Organizing Automated Tests with Advance Page Object Model

Page Object Model (POM) is an organized approach to testing web applications wherein pages are represented in the tests. The page objects contain the elements and behaviours of the associated page. This design works well because it is clear what is being tested and limits the user to actions available on that page.

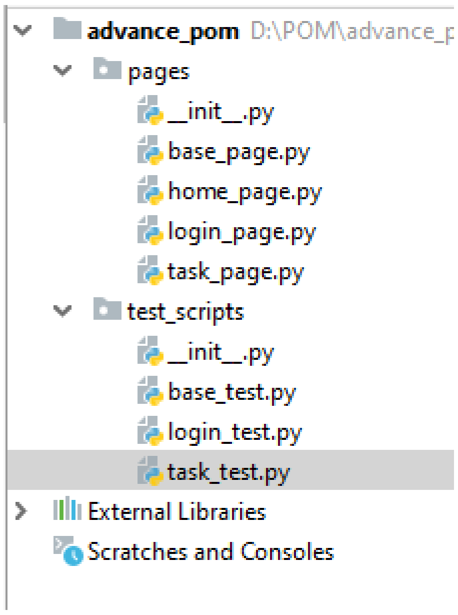
By separating the tests from the underlying data, not only is it now easier to write test cases, but you also only need to modify code in one place to update all tests. If the login error message changed to a different location, you would only need to update the LoginPage class and all the tests would still run fine. There is extra setup required to model the pages, but that time is quickly recovered when you get to creating your test suite.

Here, we are going to discuss another way of decorating the POM classes,

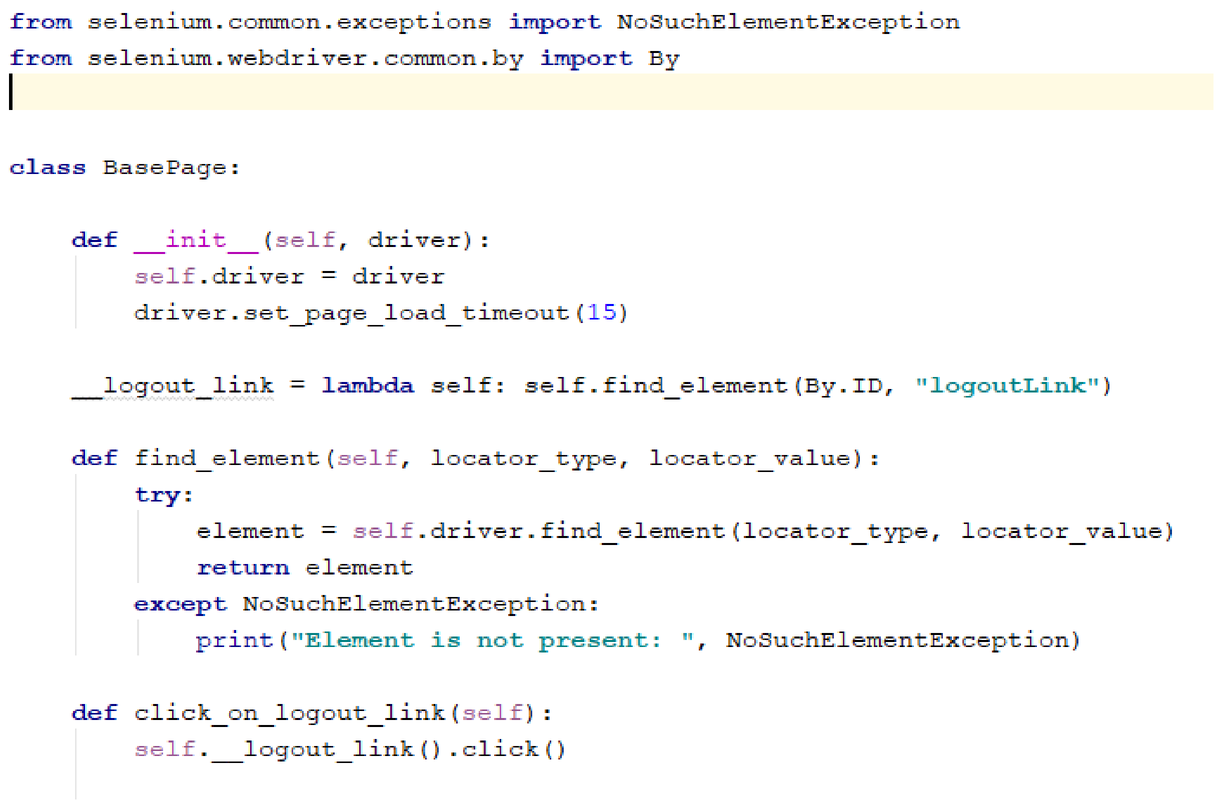
- Instead of just developing the behaviour's (ie. Function), we make those functions to return the object of next landing page.
  Example: If we develop a function to login to the application. That function has to enter the user name, password, click on the login button and should return the object of home page.
- Advantage of this way of decorating the page object model helps in avoiding creating the object of POM classes in each test scripts.

