Robotframework: Seelnium2Library with Python

Step by Step Web Automation

What is Robotframework?

- It's a generic test automation framework for **acceptance testing**
- **Acceptance test-driven development (ATDD)**
- It utilizes the **keyword-driven testing approach**
- Provides test libraries implemented either with **Python or Java**, and users **can create new higher level keywords** from existing ones using the same syntax that is used for creating test cases.
- Not much programming required

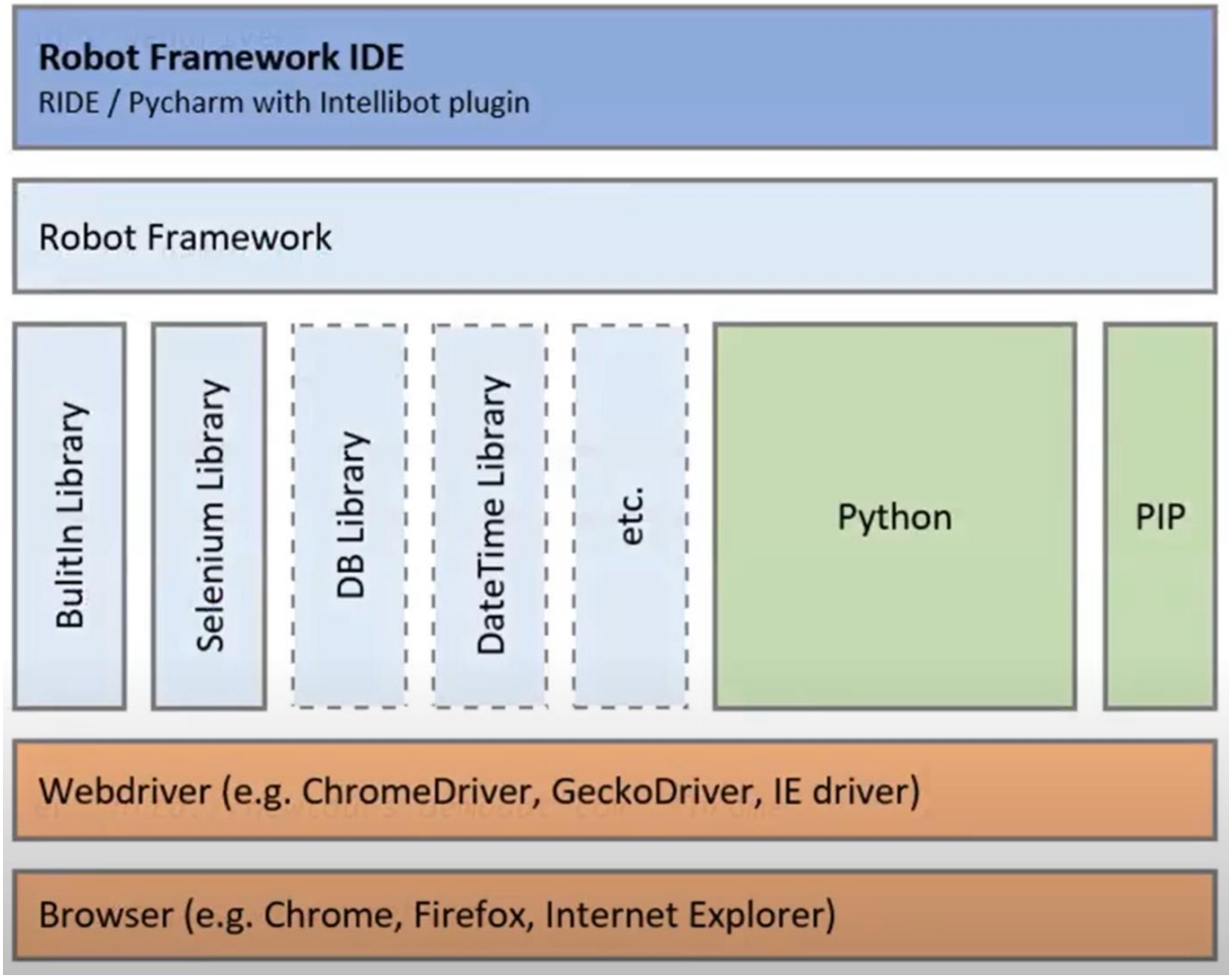
Now if you have some experience with Selenium framework irrespective of language I will show you comparison how scripts will look as you can see below first one is showing selenium script written in python without using Robotframework and its equivalent in Robotframe with using SeleniumLibrary with python

```
from selenium import webdriver

driver=webdriver.Chrome(executable_path="C:\Drivers\chromedriver_win32\chromedriver.exe")
driver.get("http://newtours.demoaut.com/")
driver.find_element_by_name("userName").send_keys("mercury")
driver.find_element_by_name("password").send_keys("mercury")
driver.find_element_by_name("login").click()
driver.close()
```

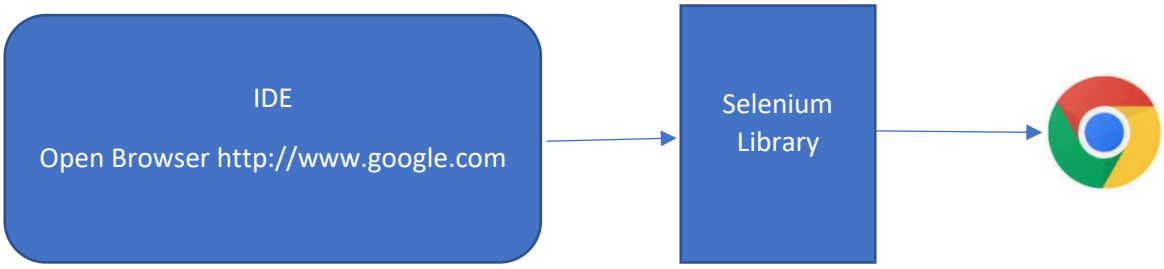

```
*** Settings ***
Library  SeleniumLibrary

*** Test Cases ***
Login Test
    Open Browser  http://newtours.demoaut.com  Chrome
    Input Text  name:userName  mercury
    Input Text  name:password  mercury
    Click Element  name:login
    Close Browser
```

You can see in Selenium-Python framework in order open browser we have to wrote command by calling function driver.get or even to set text we have to write command driver.find_element_by_name("username").sendKeys("mercury") whereas in Robotoframework we need to use simple keywords like Open Browser provide url and browser name or Input Text provide locator and text or Close Browser so its easy to read and simple keywords we will explore more in later sections

How Robotframework works?



Robotfraework contains multiple libraries like Selenium Library , DB Library etc these libraries are implemented in java and python . Once we install Robotframework we will get all these libraries and then we will add these libraries to our project.



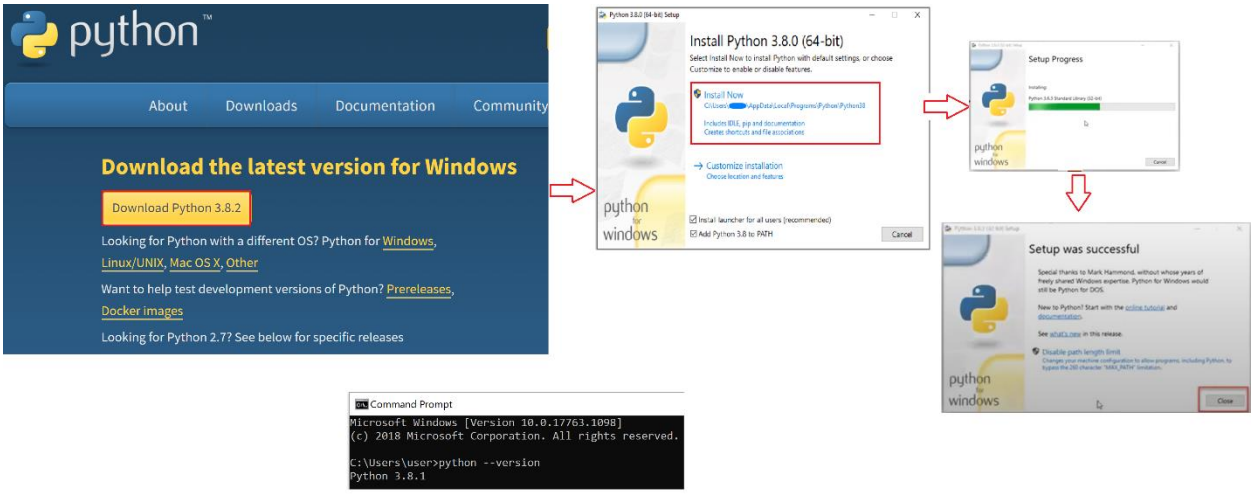
So when we run the script in IDE using kyewords it will talk to Selenium Library which has implementation of webdriver for example driver.get("URL") now this already implemented we don't have write it in script instead we use keyword Open Browser and those methose inside selenium library will talk to browser.

Now lets move towards Setup & Installations we would need to before we get started writing and executing some code:

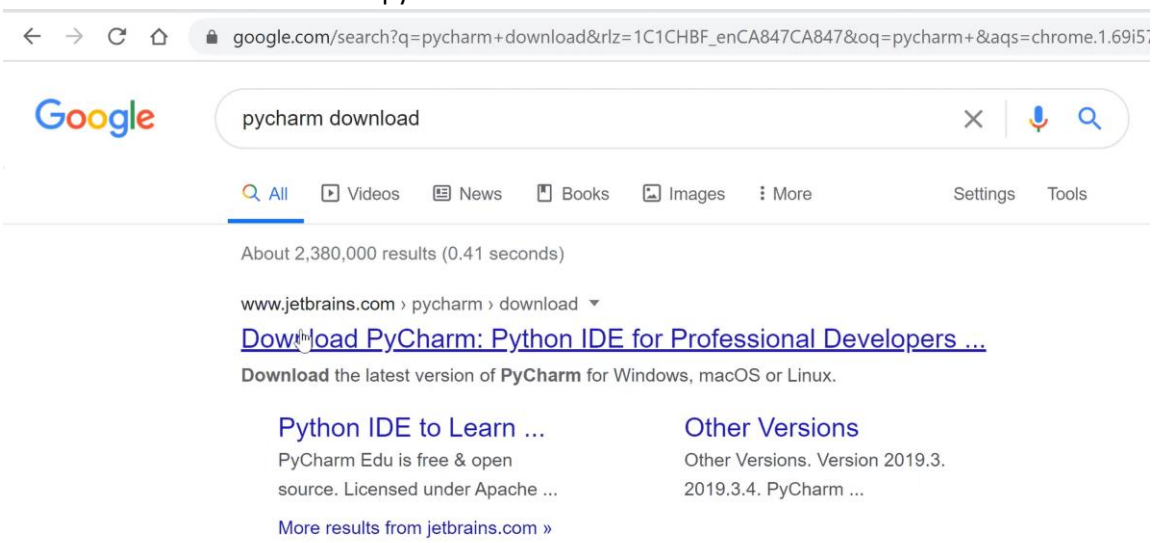
1. First, we will install python:
   Go to https://www.python.org/downloads/ . Once you click download it will download .exe file. Navigate to folder where its downloaded and run the .exe file (you don't have to be admin). Once installation wizard launchs please ensure to select

