Dummy Sites:

<https://www.lambdatest.com/selenium-playground/>

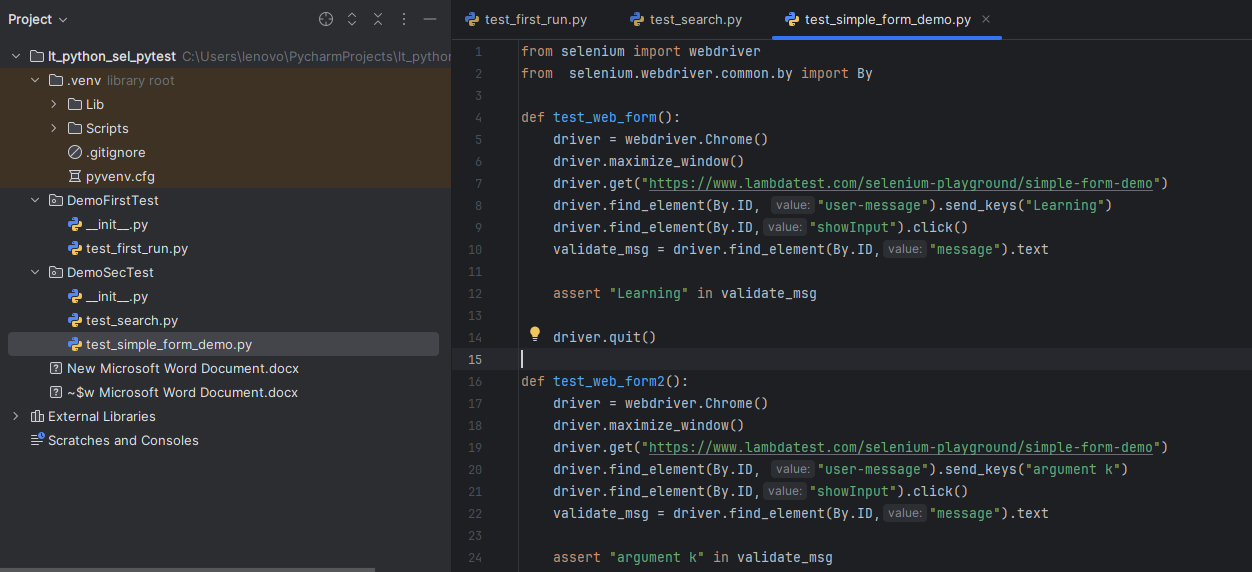
<https://ecommerce-playground.lambdatest.io/>

My Repo on GitHUB: <https://github.com/sainiashu/lt_python_sel_pytest>

basic commands:

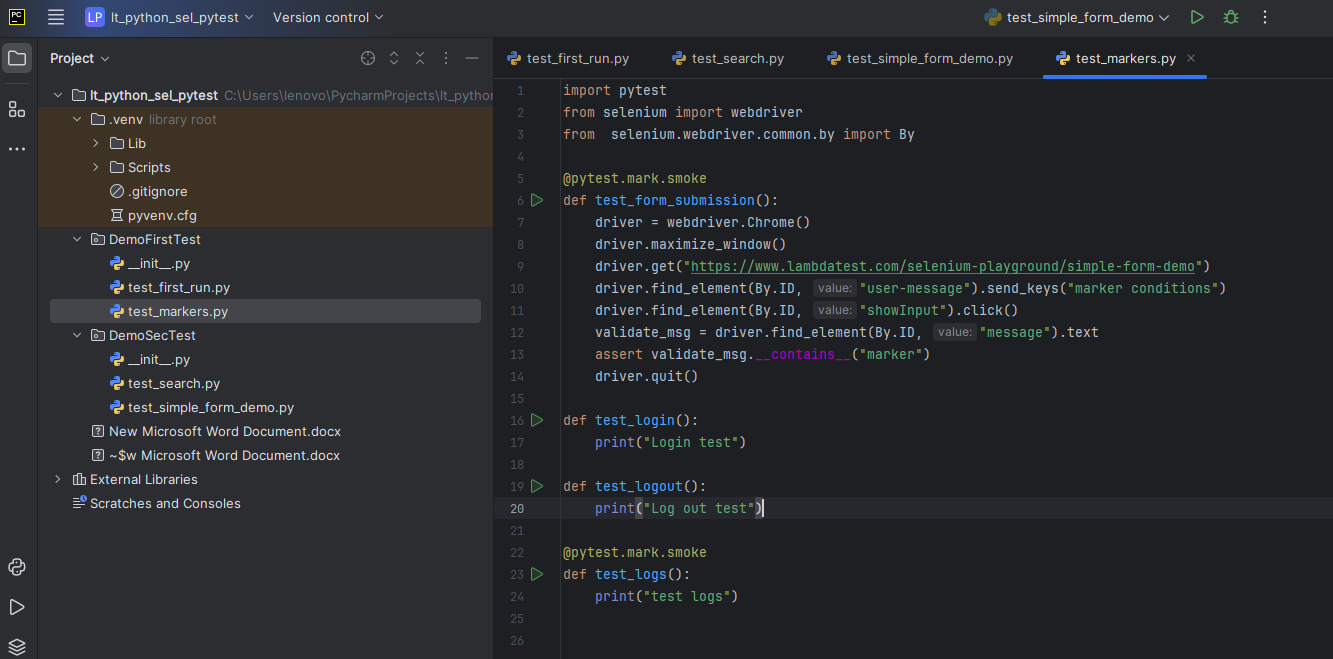
* Py.test: To run all tests
* Pytest –rA: Give you a detailed information about the executions.
* Pytest – v – rA : -v is used to get the more information about the test in the verbosed mode
* Pytest – v –rA “test location”

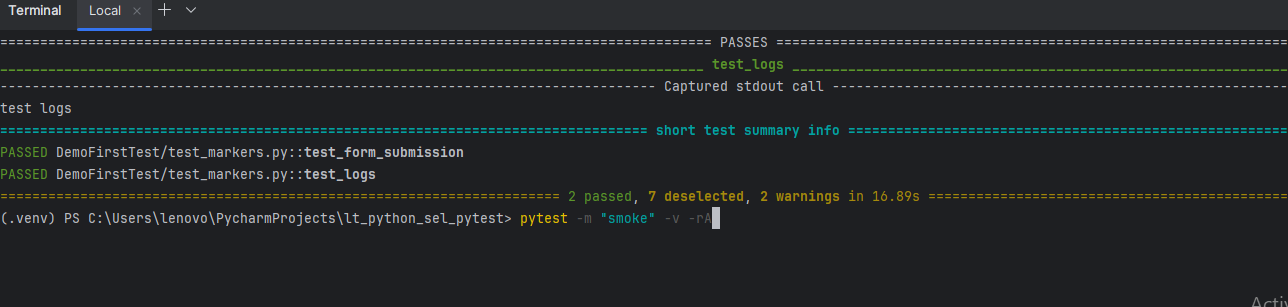
There are multiple ways to run test in pytest

* Individual test:
  + pytest -v -rA C:\Users\lenovo\PycharmProjects\lt\_python\_sel\_pytest\DemoSecTest\test\_search.py
* Tests under python packages
* Pass the python package name as an argument in the command line
  + pytest –v rA DemoSectest
* Also we can pass the python packages as as argument in the pytest command to run multiple packages as
  + pytest –v rA DemoSectest DemoFirstTest
* We can run the tests with the match string in the test name . it will execute all the tests those are fulfill the conditions and run the tests. Also reports shows the deselect tests details as well
* Use “-k” in the pytest command to match the string in all the tests those are fulfill the conditions
* You can run the test inside the python file using the below command:
  + (.venv) PS C:\Users\lenovo\PycharmProjects\lt\_python\_sel\_pytest\DemoSecTest> **pytest -v -rA test\_simple\_form\_demo.py::test\_web\_form**
* 

