45. pytest Framework - 4

Parameterization enables the Test Method to run multiple times with different data sets without the need for loops. Instead of writing looping statements, we can pass parameters to the test method, simplifying the process of repeating it with varying data.

- To pass parameters we use @pytest.mark.parametrize('num1,num2',[(1,1),(3,5),(10,10),(5,20)])

test_Parameterization.py

```
import pytest
class TestClass:
  @pytest.mark.parametrize('num1,num2',[(1,1),(3,5),(10,10),(5,20)])
  def test_calculation(self,num1,num2):
    assert num1==num2
```

To Execute - pytest -s -v day38-pytest4\test_Parameterization.py

Note - If we have huge data we use Data Driven Testing instead of parameterisation is used if huge data.

test_Parameterization2.py

```
import pytest
from selenium import webdriver
from selenium.webdriver.common.by import By
class TestClass:
  @pytest.mark.parametrize('user,pwd',
                           [("Admin", "admin123"),
                            ("adm", "admin123"),
                            ("Admin", "adm"),
                            ("adm", "adm")
                            ]
                           )
  def test_Login(self, user, pwd):
    options = webdriver.ChromeOptions()
    options.add_experimental_option("detach", True)
    self.driver = webdriver.Chrome(options=options)
    self.driver.get("https://opensource-demo.orangehrmlive.com/")
    self.driver.implicitly_wait(10)
    self.driver.find_element(By.NAME, "username").send_keys(user)
    self.driver.find_element(By.NAME, "password").send_keys(pwd)
    self.driver.find_element(By.TAG_NAME, "button").click()  # Signin
    try:
self.status = self.driver.find_element(By.XPATH, "//h6[normalize-space()='Dashboard']").is_displayed()
      self.driver.close()
      assert self.status == True
    except:
      self.driver.close()
      assert False
```

Note - we have to declare self keyword because we are declaring class variable inside a method

Pass command line options to Tests or Test Methods

• **parser.addoption** is a method provided by the pytest library to customize command-line options for your tests. It allows you to define additional command-line options that can be used when running pytest commands.
• Once options defined, you can access the values of these custom options within your test code using the **request.config** object.
• **pytest_addoption** is a hook function provided by pytest that is automatically called to allow you to add custom command-line options.
	`def pytest_addoption(parser):`
	`  parser.addoption("--browser")`
	`def browser(request):`
	`  return request.config.getoption("--browser")`

Customizing Test Metadata and Environment Information in Pytest HTML Reports

- **Hooks** are functions provided by the pytest framework used to Add/Delete/Modify Environment info in the HTML Report - **@pytest.mark.optionalhook**

Adding Environment info to HTML Report

```
def pytest_configure(config):
  config.stash[metadata_key]['Project Name'] = 'Orange HRM'
  config.stash[metadata_key]['Module Name'] = 'Login Module'
  config.stash[metadata_key]['Tester Name'] = 'KMR'
```

Delete/Modify Environment info to HTML Report

```
def pytest_metadata(metadata):
  metadata.pop("Python", None)
  metadata.pop("Plugins", None)
```

Note - Hooks and adoption are written in the fixtures in conftest.py file

Generating HTML reports - install package pytest-html

conftest.py

```
import pytest
from selenium import webdriver
from pytest_metadata.plugin import metadata_key
@pytest.fixture()
def setup(browser):
 if browser == "chrome":
    options = webdriver.ChromeOptions()
    options.add_experimental_option("detach", True)
    driver = webdriver.Chrome(options=options)
 elif browser == "edge":
    options = webdriver.EdgeOptions()
```

```
    options.add_experimental_option("detach", True)
    driver = webdriver.Edge(options=options)
  elif browser == "firefox":
    options = webdriver.FirefoxOptions()
    driver = webdriver.Firefox(options=options)
  yield driver  # Provide the driver instance to the test
  driver.quit()  # Ensure the browser is closed after the test
def pytest_addoption(parser):          # This will get the value from CLI
  parser.addoption("--browser")
@pytest.fixture()
def browser(request):      # This will return the browser value to setup method
  return request.config.getoption("--browser")
def pytest_configure(config):
   config.stash[metadata_key]['Project Name'] = 'Orange HRM'
   config.stash[metadata_key]['Module Name'] = 'Login Module'
   config.stash[metadata_key]['Tester Name'] = 'KMR'
@pytest.mark.optionalhook
def pytest_metadata(metadata):
  metadata.pop("Python", None)
  metadata.pop("Plugins", None)
```

test_CommandLine.py

```
from selenium.webdriver.common.by import By
class TestCLI:
  def test_Login(self,setup):
    self.driver=setup
    self.driver.get("https://opensource-demo.orangehrmlive.com/")
    self.driver.implicitly_wait(10)
    self.driver.find_element(By.NAME, "username").send_keys("Admin")
    self.driver.find_element(By.NAME, "password").send_keys("admin123")
    self.driver.find_element(By.TAG_NAME, "button").click()  # Signin
    try:
      self.status =
self.driver.find_element(By.XPATH,"//h6[normalize-space()='Dashboard']").is_displayed()
      assert self.status == True
    except:
      assert False
```

To Execute

- pytest -s -v --html=day38-pytest4\report.html day38-pytest4\test_CommandLine.py --browser chrome