   ☑ Add Python 3.8 to PATH

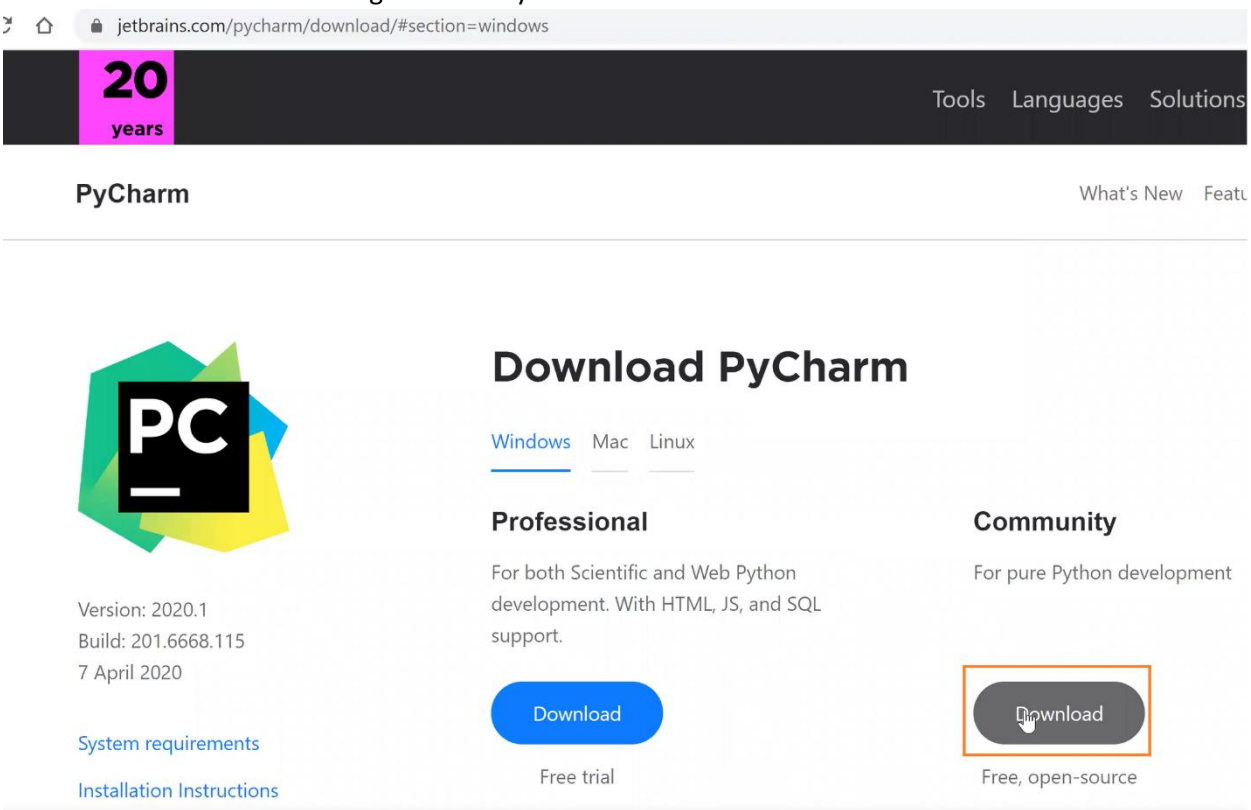
   so It directly adds python to PATH and can be accessed from command line, as you can see in below flow this option will be visible in second screenshot "Install Python 3.8.0 (64-bit)" and click Install Now. Once installation is complete you can open command prompt and type python or python --version to verify installation .



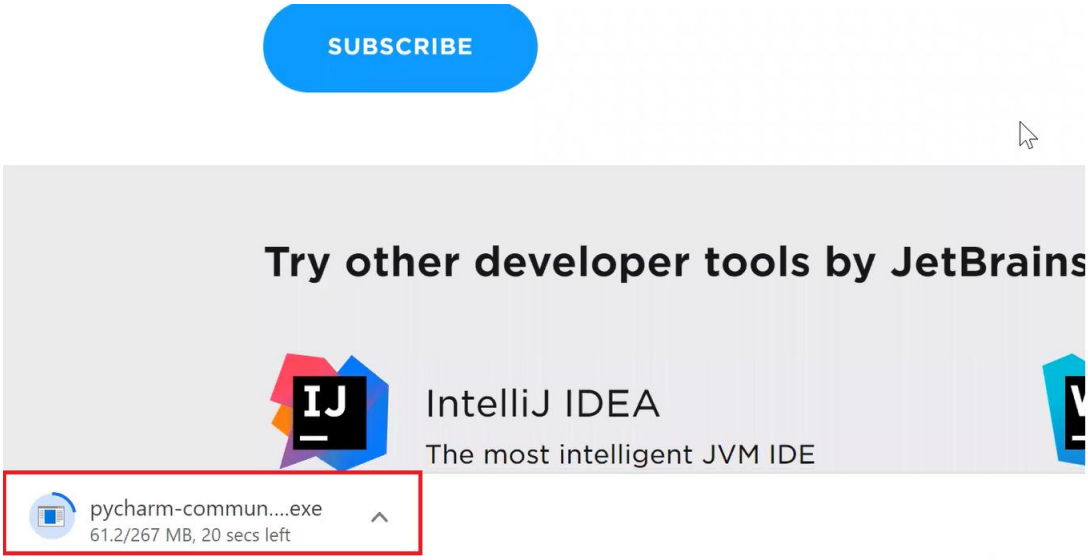
2. Second, we will install PyCharm IDE (JetBrains)
   Go to browser and search from pycharm download and click on first link.



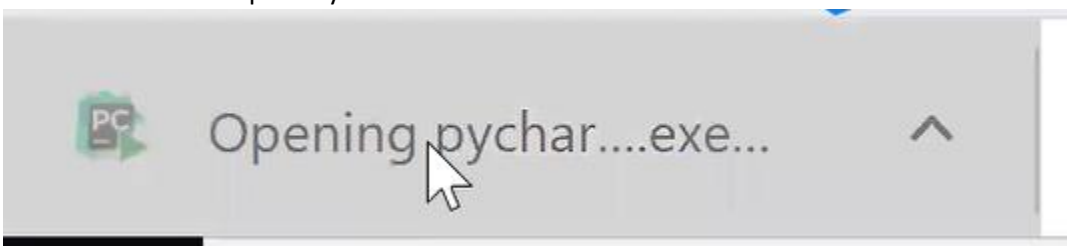
For this tutorial we will be using Community edition and click on Download.



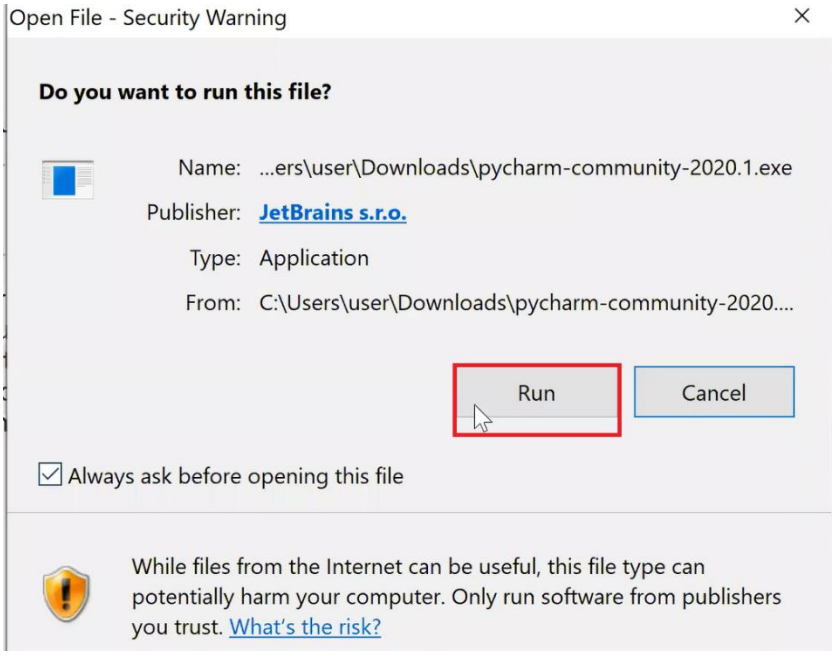
It will start downloading .exe file



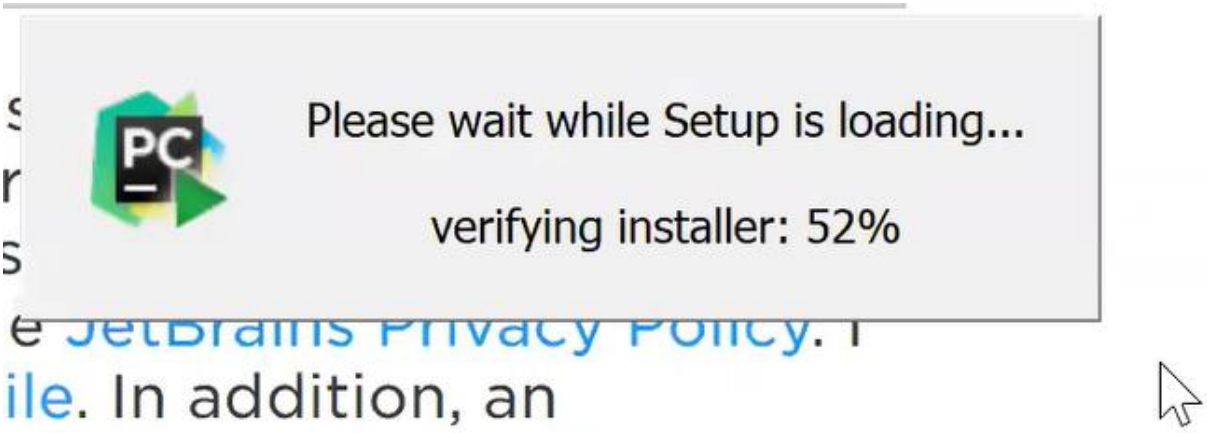
Once download completes you can double click on it