**PYTEST MARKERS**

**There are few inbuilts markers and we can create customized markers**





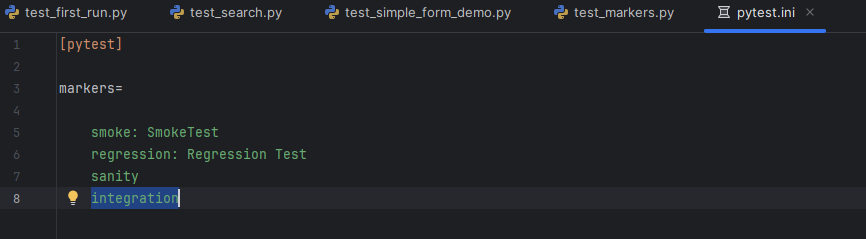
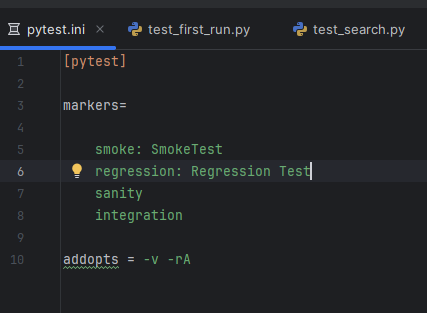
HERE YOU CAN SEE THE WARNINGS IN THE TERMINAL. So if you want to disablethe warnings the n you can write the following command

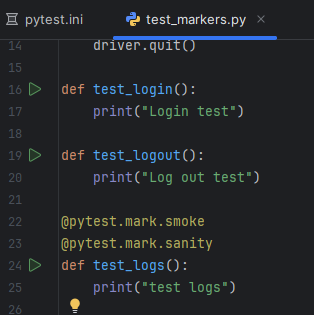
**pytest -m "smoke" -v -rA --disable-warnings**

**PYTEST.INI**

**We will create a pytest.ini file in the root directory and It works for the entire framework and it is configuration file.**

**pytest–markers : it will shows all the markers inbuilt and user define markers**

* **First we need to add pytest in square bracket “[pytest]** 
* **Create name “markers=”**
  + **Add custom markers name such as “smoke” also we can add the description here. Please see the screen shot above**
* **Rather than typing the –v –rA repeatedly in the command line we can pass this information in the same pytest.ini file**
* 

**Also we can use combination of markers and use it our test:**

**pytest -m "smoke and sanity"**

**it will execute only 1 test**

**Other Examples**

**Pytest –m “smoke and not sanity”**

**Now it will run 2 test cases**

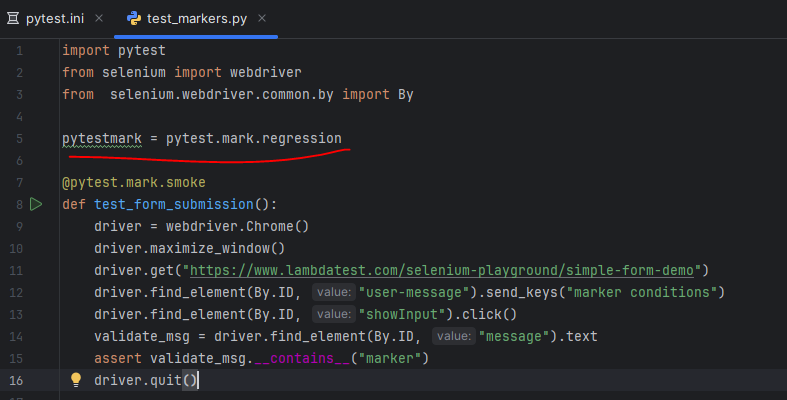
**Pytest –m “not smoke”**

**It will run all the test those are not smoke markers**

**Pytest -m “smoke or sanity”**

**AS OF NOW WE DEFINE THE FUNTIONS LEVEL**

**NOW WE CAN DEFINE IN THE FILE LEVEL**

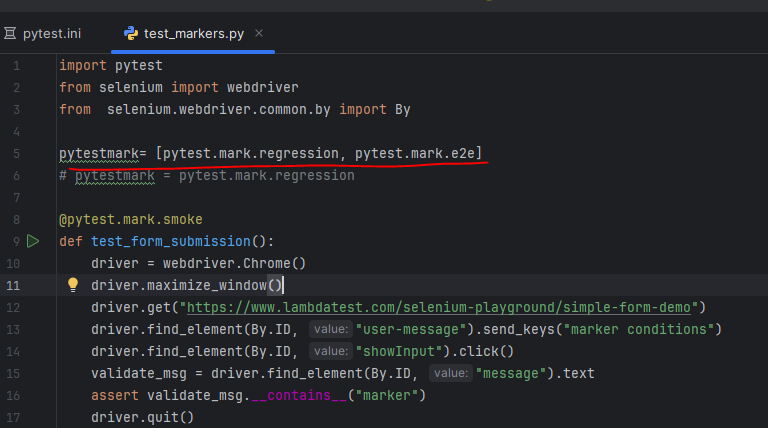


**Pytest –m “regression”**

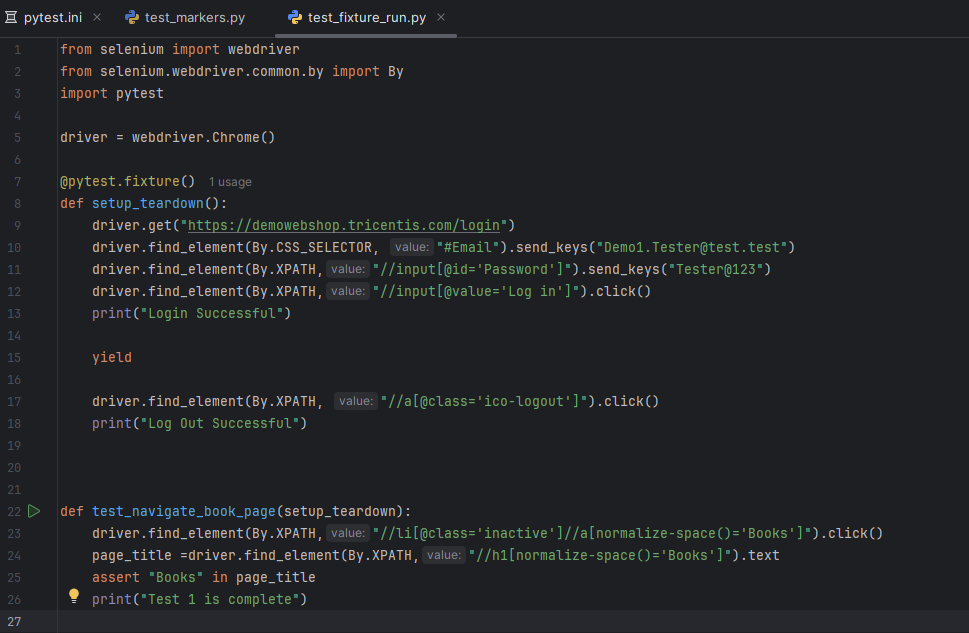
**Above command run all tests inside the tes\_markers.py file because of the following command:**

**pytestmark= pytest.mark.regression**

**IF WE WANT TO ADD MORE MARKERS IN THE FILE LEVEL THEN WE CAN USE THE SQUARE BRACKET “[]” AND PASS MARKER NAME INSIDE THE SQUARE BRACKET**

