Logger.py

| └─ ReadProperties.py

| └─ XLUtils.py



Credence IT Professional Training Institute

Python Framework Notes

3. Step 3: Automate the Login Test Case

I. Create a Page Object Class:

- Inside pageObjects, create a file named LoginPage.py where you define the locators and actions for the login page.

II. Create a Test Case:

- Write the test case in the testCases folder (e.g., test_Login.py).

III. Setup conftest.py:

- Add browser setup and teardown logic in conftest.py. This file helps you manage test fixtures for the entire project.

4. Step 4: Capture Screenshots on Failures

- Add a Screenshots folder to save screenshots.
- Use a utility function to capture screenshots when a test fails.
- Example:

```
import os
```

```
def take_screenshot(driver, test_name, status):
```

```
    """Captures a screenshot and saves it with the test name and status."""
```

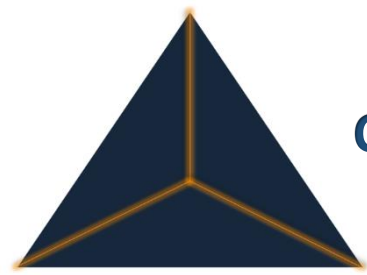
```
    screenshot_dir = "./Screenshots/"
```

```
    os.makedirs(screenshot_dir, exist_ok=True)
```

```
    screenshot_path = os.path.join(screenshot_dir, f"{test_name}_{status}.png")
```

```
    driver.save_screenshot(screenshot_path)
```

```
    print(f"Screenshot saved: {screenshot_path}")
```



Credence IT Professional Training Institute

Python Framework Notes

5. Step 5: Use an INI File for Common Values

I. Add config.ini:

- Inside the Configurations folder, add a config.ini file for storing values like URLs, usernames, and passwords.

Example

config.ini:

```
[common]
```

```
base_url = https://example.com
```

```
username = test_user
```

```
password = test_password
```

II. Create a Reader for Config Values:

- Add a ReadProperties.py file in the utilities folder to fetch values from config.ini.

Example:

```
import configparser
```

```
class ReadConfig:
```

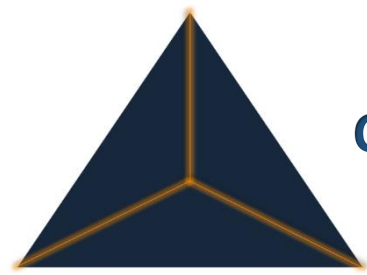
```
    config = configparser.ConfigParser()
```

```
    config.read('./Configurations/config.ini')
```

```
    @staticmethod
```

```
    def get_base_url():
```

```
        return ReadConfig.config.get('common', 'base_url')
```



Credence IT Professional Training Institute

Python Framework Notes

6. Step 6: Add Logs to Test Cases

I. Add Logger Class:

- Create Logger.py in the utilities folder.

Example:

```
import logging
```

```
class LogGenerator:
```

```
    @staticmethod
```

```
def loggen():
```

```
    logging.basicConfig(filename="./Logs/automation.log",
```

```
                        format='%(asctime)s - %(levelname)s - %(message)s',
```

```
                        level=logging.INFO)
```

```
    return logging.getLogger()
```

II. Add Logs to Test Cases:

- Use the logger in your test cases to track execution and errors.

7. Step 7: Run Tests on Desired Browser / Parallel Browsers

I. Command-Line Browser Selection:

- Update conftest.py to accept browser name as a command-line argument.

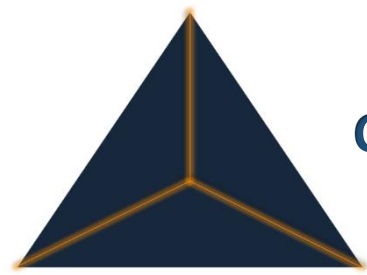
Example:

```
import pytest
```

```
from selenium import webdriver
```

```
def pytest_addoption(parser):
```

```
    parser.addoption("--browser")
```



Credence IT Professional Training Institute

Python Framework Notes

```
@pytest.fixture
def setup(request):
    browser = request.config.getoption("--browser")
    if browser == "chrome":
        driver = webdriver.Chrome()
    elif browser == "firefox":
        driver = webdriver.Firefox()
    else:
        driver = webdriver.Edge()
    driver.maximize_window()
    yield driver
    driver.quit()
```

II. Run Commands:

- For a specific browser:

```
pytest -s -v testCases/test_Login.py --browser chrome
```

8. Step 8: Generate HTML Reports

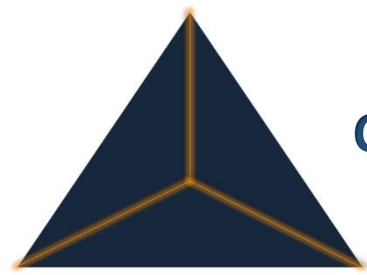
I. Use pytest hooks in conftest.py to integrate HTML reports.

Example:

```
def pytest_configure(config):
    config._metadata["Project Name"] = "Test Automation"
    config._metadata["Tester"] = "Your Name"
```

II. Command to Generate Report:

```
pytest -s -v --html=Reports/report.html testCases/test_Login.py --browser chrome
```

Credence IT Professional Training Institute

Python Framework Notes

9. Step 9: Automate Data-Driven Test Cases (Updated)

I. Prepare Test Data:

- Save an Excel file with test data in the TestData folder.

II. Read Excel Data:

- Use openpyxl in a utility (XLUtils.py) to read/write Excel data in tests.

III. Generate Fake Data:

- Use pytest-faker to create randomized test data on the fly. This is useful for testing forms, input fields, or dynamic data scenarios.
- Example use case:
 - Inject a faker fixture provided by pytest-faker into your test cases to generate names, emails, addresses, and more.

Example:

```
def test Example _with_fake_data(faker):  
  
    name = faker.name()  
  
    email = faker.email()  
  
    print(f"Generated Name: {name}, Email: {email}")
```

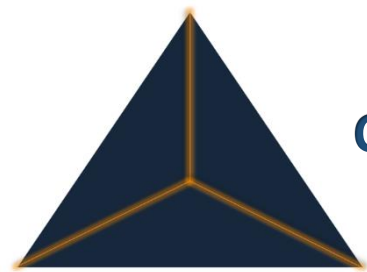
10. Step 10: Adding New Test Cases

- Add more test cases as needed in the testCases folder and follow the same structure

11. Step 11: Group Tests

I. Add markers to test cases

```
@pytest.mark.sanity  
@pytest.mark.regression
```



Credence IT Professional Training Institute

Python Framework Notes

II. Add markers to pytest.ini

```
[pytest]
markers =
    sanity
    regression
```

12. Step 12: Retry Failed Tests

- Use the pytest-rerunfailures library to retry failed tests automatically.
- Add the --reruns option in the command:

```
pytest -s -v --reruns 2 testCases/test_Login.py
```

13. Step 13: Test Execution Output Enhancements

I. Improve Test Output with Pytest-sugar:

- Install pytest-sugar to enhance terminal output. It provides a cleaner and more readable display of test progress and results.
- Test execution with this library will include a progress bar, better formatting, and test statuses like success, failure, or skipped.

14. Step 14: Run Tests in Jenkins

- Create a Jenkins job, configure the build to run pytest commands, and integrate GitHub for version control.

15. Step 15: Push Build to GitHub

- Use Git commands to push your automation project to GitHub.

-----END-----