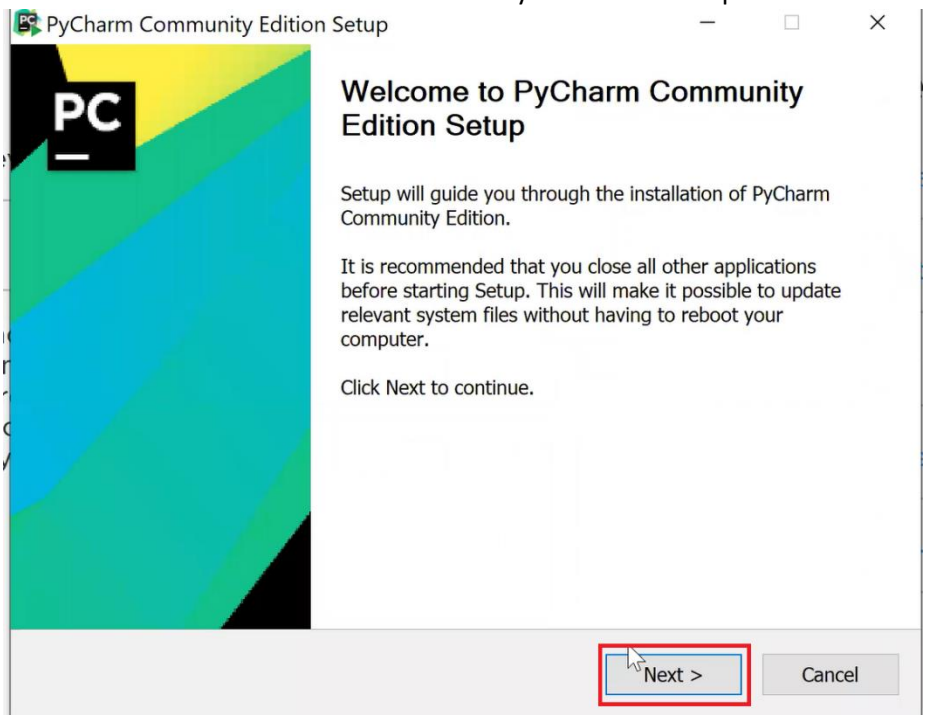
You will see prompt and click on Run



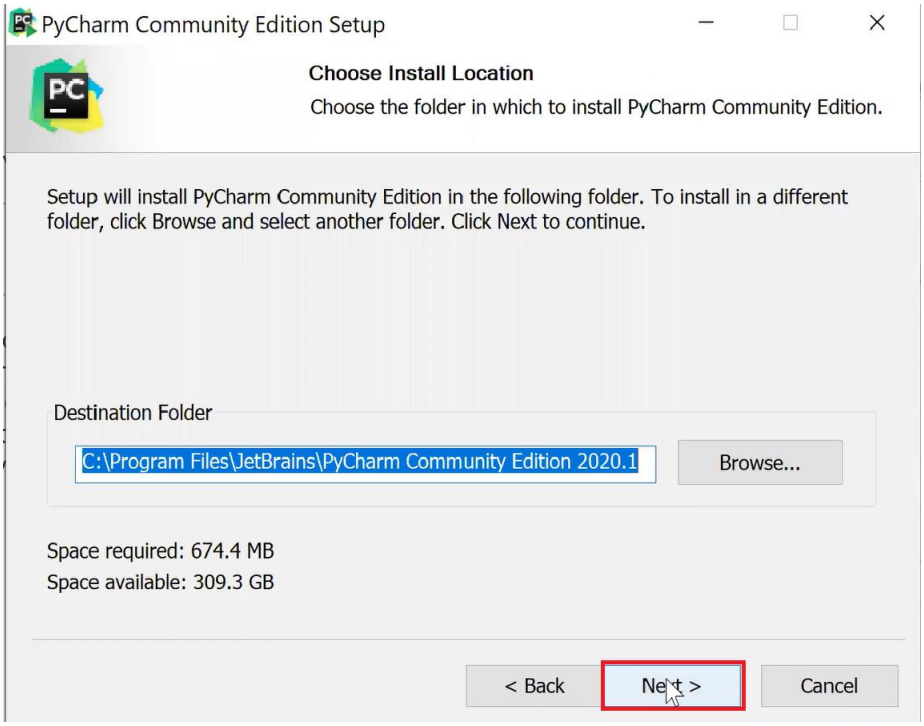
It will start verifying installer before installation



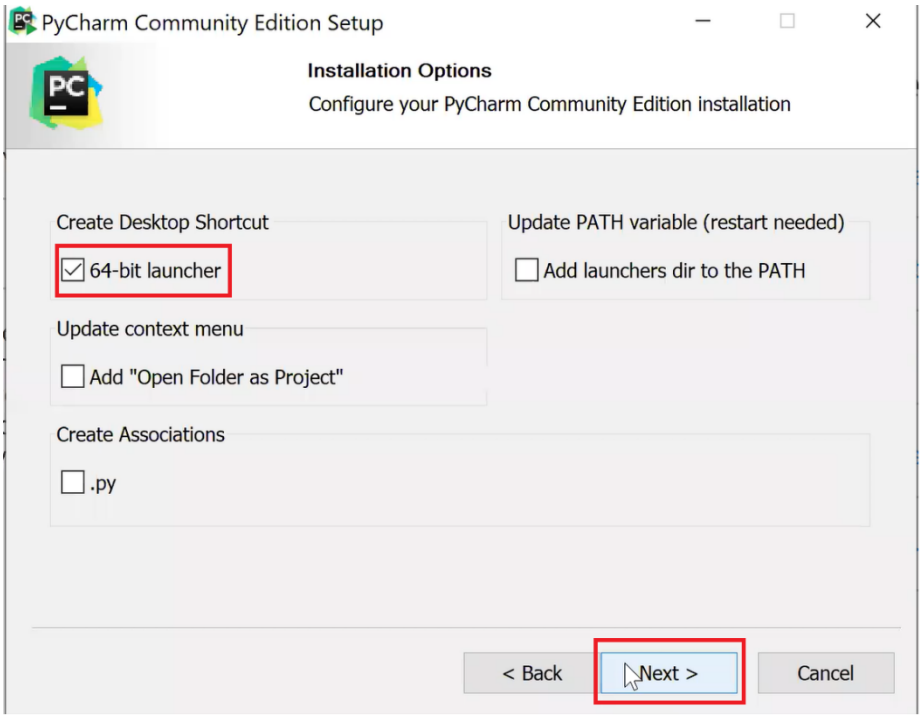
Once installer is verified it will automatically launch the Setup wizard . Click on Next button



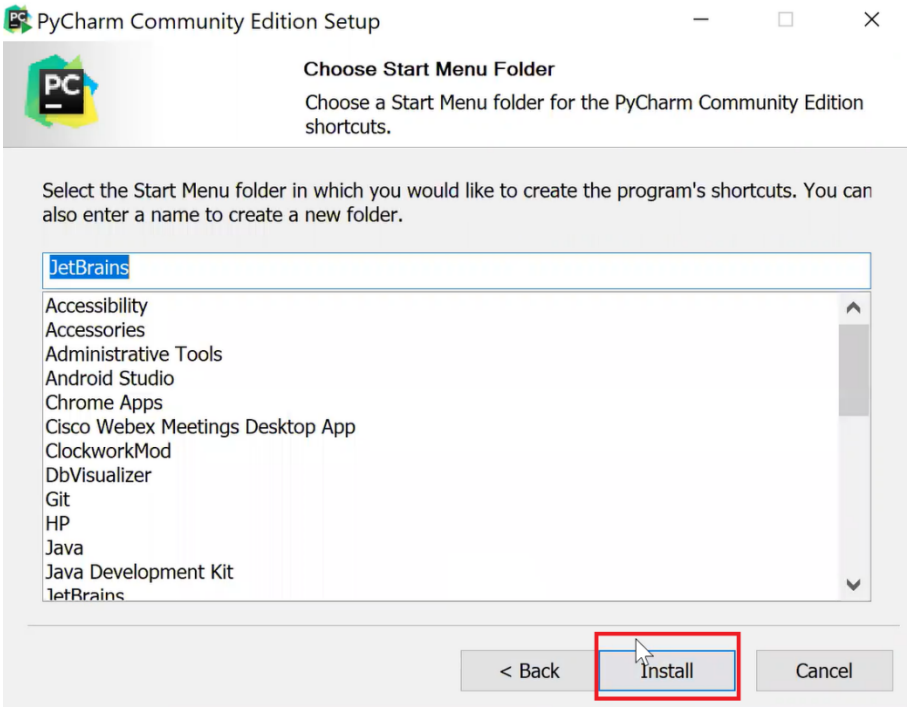
Click on Next



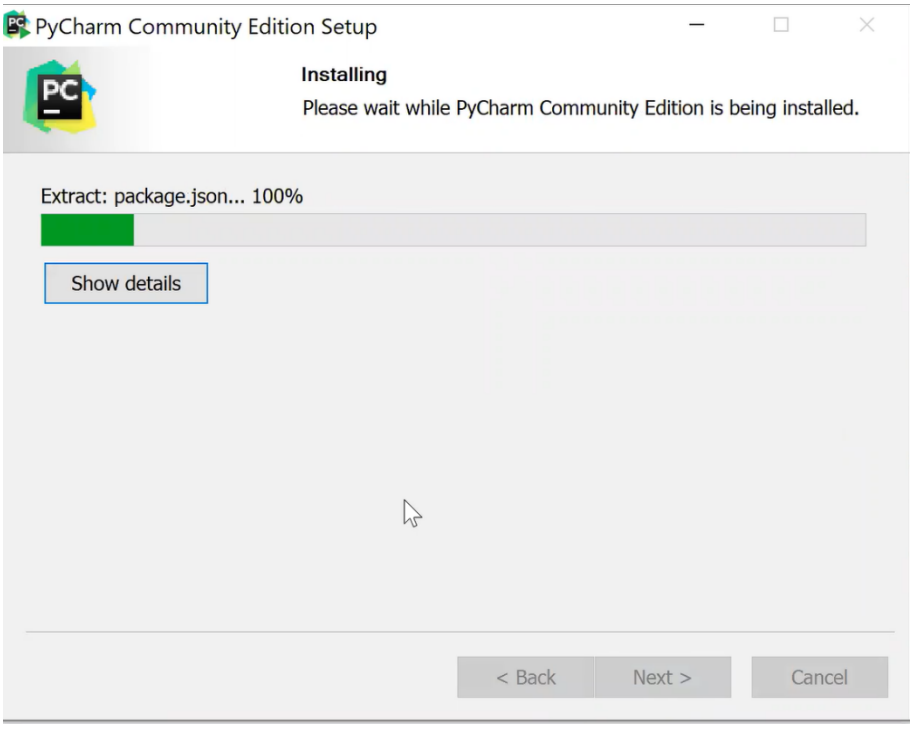
Select 64-bit launcher and click on Next button



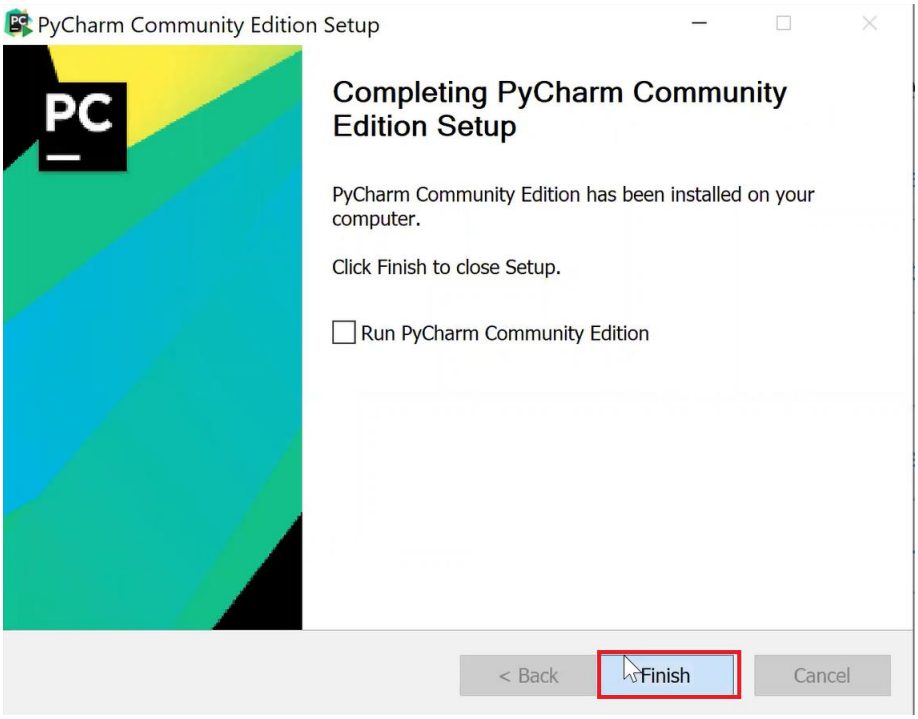
Click on install



It will start installation



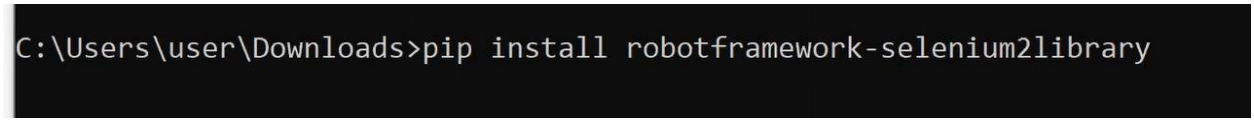
Once installation completes click Finish