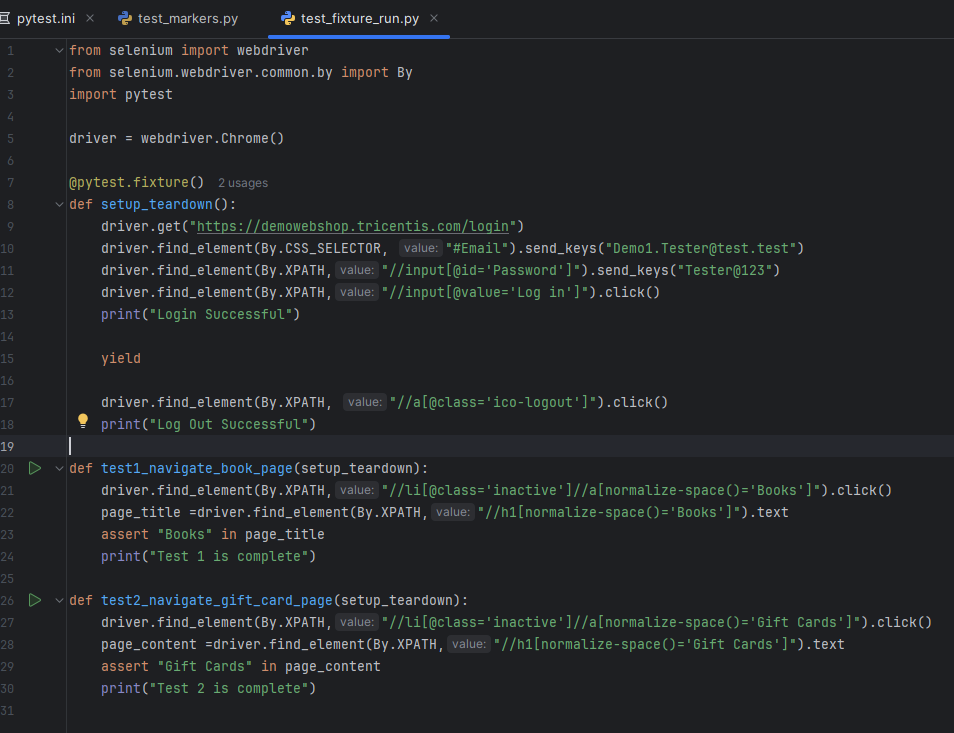


**FIXTURE USAGE**

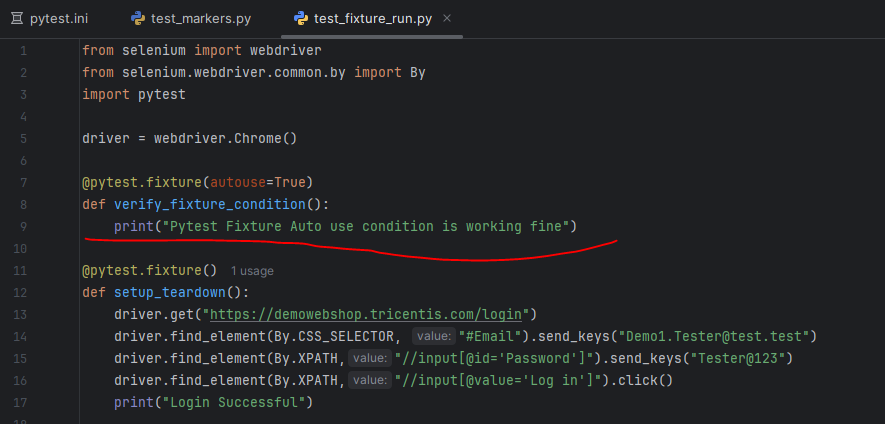


**Execute multiple test using @pytest.fixture()**

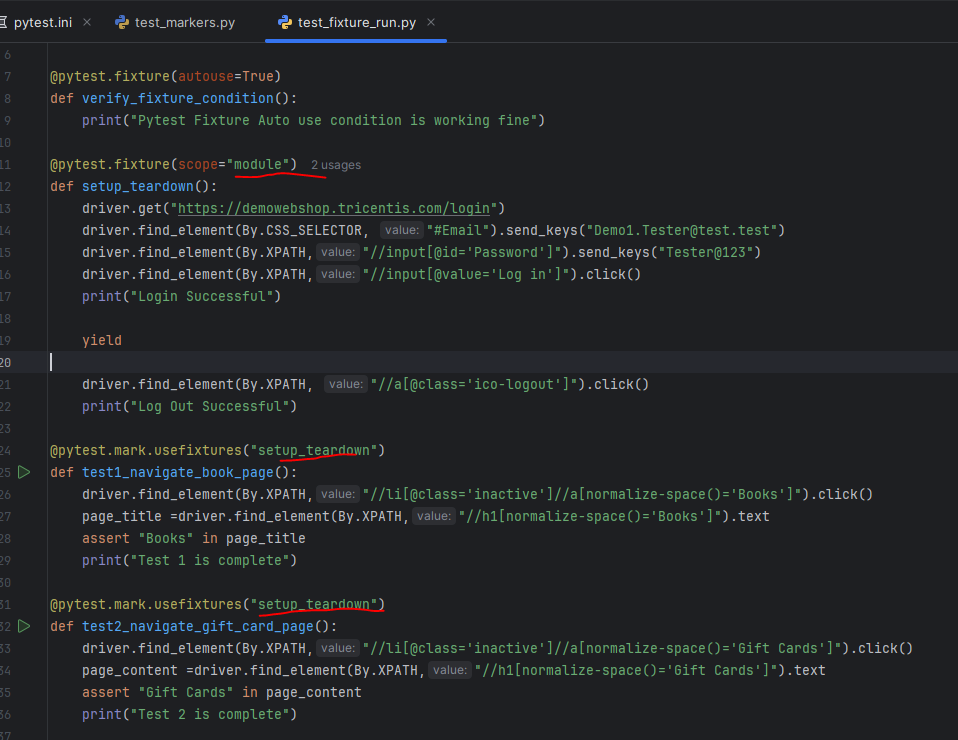
**In the below example you can see the that it will first run the setup\_teardown function before the actual test run such as test1 and test2**



**NOTE: Also you can use the @pytest.fixture(autouse=True) it will use the fixture by default for all the tests in case you forgot to mentioned the setup\_teardown fixture function into the test**



**By default scope of fixture is function. But we can set to class, module, package or session**



**In the below image we can use the**

**@pytest.mark.usefixtures(“setup\_teardown”)**

@pytest.mark.usefixtures("setup\_teardown")  
def test1\_navigate\_book\_page():  
 driver.find\_element(By.XPATH,"//li[@class='inactive']//a[normalize-space()='Books']").click()  
 page\_title =driver.find\_element(By.XPATH,"//h1[normalize-space()='Books']").text  
 assert "Books" in page\_title  
 print("Test 1 is complete")

**Scope define in module level**

@pytest.fixture(autouse=True)  
def verify\_fixture\_condition():  
 print("Pytest Fixture Auto use condition is working fine")  
  
@pytest.fixture(scope="module")  
def setup\_teardown():  
 driver.get("https://demowebshop.tricentis.com/login")  
 driver.find\_element(By.CSS\_SELECTOR, "#Email").send\_keys("Demo1.Tester@test.test")  
 driver.find\_element(By.XPATH,"//input[@id='Password']").send\_keys("Tester@123")  
 driver.find\_element(By.XPATH,"//input[@value='Log in']").click()  
 print("Login Successful")  
  
 yield  
  
 driver.find\_element(By.XPATH, "//a[@class='ico-logout']").click()  
 print("Log Out Successful")

**If we define the fixture scope in function level then it will execute test one by one but always use setup\_teardown function which include both login and logout. However, in case of module login and logout done only one time**