Create a project "advance_pom". And create a folder structure like below.

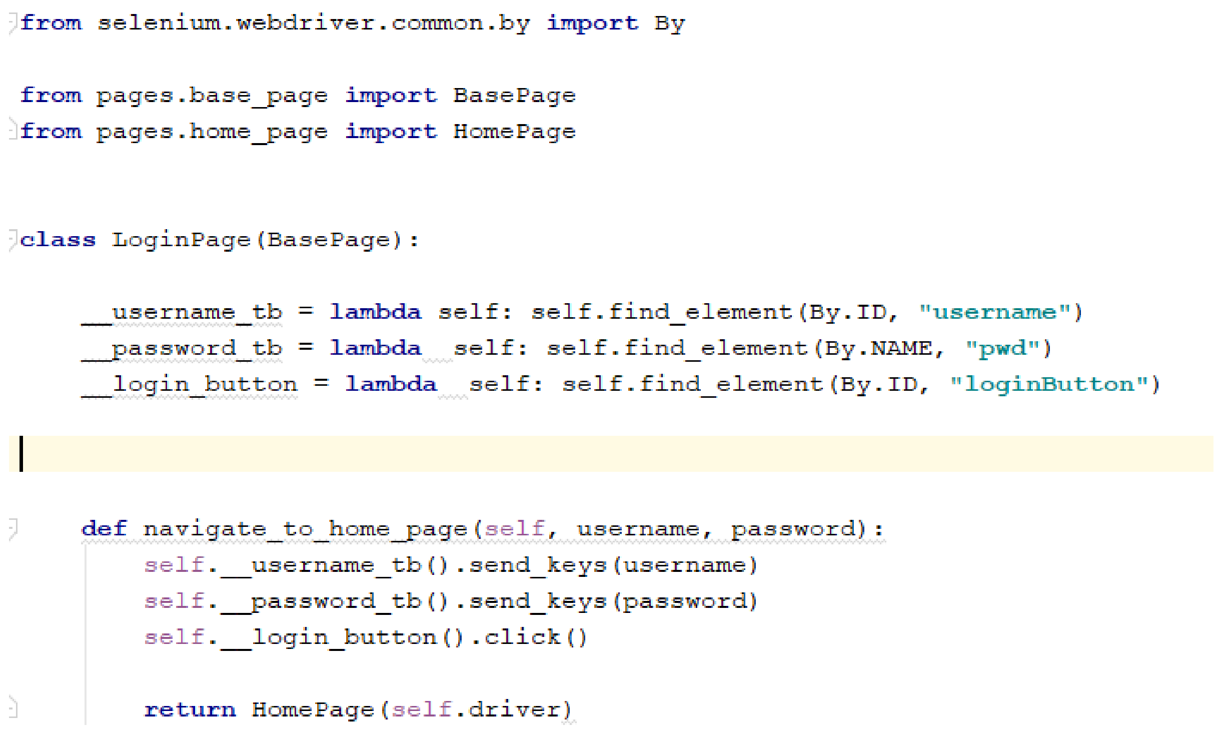
Now, will create the POM classes using the new approach,