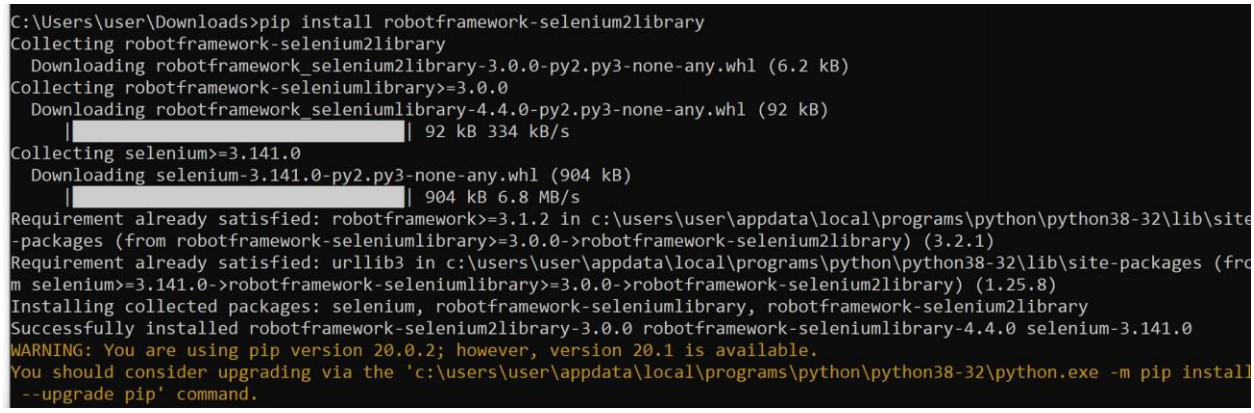
3. Third, we will install robotframework
   Open command prompt and type command pip install robotframework

   It will install the robotframework

4. Fourth, we will install robotfrrame selenium library
   In command prompt window type command pip install robotframework-selenium2library

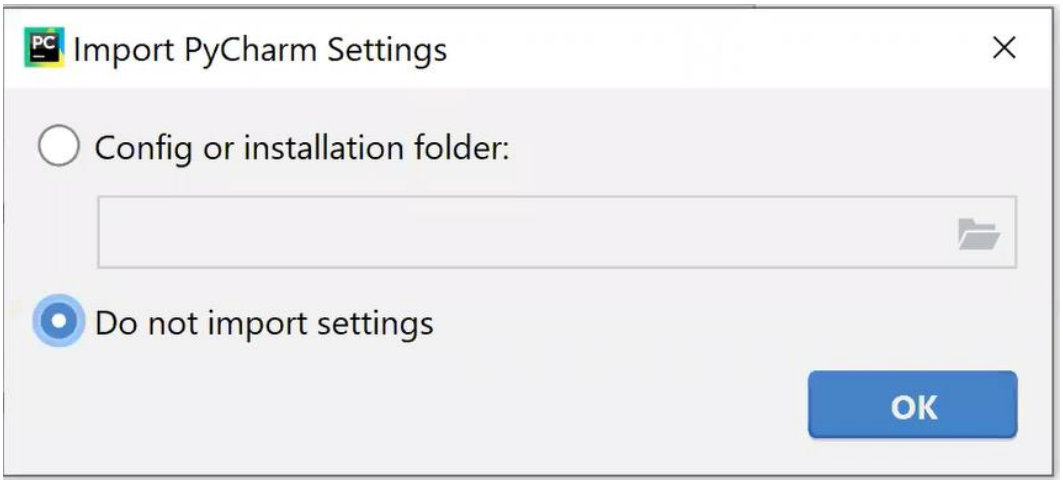


   It will install robotframework-selenium2library

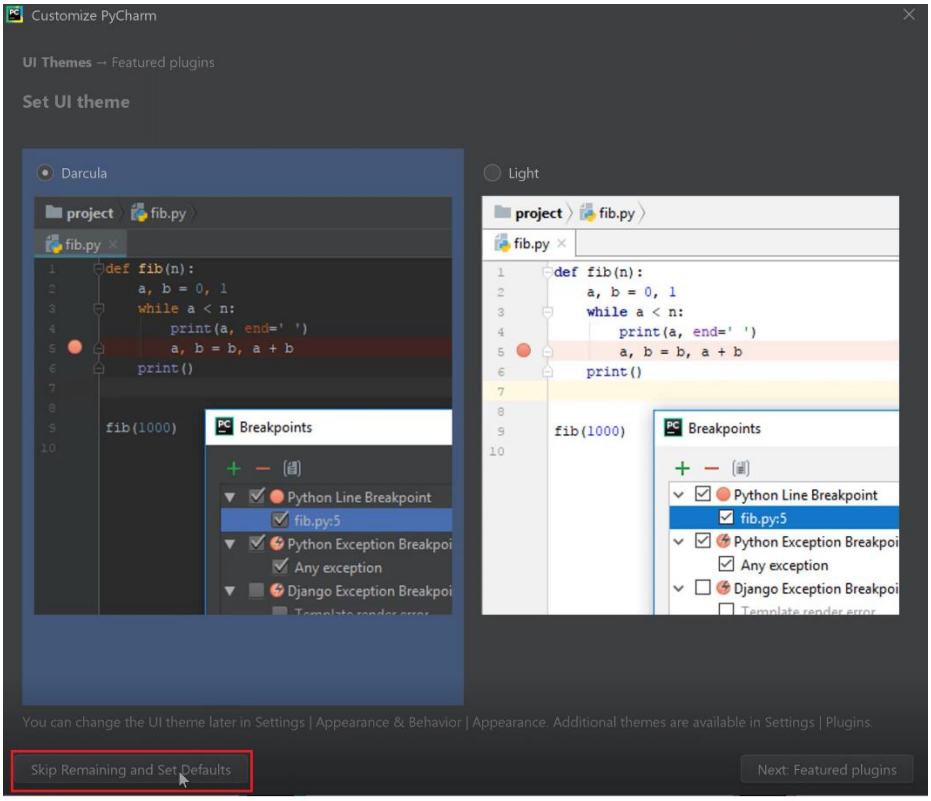


Now lets launch the Pycharm as we will be doing settings and configurations.

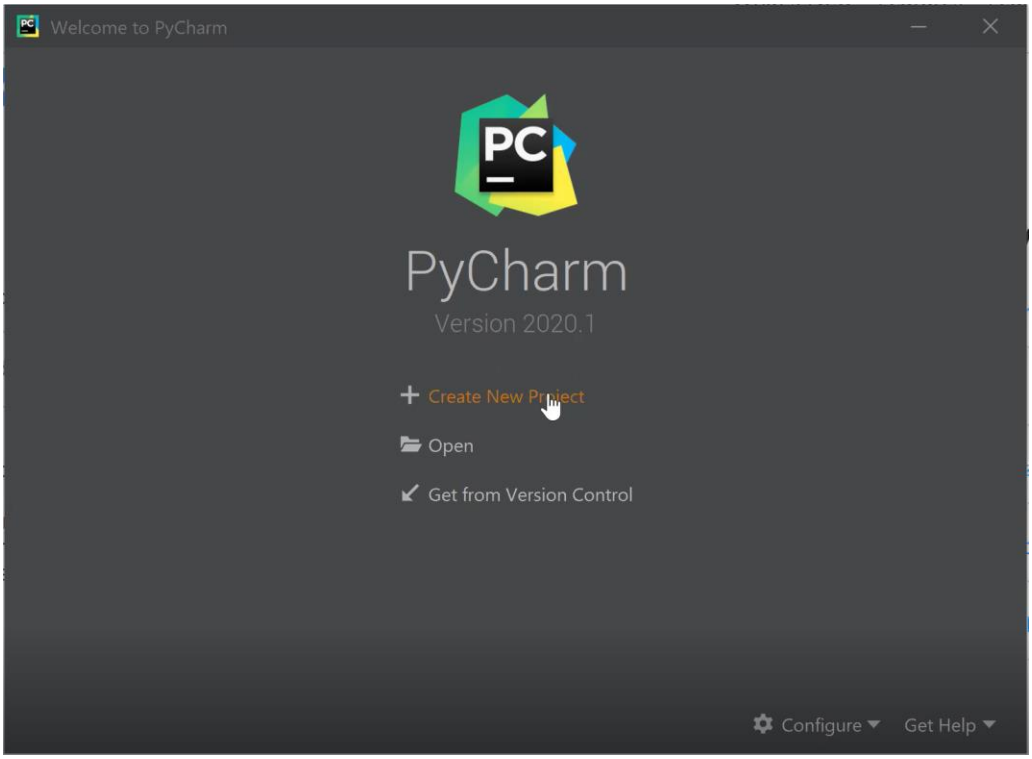
Launching first time it will prompt the following, we will be doing setting ourselves thus select “Do not import settings” and click Ok button



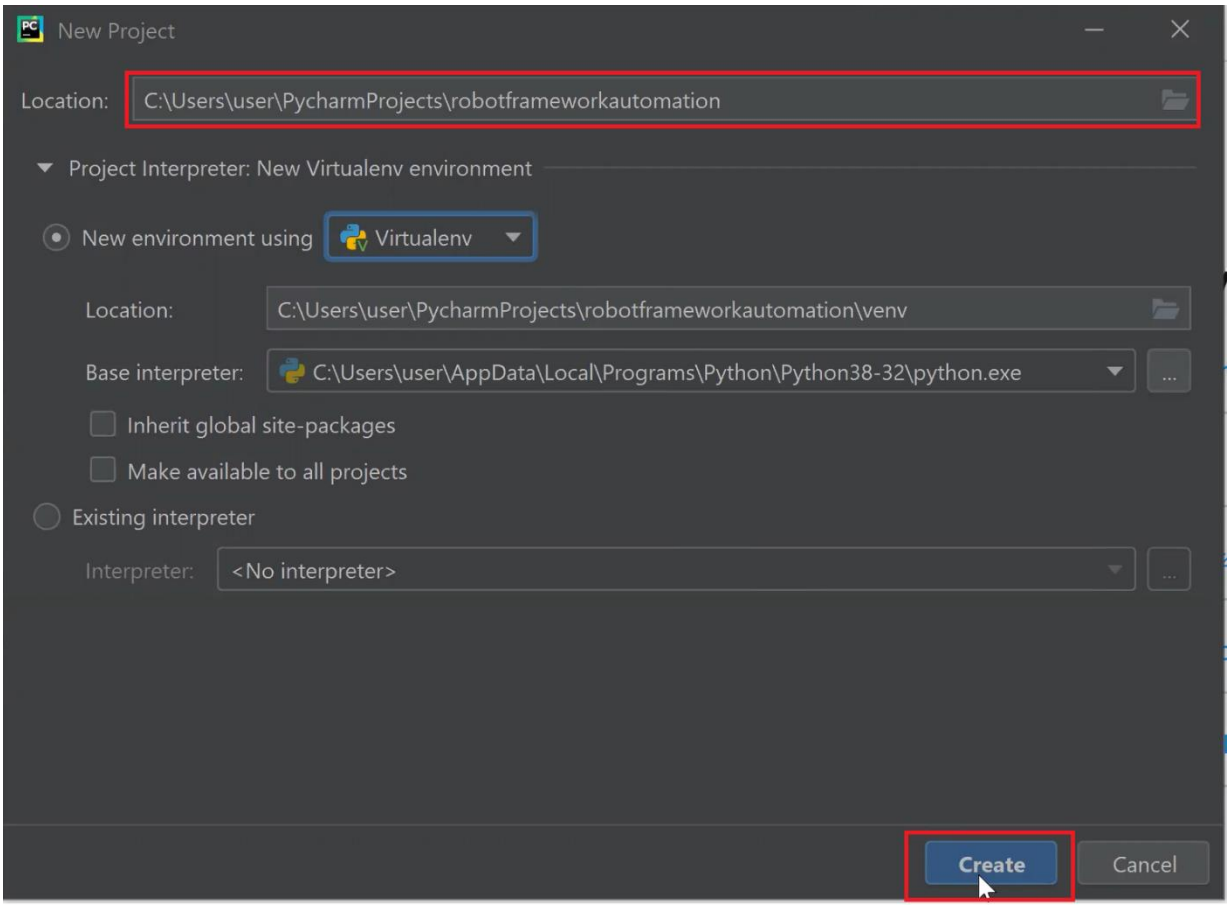
Lets use default setting click on “Skip Remaining and Set Defaults”