**PYTEST PARAMTERIZE**

**PARAMETERIZE TEST WITH ONE DECORATOR**

from selenium import webdriver  
from selenium.webdriver.common.by import By  
  
import pytest  
  
@pytest.mark.parametrize("num1,num2,expected\_result",  
 [  
 ("10","20","30"),  
 ("30","20","50"),  
 ("5","5","30"),  
 ])  
  
def test\_parameter(num1,num2,expected\_result):  
 driver = webdriver.Chrome()  
 driver.get("https://www.lambdatest.com/selenium-playground/simple-form-demo")  
 driver.maximize\_window()  
 driver.find\_element(By.XPATH,"//input[@id='sum1']").send\_keys(num1)  
 driver.find\_element(By.XPATH,"//input[@id='sum2']").send\_keys(num2)  
 driver.find\_element(By.XPATH,"//button[normalize-space()='Get Sum']").click()  
 actual\_total =driver.find\_element(By.XPATH,"//p[@id='addmessage']").text  
  
 assert actual\_total == expected\_result,\  
 "Actual vs Expected values does not match"  
 driver.quit()

**PARAMETERIZE TEST WITH TWO DECORATOR**

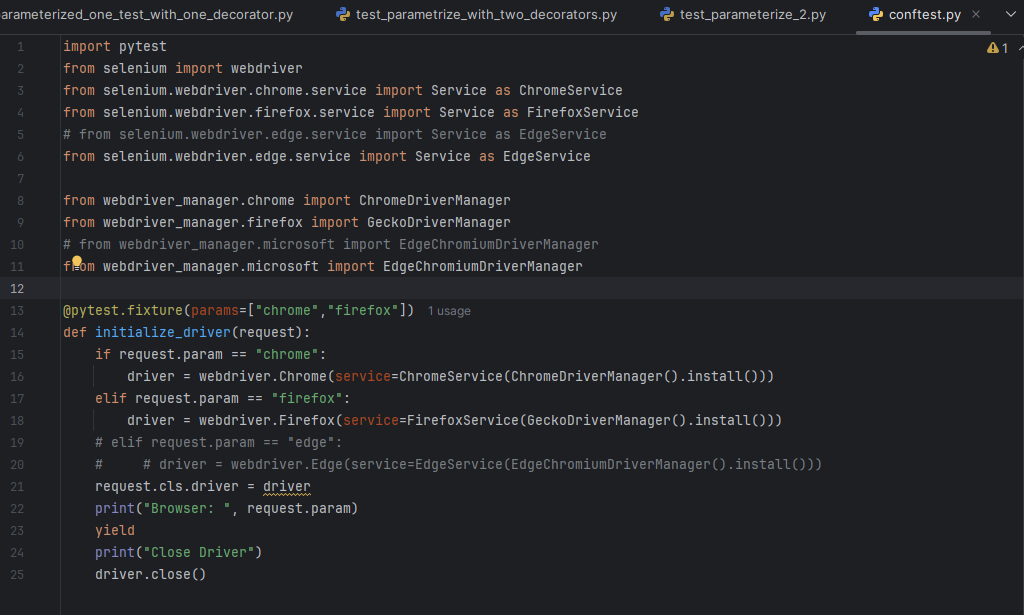
import pytest  
from selenium import webdriver  
  
from selenium.webdriver.chrome.service import Service as ChromeService  
from selenium.webdriver.firefox.service import Service as FirefoxService  
  
from webdriver\_manager.chrome import ChromeDriverManager  
from webdriver\_manager.firefox import GeckoDriverManager  
  
@pytest.mark.parametrize("browser",["chrome","firefox"])  
@pytest.mark.parametrize("url",["https://www.lambdatest.com/selenium-playground/simple-form-demo",  
 "https://demowebshop.tricentis.com/"]  
 )  
def test\_open\_browser(browser,url):  
 if browser == "chrome":  
 driver = webdriver.Chrome(service=ChromeService(ChromeDriverManager().install()))  
 driver.get(url)  
 elif browser == "firefox":  
 driver = webdriver.Firefox(service=FirefoxService(GeckoDriverManager().install()))  
 driver.get(url)  
 else:  
 raise ValueError("Browser not supported")  
  
 driver.maximize\_window()  
  
 driver.quit()

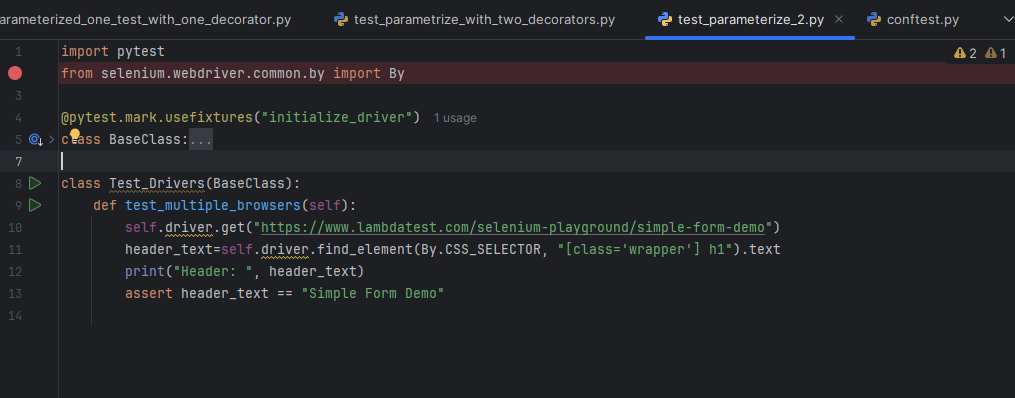
**ANOTHER EXAMPLE OF PARAMETERIZATOIN IN PYTEST**

import pytest  
from selenium import webdriver  
from selenium.webdriver.common.by import By  
  
from selenium.webdriver.chrome.service import Service as ChromeService  
from selenium.webdriver.firefox.service import Service as FirefoxService  
# from selenium.webdriver.edge.service import Service as EdgeService  
from selenium.webdriver.edge.service import Service as EdgeService  
  
from webdriver\_manager.chrome import ChromeDriverManager  
from webdriver\_manager.firefox import GeckoDriverManager  
# from webdriver\_manager.microsoft import EdgeChromiumDriverManager  
from webdriver\_manager.microsoft import EdgeChromiumDriverManager  
  
@pytest.fixture(params=["chrome","firefox"])  
def initialize\_driver(request):  
 if request.param == "chrome":  
 driver = webdriver.Chrome(service=ChromeService(ChromeDriverManager().install()))  
 elif request.param == "firefox":  
 driver = webdriver.Firefox(service=FirefoxService(GeckoDriverManager().install()))  
 # elif request.param == "edge":  
 # # driver = webdriver.Edge(service=EdgeService(EdgeChromiumDriverManager().install()))  
 request.cls.driver = driver  
 print("Browser: ", request.param)  
 yield  
 print("Close Driver")  
 driver.close()  
  
  
@pytest.mark.usefixtures("initialize\_driver")  
class BaseClass:  
 pass  
  
class Test\_Drivers(BaseClass):  
 def test\_multiple\_browsers(self):  
 self.driver.get("https://www.lambdatest.com/selenium-playground/simple-form-demo")  
 header\_text=self.driver.find\_element(By.CSS\_SELECTOR, "[class='wrapper'] h1").text  
 print("Header: ", header\_text)  
 assert header\_text == "Simple Form Demo"