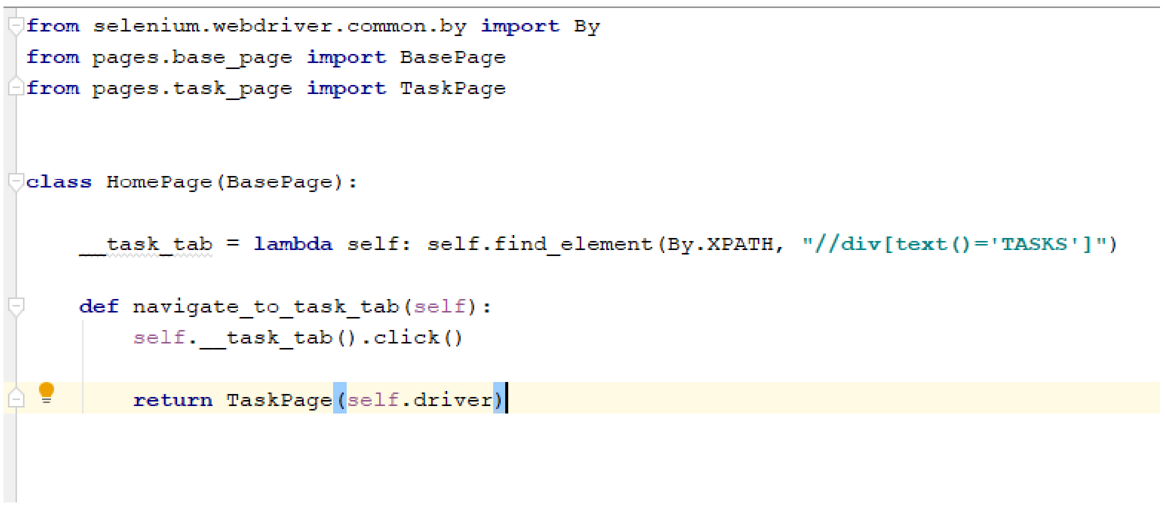
```
advance_pom D:\POM\advance_p
    pages
        __init__.py
        base_page.py
        home_page.py
        login_page.py
        task_page.py
    test_scripts
        __init__.py
        base_test.py
        login_test.py
        task_test.py
External Libraries
Scratches and Consoles
```



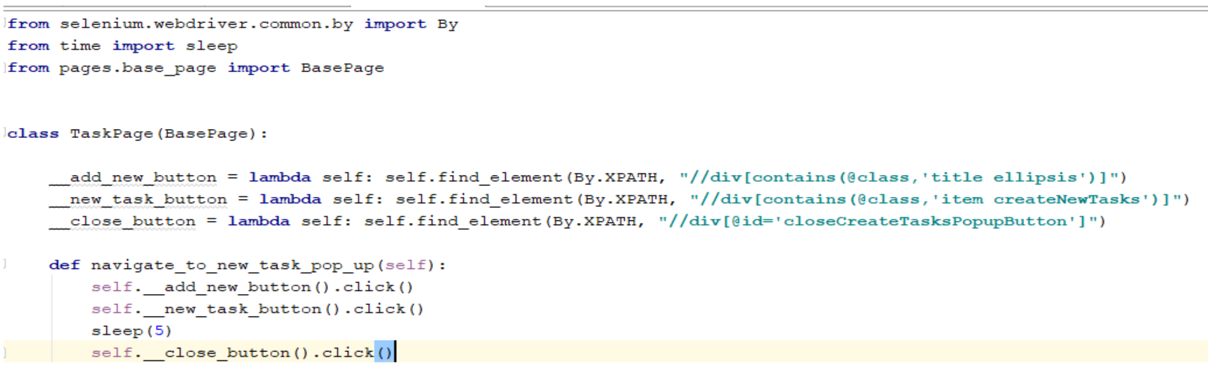
```
from selenium.common.exceptions import NoSuchElementException
from selenium.webdriver.common.by import By


class BasePage:

    def __init__(self, driver):
        self.driver = driver
        driver.set_page_load_timeout(15)

    __logout_link = lambda self: self.find_element(By.ID, "logoutLink")

    def find_element(self, locator_type, locator_value):
        try:
            element = self.driver.find_element(locator_type, locator_value)
            return element
        except NoSuchElementException:
            print("Element is not present: ", NoSuchElementException)

    def click_on_logout_link(self):
        self.__logout_link().click()
```