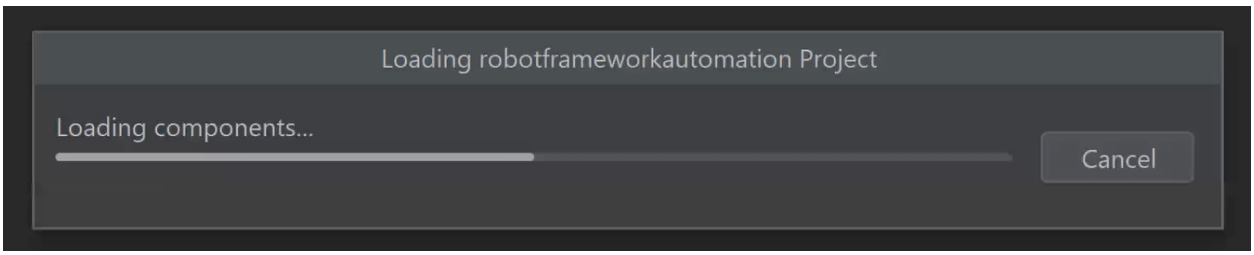
Click on Create New Project



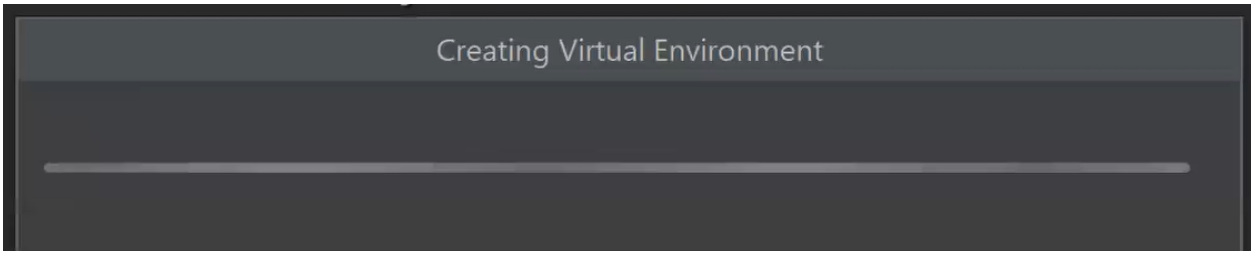
Set the project name in Location as you can see it was untitled at the end I changed it to robotframeworkautomation . Keep rest as default and click on Create



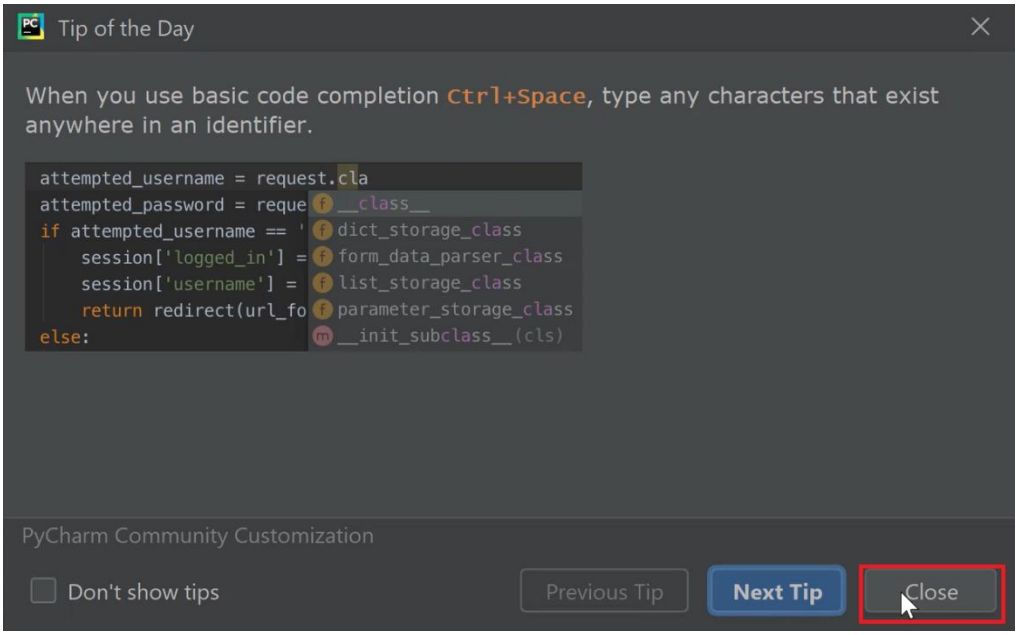
It will start downloading components



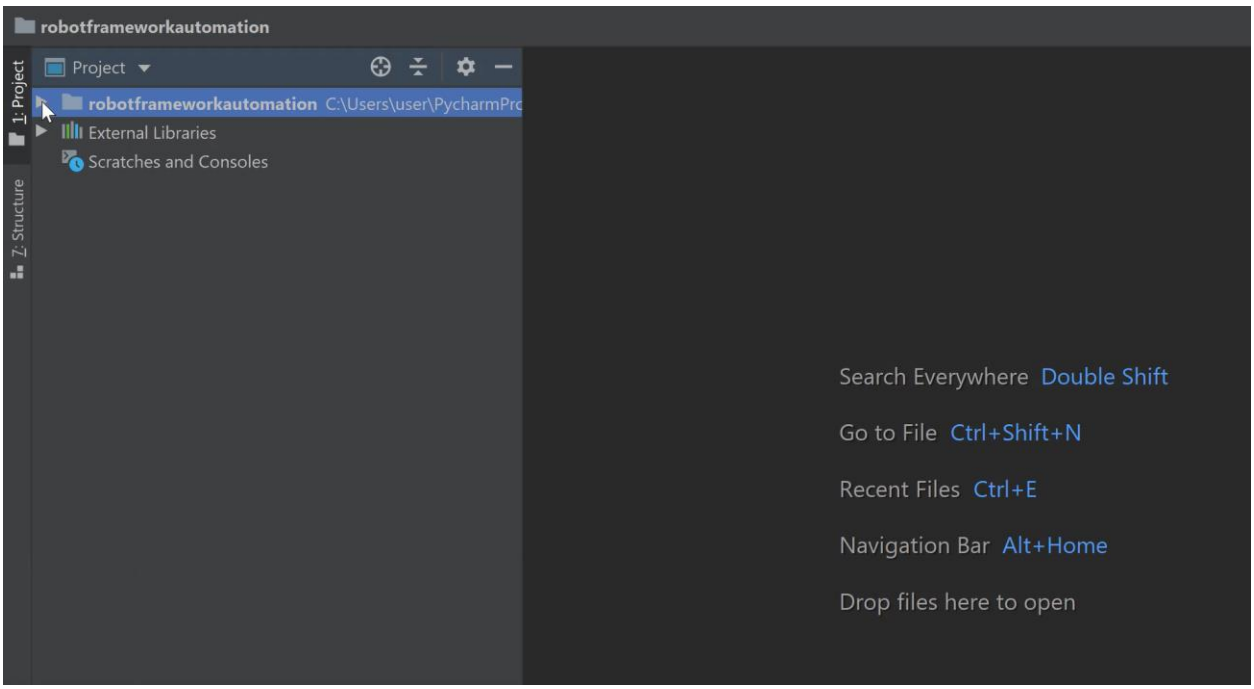
& will create virtual environment



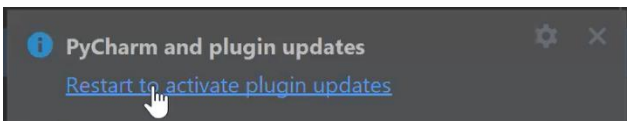
It will prompt for tips just click close button



It will launch the project

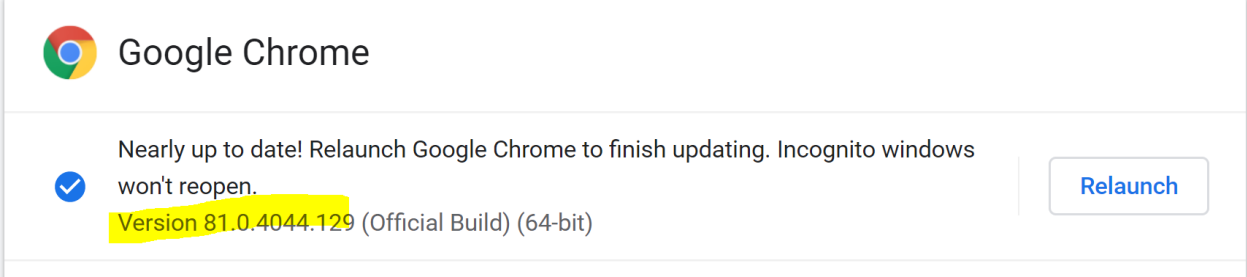


If you see the following prompt click on “Restart to active plugin updates” It will reluach Pycharm



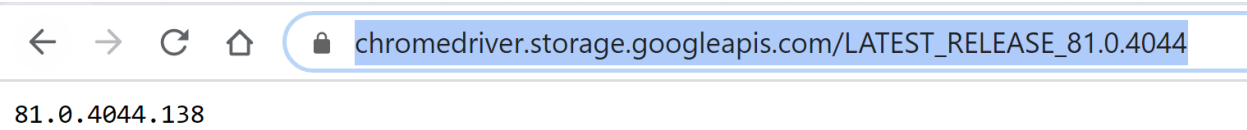
Since its browser based automation we will need to download chrome driver exe

First find out the chrome version you are using



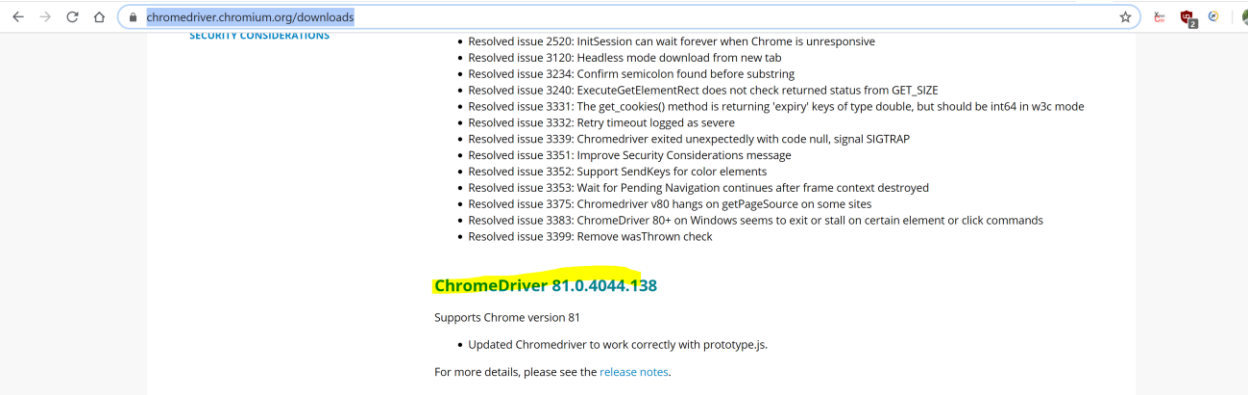
In my case version is 81.0.4044.129

Now to find exact driver version remove the last part and append it to this like this https://chromedriver.storage.googleapis.com/LATEST_RELEASE_81.0.4044 Now if run this in chrome browser I will get actual version