**NOW WE USE THE CONFTEST.PY FILE IS PURPOSE TO PROVIDE FIXTURE FOR THE ENTIRE DIRECTORY**

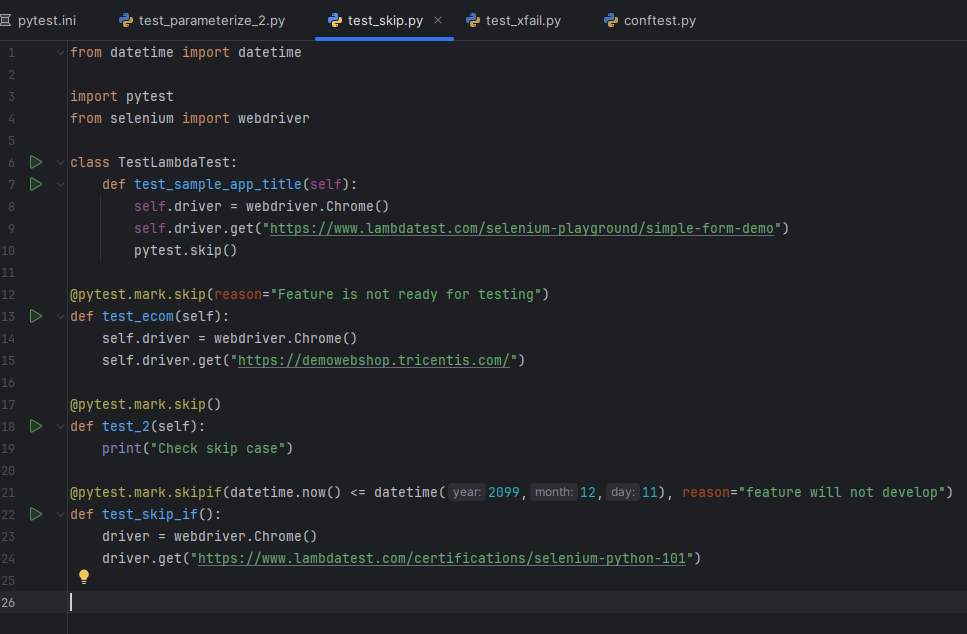
**Cut the fixture-related code and pasted into the conftest.py file**



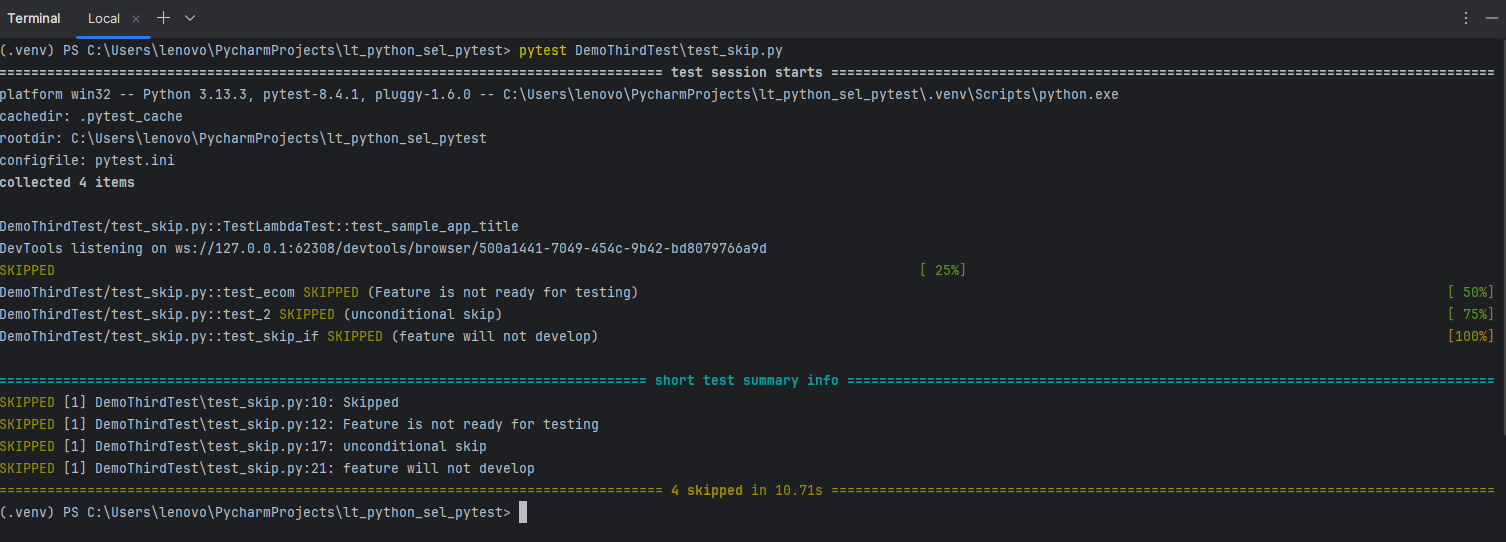
**Fixture content remove from the below mentioned file(test\_parameterizex\_2.py):**

**PYTEST SKIP TEST LOGIC**

**CODE BASE**



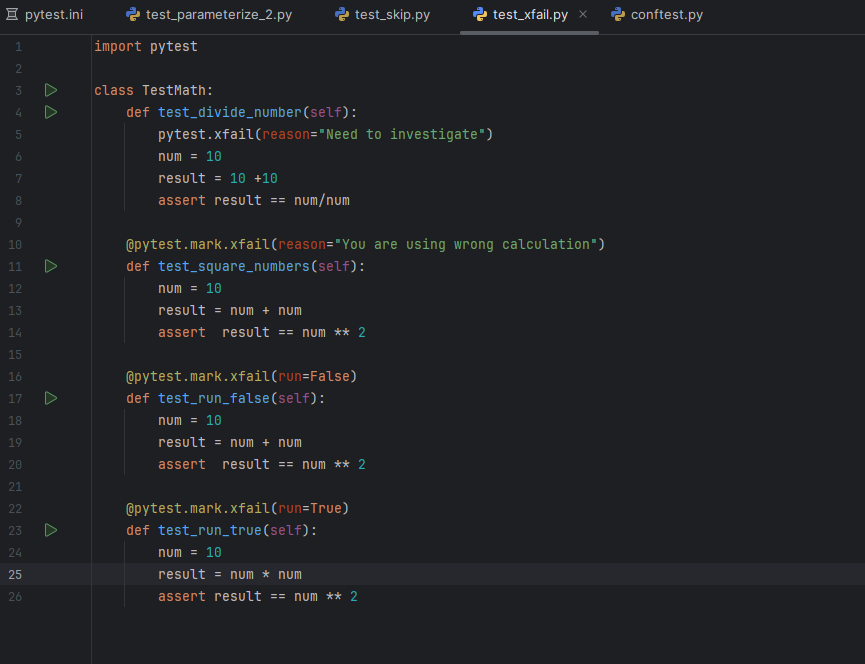
**OUTPUT OF THE ABOVE CODE**



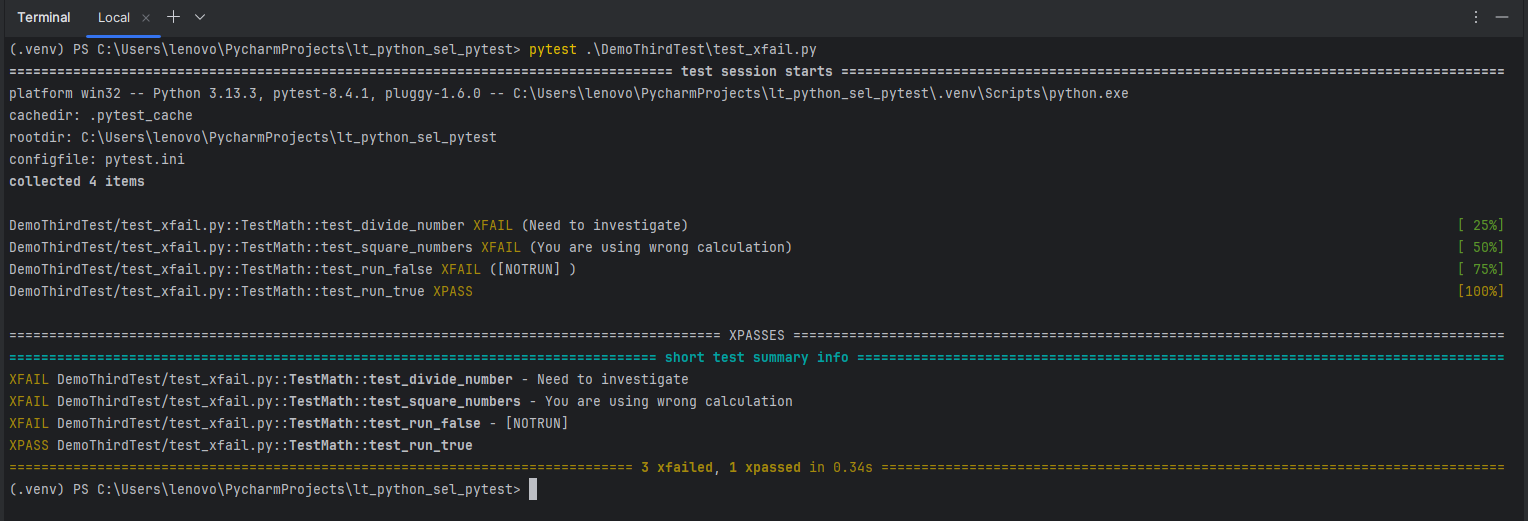
1. **If we pass pytest.skip in the function then it just skipped thefucntion without sharing the information inthe logs**
2. **If we use @pytest.mark.skip(reason=”feature is not ready for testing”) decorator that info. Will print in the report logs.**
3. **If we @pytest.mark.skip()this case it will print this   
   DemoThirdTest/test\_skip.py::test\_2 SKIPPED (unconditional skip)**
4. **If we use**
   1. @pytest.mark.skipif(datetime.now() <= datetime(2099,12,11), reason="feature will not develop")