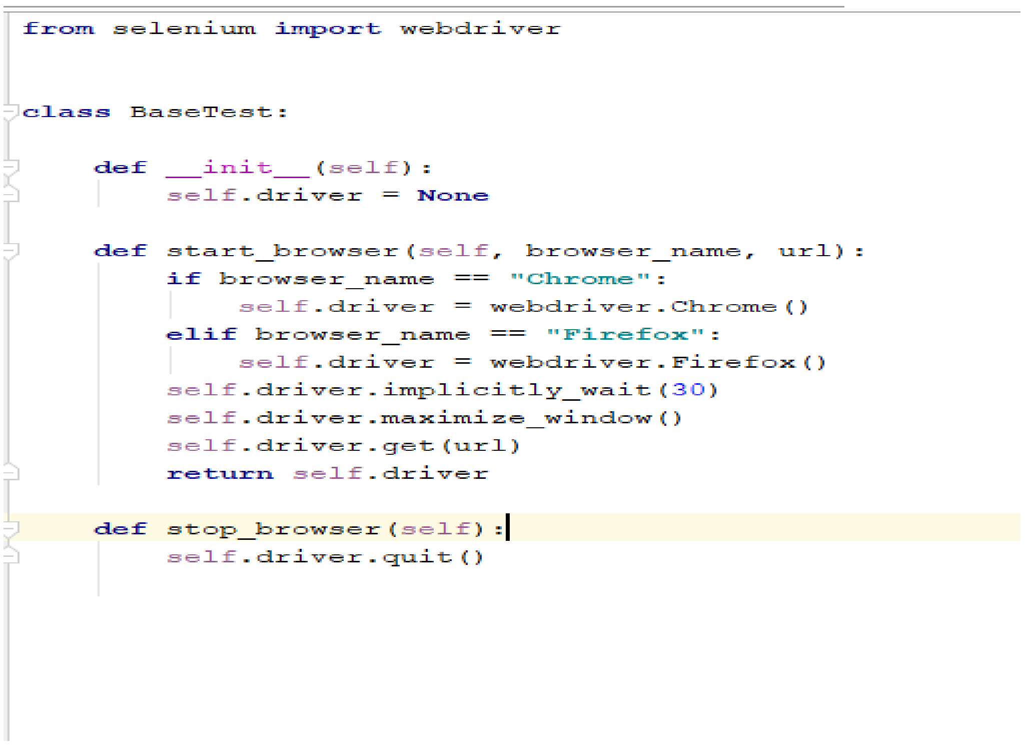
```
from selenium.webdriver.common.by import By

from pages.base_page import BasePage
from pages.home_page import HomePage


class LoginPage(BasePage):

    __username_tb = lambda self: self.find_element(By.ID, "username")
    __password_tb = lambda  self: self.find_element(By.NAME, "pwd")
    __login_button = lambda  self: self.find_element(By.ID, "loginButton")


    def navigate_to_home_page(self, username, password):
        self.__username_tb().send_keys(username)
        self.__password_tb().send_keys(password)
        self.__login_button().click()

        return HomePage(self.driver)
```



```
from selenium.webdriver.common.by import By
from pages.base_page import BasePage
from pages.task_page import TaskPage


class HomePage(BasePage):

    __task_tab = lambda self: self.find_element(By.XPATH, "//div[text()='TASKS']")

    def navigate_to_task_tab(self):
        self.__task_tab().click()

        return TaskPage(self.driver)
```

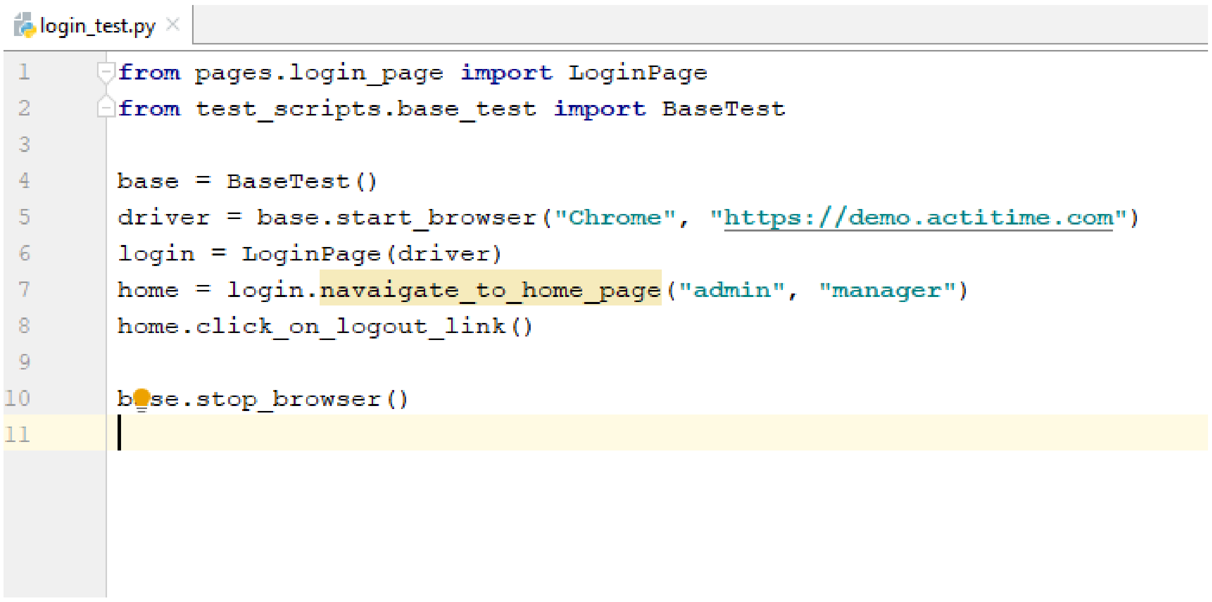


```
from selenium.webdriver.common.by import By
from time import sleep
from pages.base_page import BasePage


class TaskPage(BasePage):

    __add_new_button = lambda self: self.find_element(By.XPATH, "//div[contains(@class,'title ellipsis')]")
    __new_task_button = lambda self: self.find_element(By.XPATH, "//div[contains(@class,'item createNewTasks')]")
    __close_button = lambda self: self.find_element(By.XPATH, "//div[@id='closeCreateTasksPopupButton']")

    def navigate_to_new_task_pop_up(self):
        self.__add_new_button().click()
        self.__new_task_button().click()
        sleep(5)
        self.__close_button().click()
```