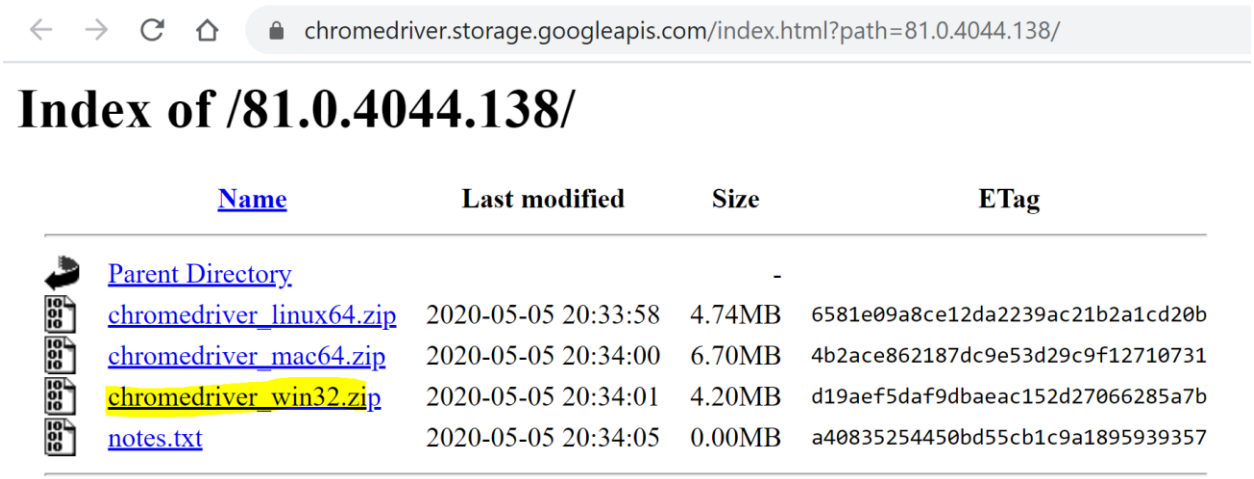
So I would need chrome driver version 81.0.4044.138 . Now navigate to https://chromedriver.chromium.org/downloads and look for chrome driver with this version

Here I found

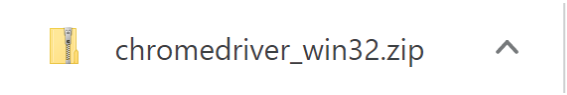


Click on the ChromeDriver 81.0.4044.138 and it will direct to file system .

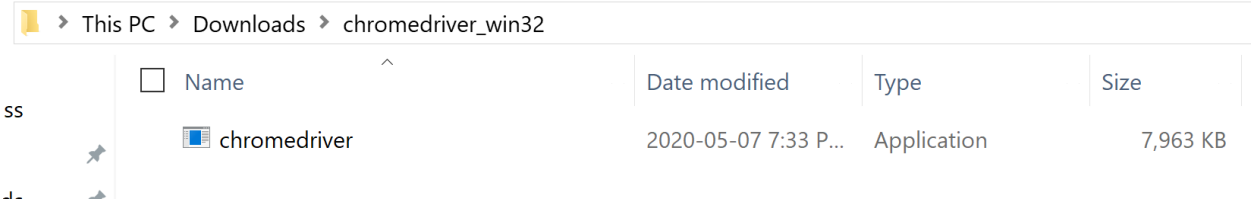
Click on chromedriver_win32.zip and it will download



Once download completes Navigate to folder and unzip it

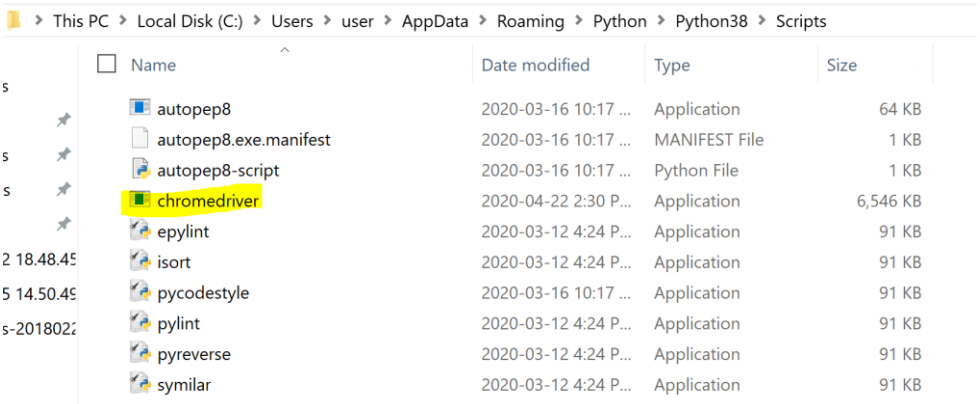


You will see the driver



We have to add this chromdriver.exe in python script directory in my system

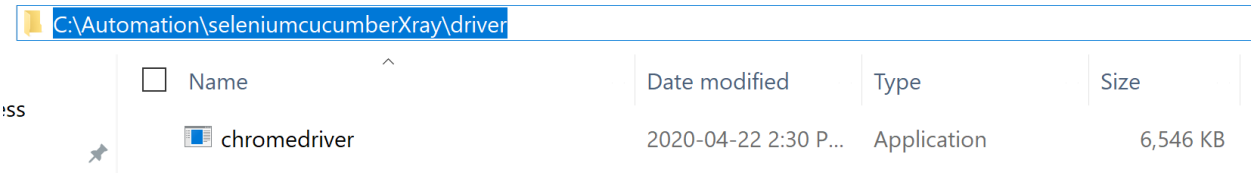
Path is C:\Users\user\AppData\Roaming\Python\Python38\Scripts



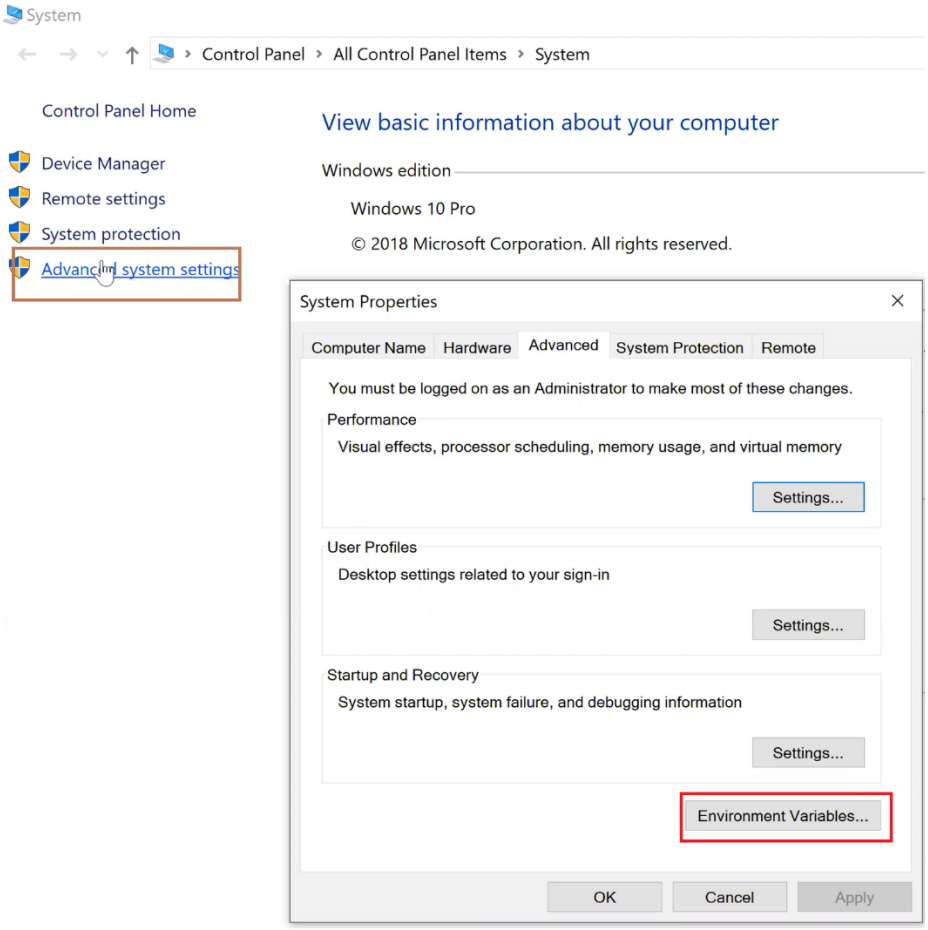
Now I will add the filepath where chromdriver.exe is in environment PATH

What I did was I created a new folder driver and copied chromedriver.exe in it

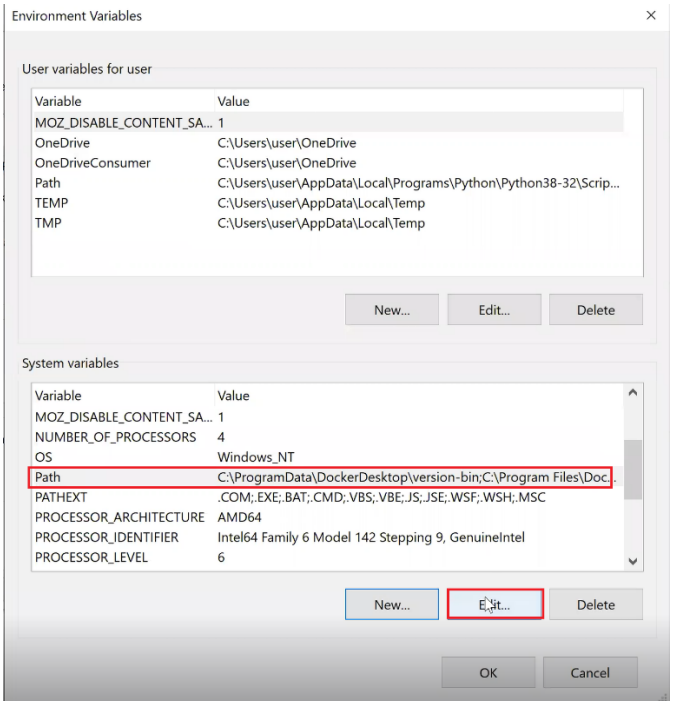
C:\Automation\seleniumcucumberXray\driver



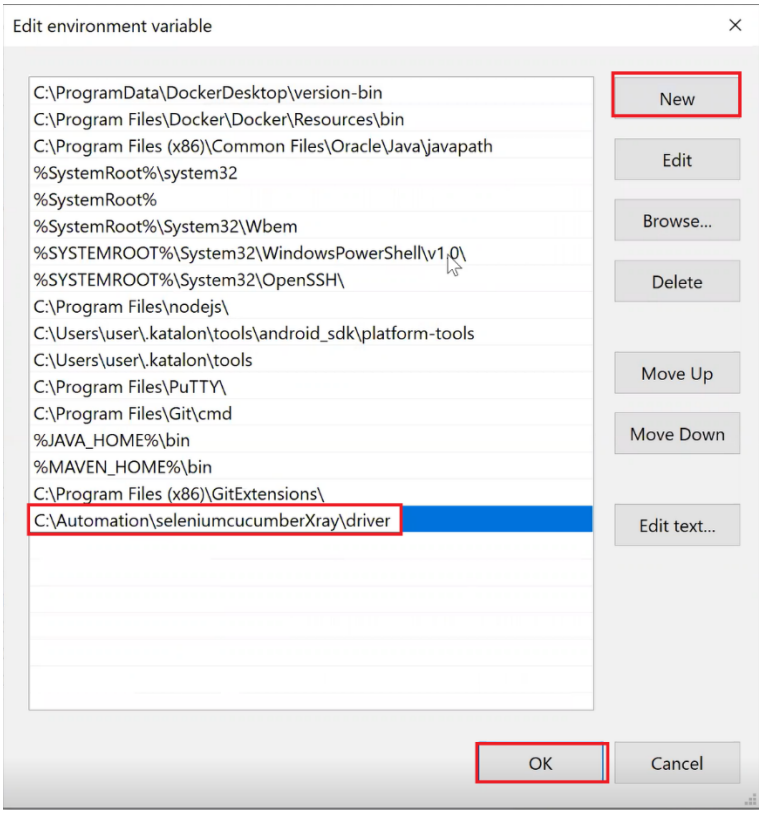
Now copy this filepath as we will add in following step to system variables PATH