**Output: DemoThirdTest/test\_skip.py::test\_skip\_if SKIPPED (feature will not develop)**

1. **Pytest.xfail EXAMPLES**

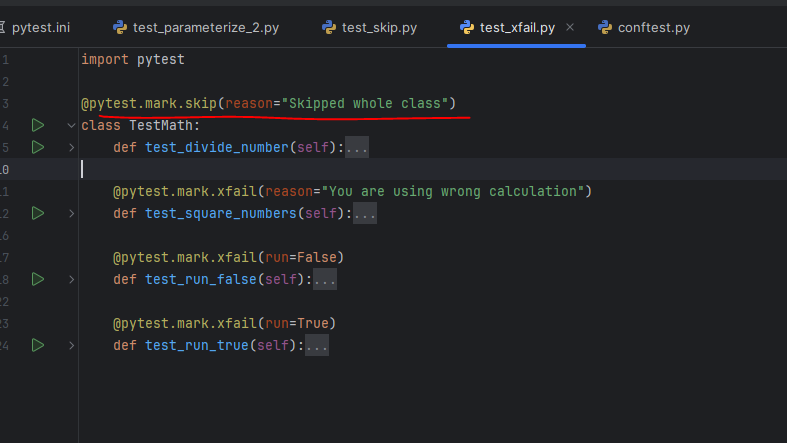


**OUTPUT:**



**SKIP THE COMPLETE CLASS**

**@pytest.mark.skip(reason=”skip the whole class”)**



**Output**

**================================================================================= short test summary info =================================================================================**

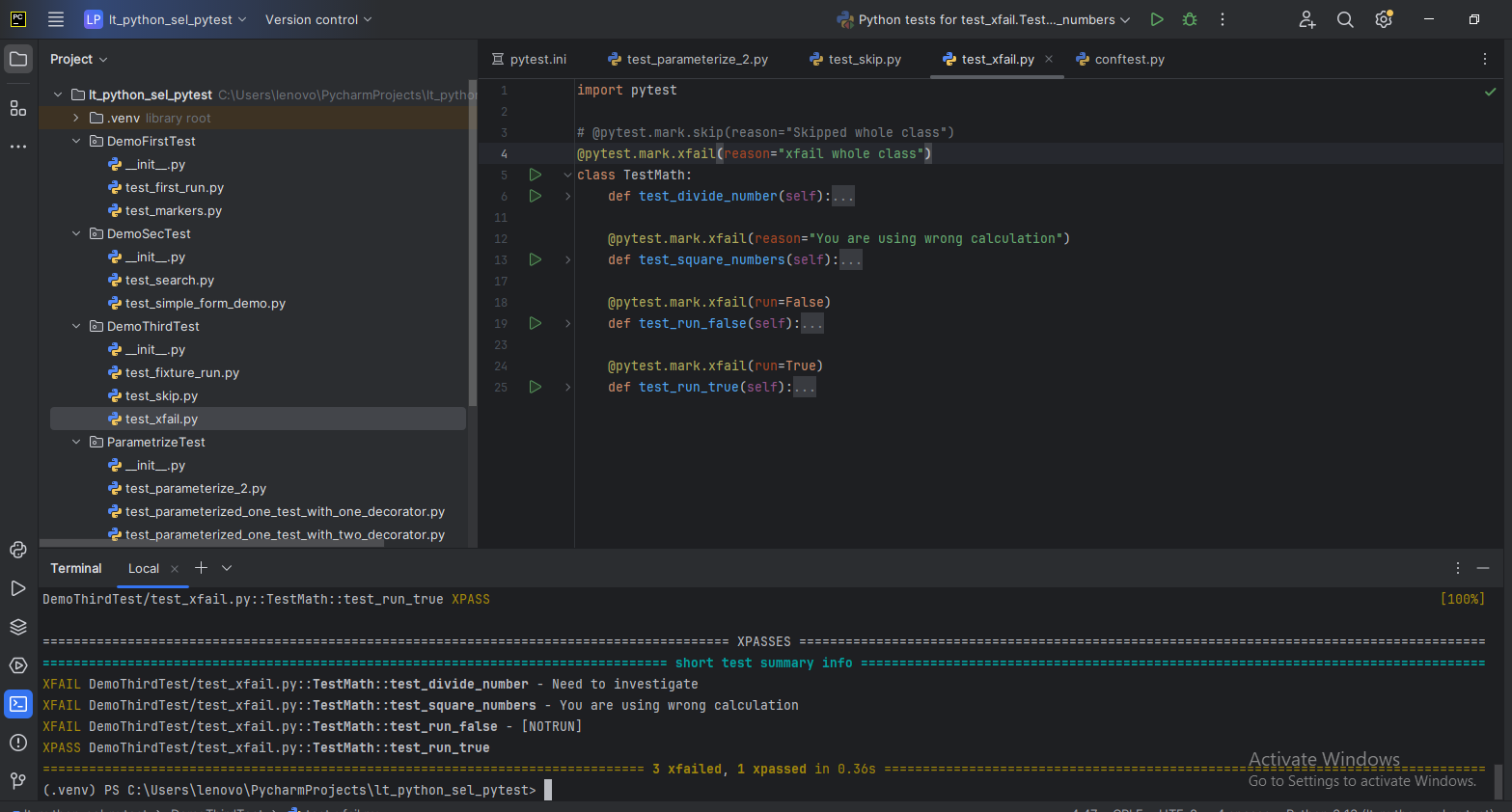
**SKIPPED [1] DemoThirdTest\test\_xfail.py: Skipped whole class**