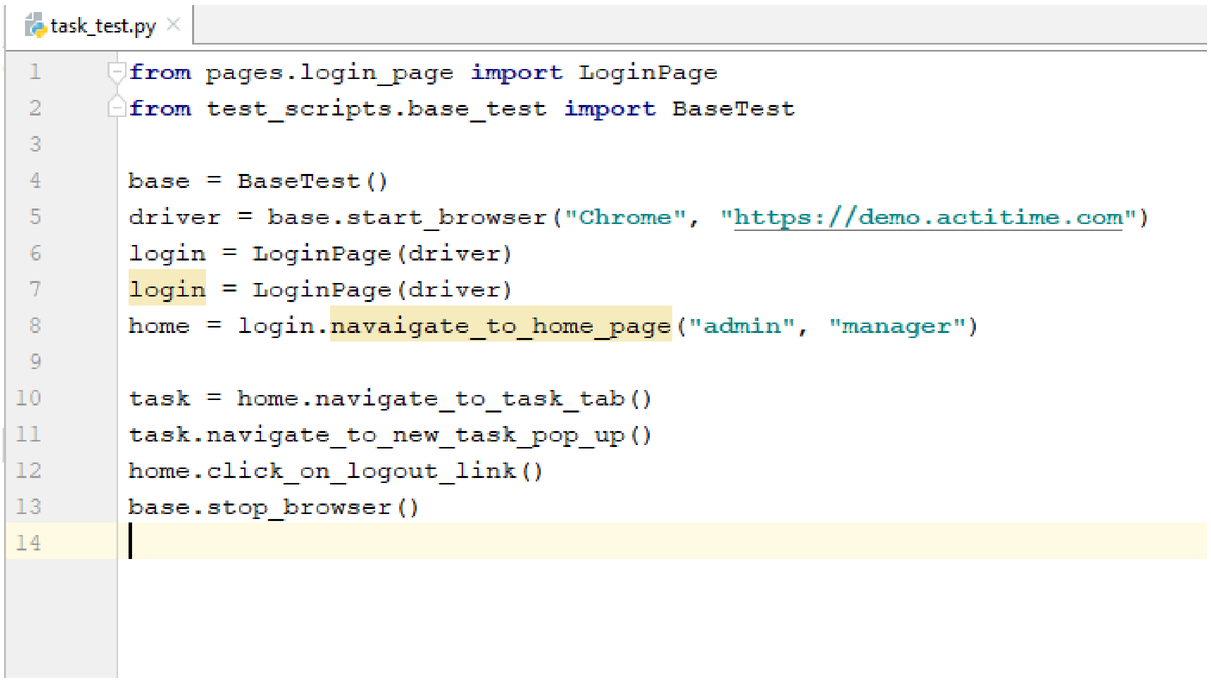


```
from selenium import webdriver


class BaseTest:

    def __init__(self):
        self.driver = None

    def start_browser(self, browser_name, url):
        if browser_name == "Chrome":
            self.driver = webdriver.Chrome()
        elif browser_name == "Firefox":
            self.driver = webdriver.Firefox()
        self.driver.implicitly_wait(30)
        self.driver.maximize_window()
        self.driver.get(url)
        return self.driver

    def stop_browser(self):
        self.driver.quit()
```

Pramod K S


login_test.py

```
from pages.login_page import LoginPage
from test_scripts.base_test import BaseTest

base = BaseTest()
driver = base.start_browser("Chrome", "https://demo.actitime.com")
login = LoginPage(driver)
home = login.navaigate_to_home_page("admin", "manager")
home.click_on_logout_link()

base.stop_browser()

```


task_test.py

```
from pages.login_page import LoginPage
from test_scripts.base_test import BaseTest

base = BaseTest()
driver = base.start_browser("Chrome", "https://demo.actitime.com")
login = LoginPage(driver)
login = LoginPage(driver)
home = login.navaigate_to_home_page("admin", "manager")

task = home.navigate_to_task_tab()
task.navigate_to_new_task_pop_up()
home.click_on_logout_link()
base.stop_browser()

```