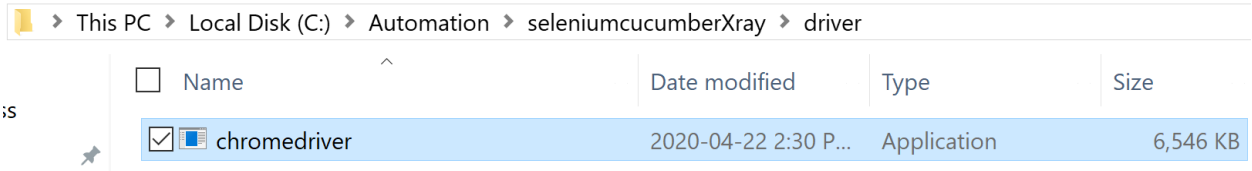
Select PATH under System variables and click Edit button



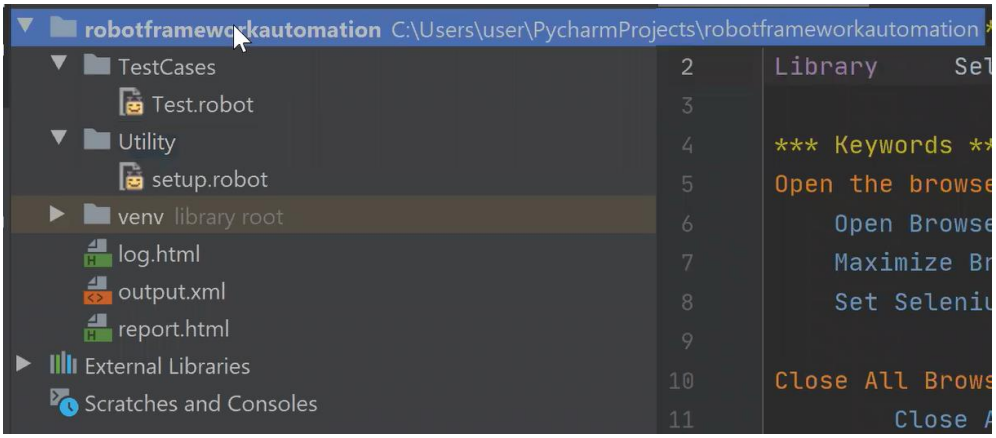
Click on New button and past the filepath to chrome driver C:\Automation\seleniumcucumberXray\driver and click ok.



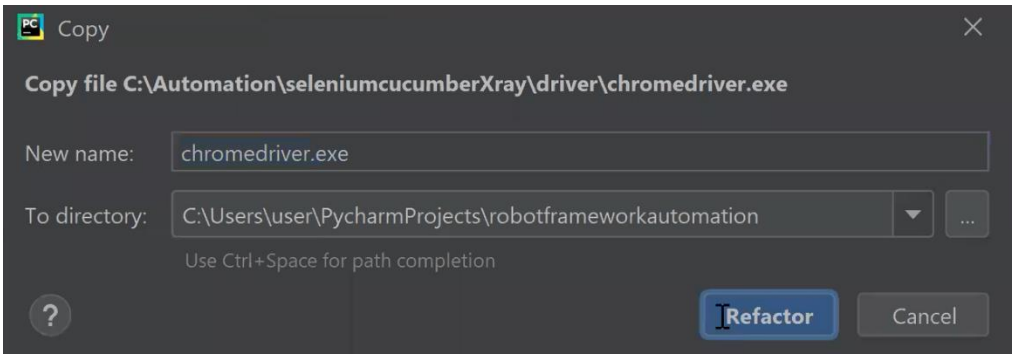
Now copy the chromedriver.exe ctrl+c



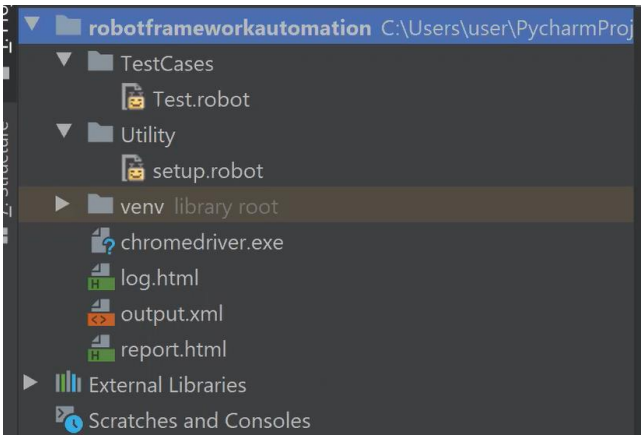
Navigate to Pycharm and paste it ctrl+v at project level



Click on Refactor button

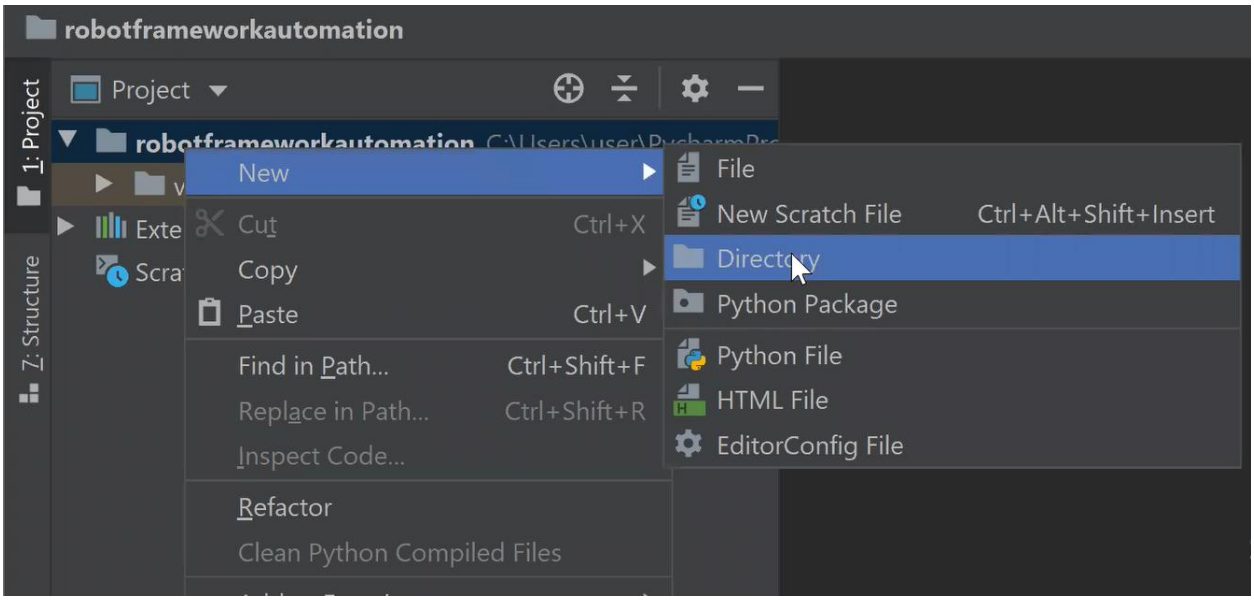


You will see chromedriver.exe

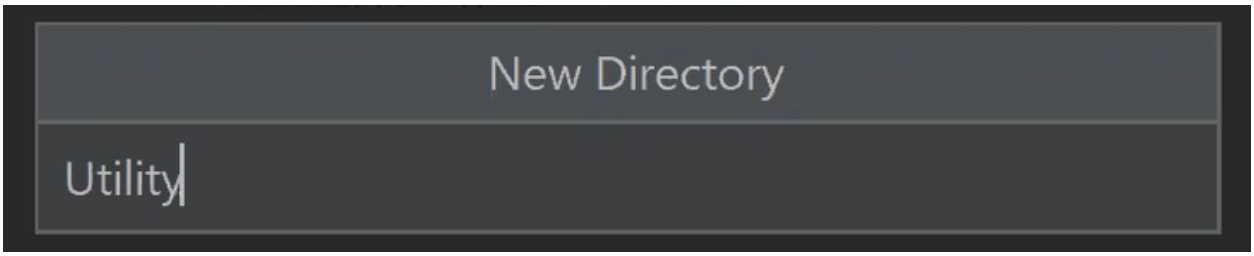


Now lets get started with writing first basic test case. During the creation of this first simple test case we will do framework setup and as we progress we will update setup along with it.

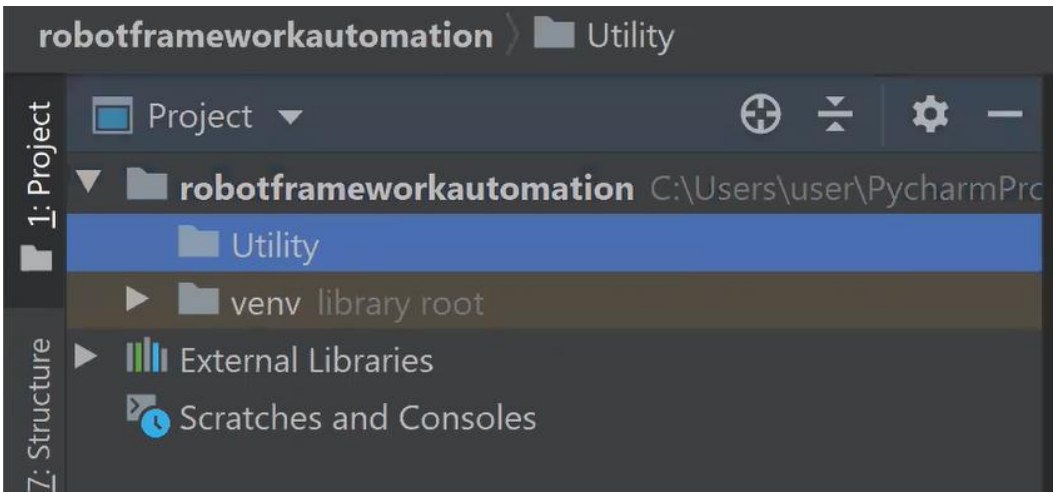
Lets create basic structure, right click project->New->Directory



Name the directory Utility

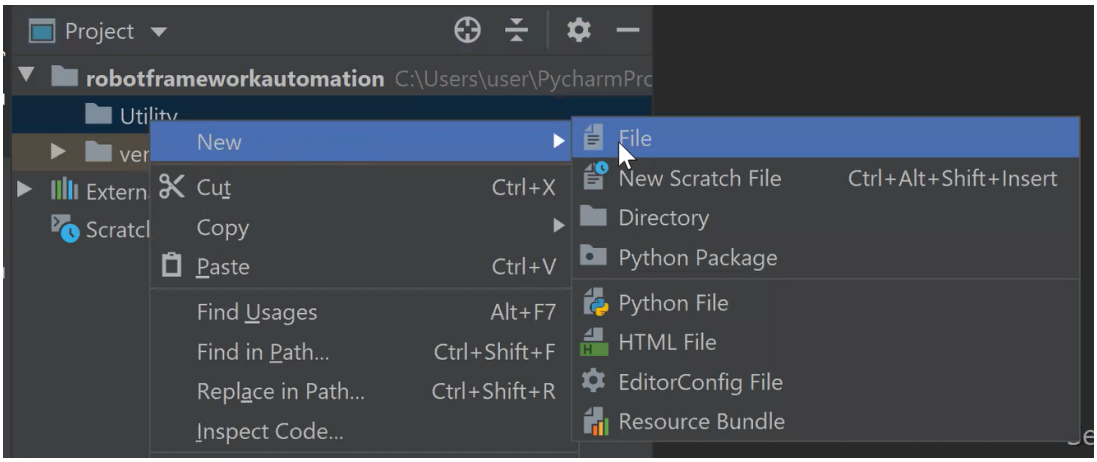


You will see it in Project Explorer

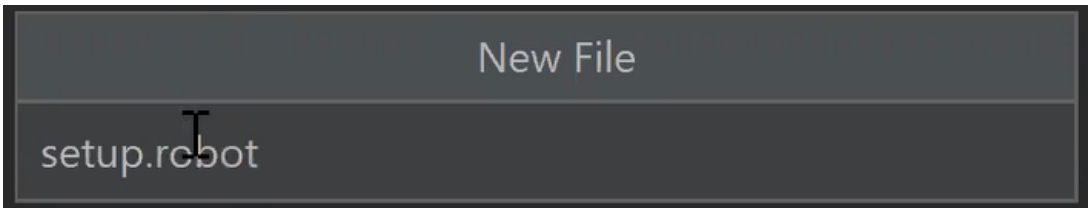


In Utility folder we will keep our test setup file and selenium keywords file, so we can re-use them in our test classes.