**SKIPPED [1] DemoThirdTest\test\_xfail.py:11: Skipped whole class**

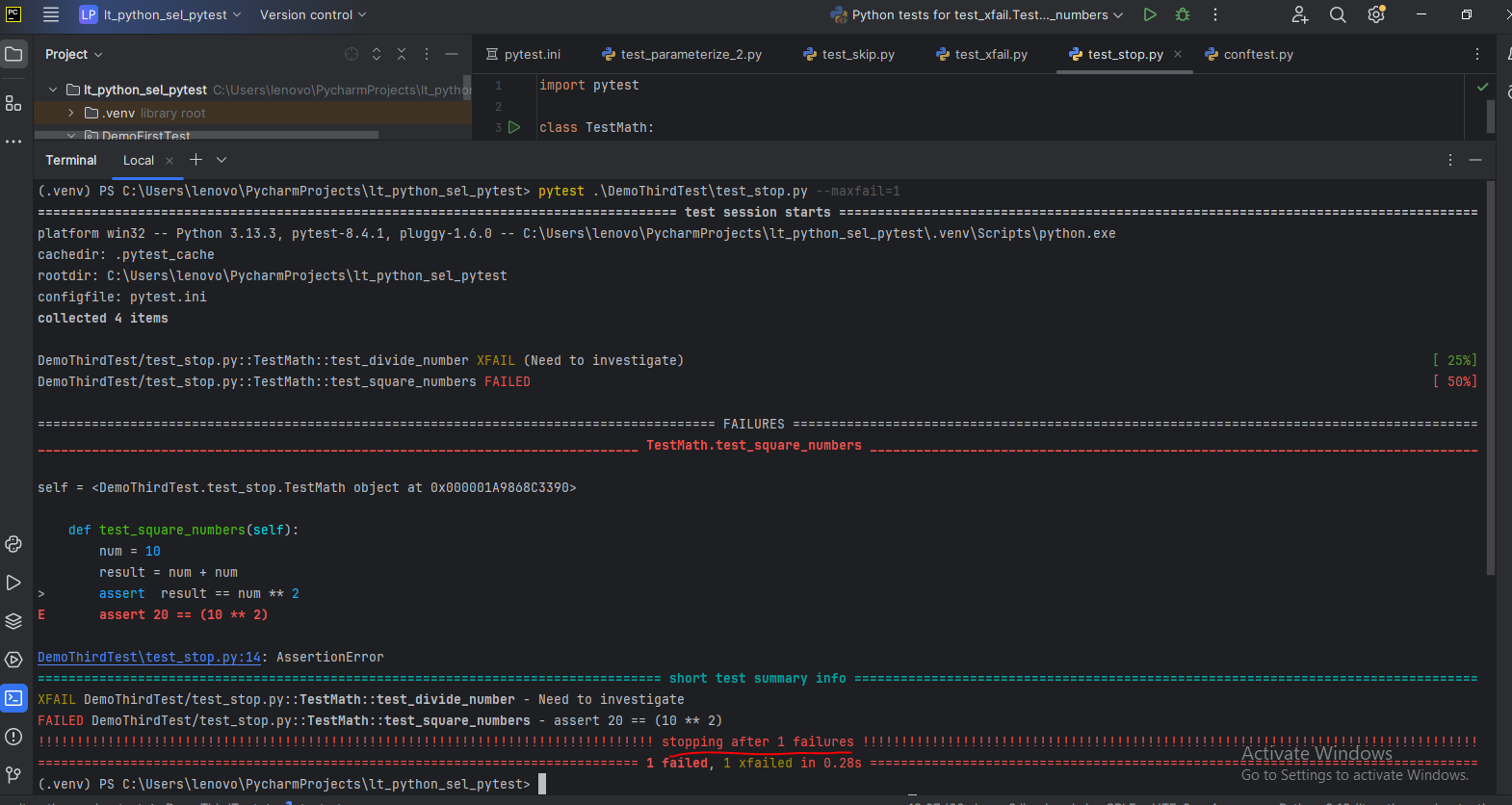
**SKIPPED [1] DemoThirdTest\test\_xfail.py:17: Skipped whole class**

**SKIPPED [1] DemoThirdTest\test\_xfail.py:23: Skipped whole class**

**XFAIL EXAMPLE**



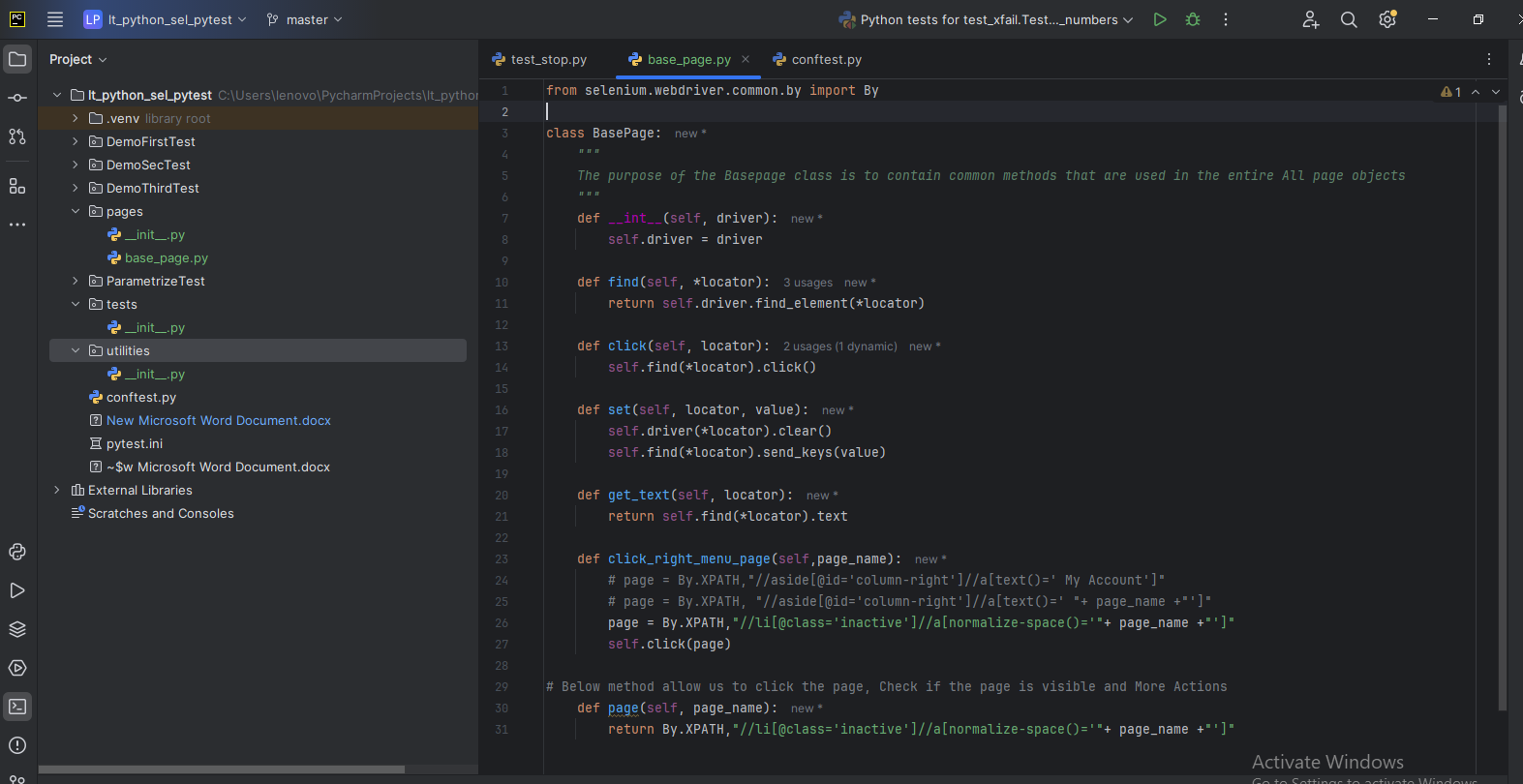
**STOP THE TEST AFTER TEST FAILURE  
USE THEE FOLLOWING COMMANDS  
(.venv) PS C:\Users\lenovo\PycharmProjects\lt\_python\_sel\_pytest> pytest .\DemoThirdTest\test\_stop.py --maxfail=1**



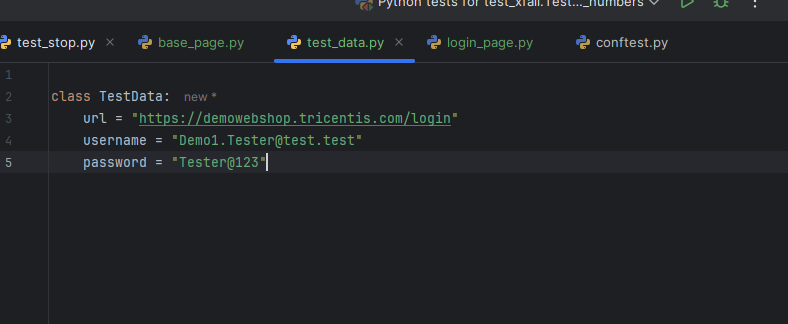
**======================PAGE OBJECT, TEST CLASS AND UTILITIES==============**

1. **First we need to create 3python packages for pages, tests and utilities**
2. **Create base\_page.py file under the pages python packages. It used to store all the common function that are used in the all the page objects classes**
3. **Now create a common methods and initialized the class and driver**

**base\_page.py**

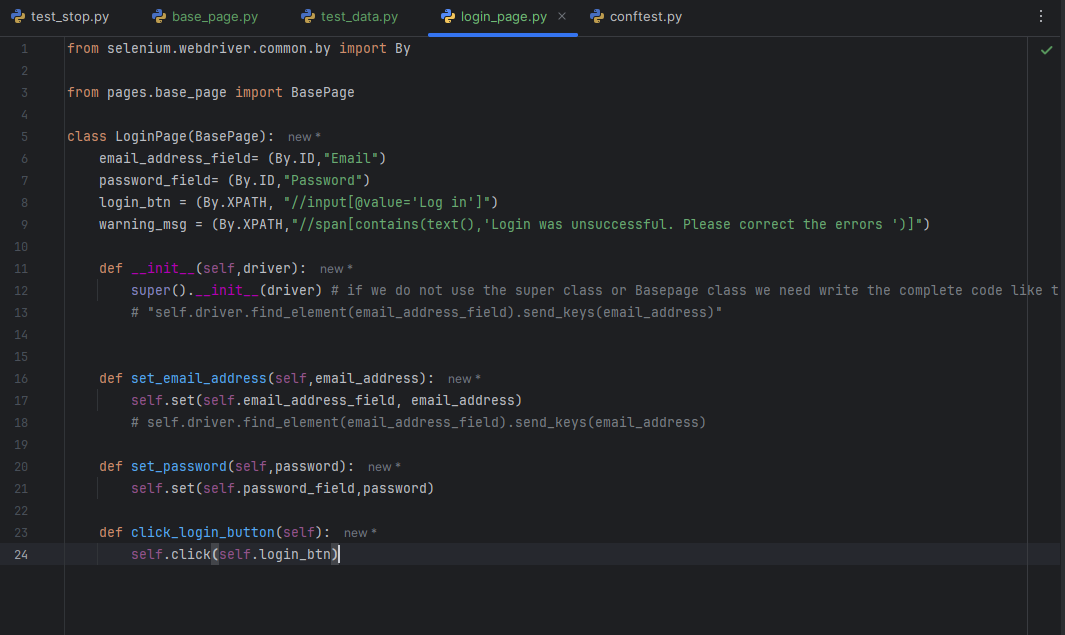


from selenium.webdriver.common.by import By  
  
class BasePage:  
 *"""  
 The purpose of the Basepage class is to contain common methods that are used in the entire All page objects  
 """* def \_\_int\_\_(self, driver):  
 self.driver = driver  
  
 def find(self, \*locator):  
 return self.driver.find\_element(\*locator)  
  
 def click(self, locator):  
 self.find(\*locator).click()  
  
 def set(self, locator, value):  
 self.driver(\*locator).clear()  
 self.find(\*locator).send\_keys(value)  
  
 def get\_text(self, locator):  
 return self.find(\*locator).text  
  
 def click\_right\_menu\_page(self,page\_name):  
 # page = By.XPATH,"//aside[@id='column-right']//a[text()=' My Account']"  
 # page = By.XPATH, "//aside[@id='column-right']//a[text()=' "+ page\_name +"']"  
 page = By.XPATH,"//li[@class='inactive']//a[normalize-space()='"+ page\_name +"']"  
 self.click(page)  
  
# Below method allow us to click the page, Check if the page is visible and More Actions  
 def page(self, page\_name):  
 return By.XPATH,"//li[@class='inactive']//a[normalize-space()='"+ page\_name +"']"

1. **Now create a test\_data.py file under utilities python package**
2. 

**Now added more locators in the login page**

from selenium.webdriver.common.by import By  
  
from pages.base\_page import BasePage  
  
class LoginPage(BasePage):  
 email\_address\_field= (By.ID,"Email")  
 password\_field= (By.ID,"Password")  
 login\_btn = (By.XPATH, "//input[@value='Log in']")  
 warning\_msg = (By.XPATH,"//span[contains(text(),'Login was unsuccessful. Please correct the errors ')]")  
  
 def \_\_init\_\_(self,driver):  
 super().\_\_init\_\_(driver) # if we do not use the super class or Basepage class we need write the complete code like this  
 # "self.driver.find\_element(email\_address\_field).send\_keys(email\_address)"  
  
  
 def set\_email\_address(self,email\_address):  
 self.set(self.email\_address\_field, email\_address)  
 # self.driver.find\_element(email\_address\_field).send\_keys(email\_address)  
  
 def set\_password(self,password):  
 self.set(self.password\_field,password)  
  
 def click\_login\_button(self):  
 self.click(self.login\_btn)



**MiSSING AREA NEED TO ADD HERE**

**LOGIN\_TEST.PY FILE AND OTHERS RELATED UPDATES**

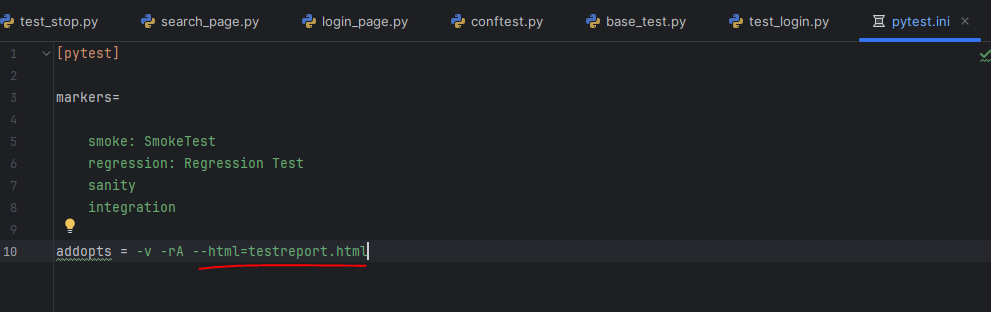
**======HTML REPORT**

**INSTALL: pip install pytest-html**

**Command line: pytest test\_login.py --html=testreport.html**

**It will generate the report under the test folder.**

**Also, you can update the pytest.ini file it will works for the entire project**



**=========ALLURE REPORT============**

**INSTALL ALLURE REPORT PACKAGE**

**Pip install allure-pytest**

**Pytest testfile name.py –alluredir=AllureReport**

**To convert or presentable form of the allure json and txt file we NEED TO USE ALLURE COMMAND LINE PACKAGE WHICH WE CAN INSTALL FROM THIS LINK:**[**https://repo.maven.apache.org/maven2/io/qameta/allure/allure-commandline**](https://repo.maven.apache.org/maven2/io/qameta/allure/allure-commandline)

**THEN DOWNLOAD THE LATEST ZIP FILE AND EXTRACT AT YOUR LOCAL SYSTE,**

**GO TO BIN FOLDER COPY THE PATH AND SET PATH IN TO THE ENVIRONMENT VARIABLES**

**After tht run following command to generate the allure report and report stored into the**