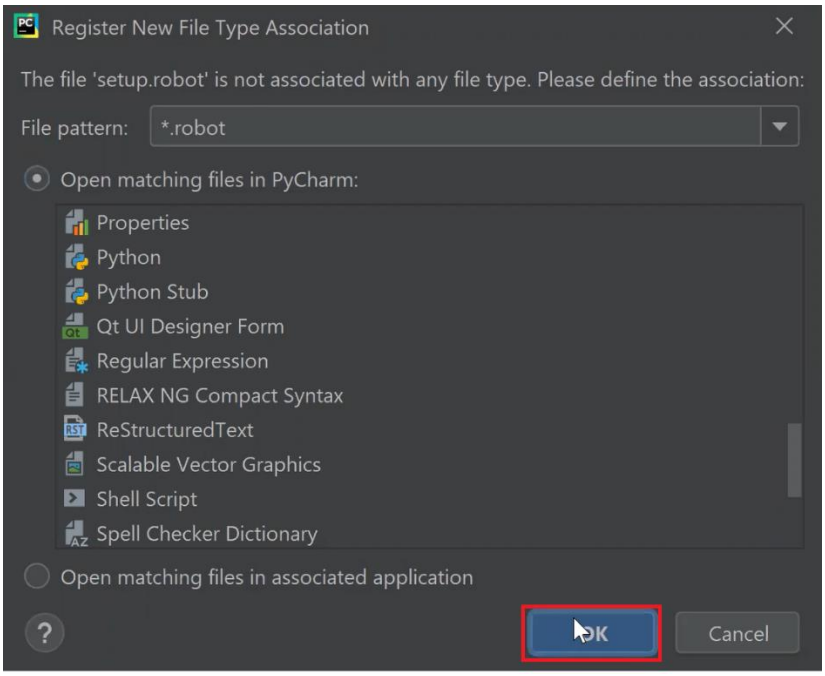
First, we will create file Setup.robot right click Utility folder ->New->File



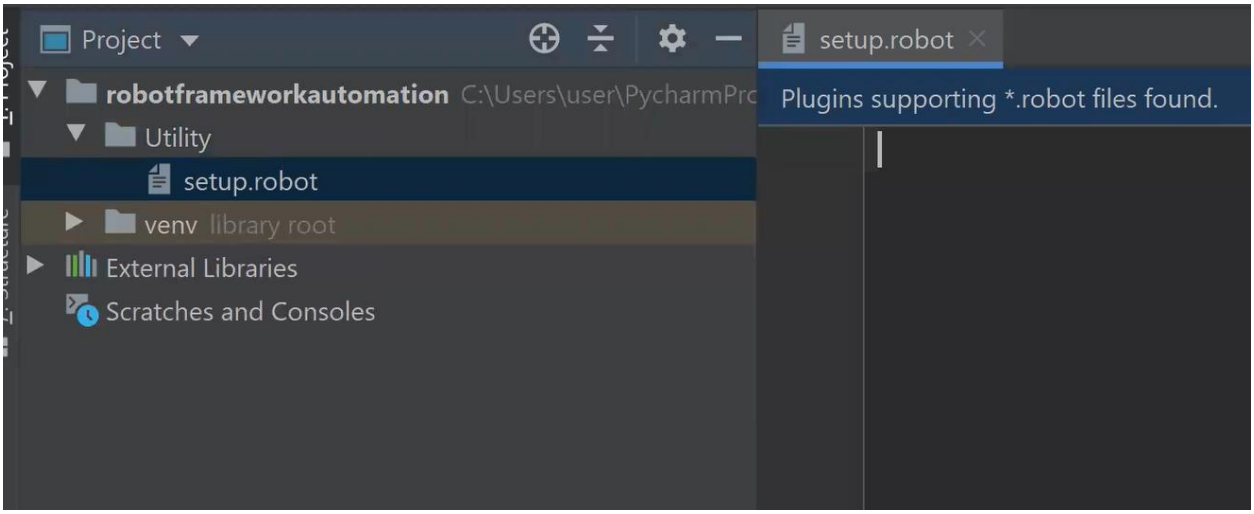
Set name as setup.robot



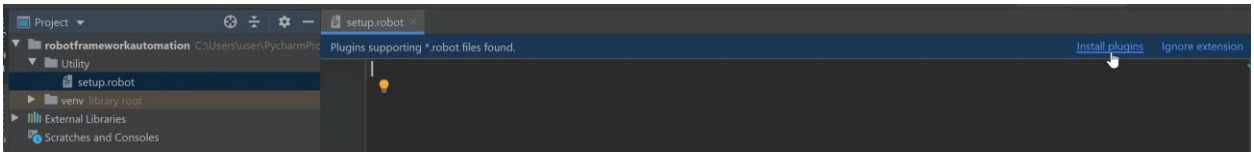
Click Ok in next prompt



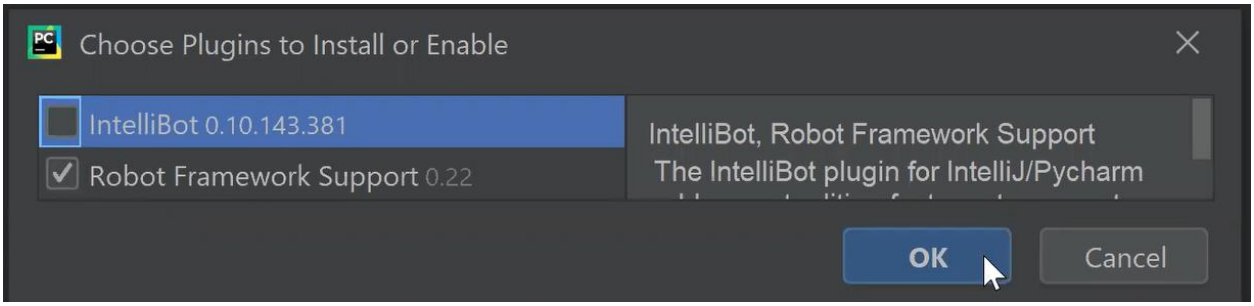
You will see the file created



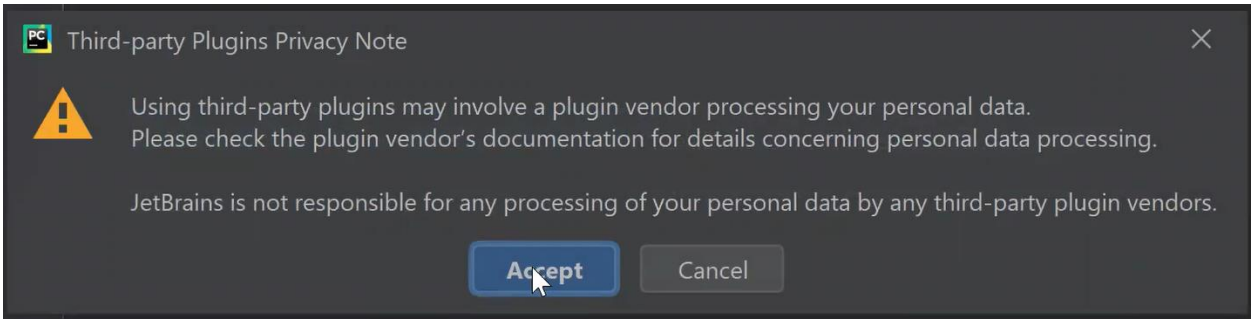
You will notice the prompt for install plugins click on it



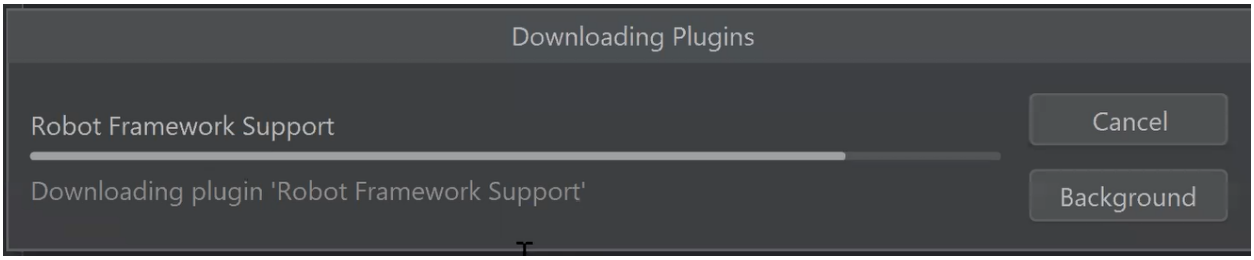
Uncheck IntelliBot and click OK



Click Accept



It will start installing plugins



Lets start writing script for setup.robot.

There are different sections first is Settings

```
*** Settings ***
```

This section is used for:

1) Importing **test libraries**, **resource files** and **variable files**.

2) Defining metadata for **test suites** and **test cases**.

For now, we will import selenium2library

```
*** Settings ***
Library     Selenium2Library
```

Next we will add Keywords section

```
*** Keywords ***
```

This section is used for:
Creating user keywords from existing lower-level keywords

For now we will add simple keywords

```
*** Keywords ***
Open the browser in
   Open Browser   "https://demo.nopcommerce.com/" Chrome
   Maximize Browser Window
   Set Selenium Timeout   10s

Close All Browser Window
      Close All Browsers
```

Here you can see we are you using Open Browser it takes parameter URL and Browser to launch browser and navigate to that url

You can find library of keywords here
http://robotframework.org/Selenium2Library/Selenium2Library.html

Keywords

Keyword	Arguments	Documentation
Open Browser	*url, browser=firefox, alias=None, remote_url=False, desired_capabilities=None, ff_profile_dir=None*	Opens a new browser instance to the given `url`. The `browser` argument specifies which browser to use, and the supported browser are listed in the table below. The browser names are case-insensitive and some browsers have multiple supported names. **Browser** / **Name(s)**: Firefox: firefox, ff; Google Chrome: googlechrome, chrome, gc; Internet Explorer: internetexplorer, ie; Edge: edge; Safari: safari; Opera: opera; Android: android; Iphone: iphone; PhantomJS: phantomjs; HTMLUnit: htmlunit; HTMLUnit with Javascript: htmlunitwithjs To be able to actually use one of these browsers, you need to have a matching Selenium browser driver available. See the project documentation for more details. Optional `alias` is an alias given for this browser instance and it can be used for switching between browsers. An alternative approach for switching is using an index returned by this keyword. These indices start from 1, are incremented when new browsers are opened, and reset back to 1 when *Close All Browsers* is called. See *Switch Browser* for more information and examples. Optional `remote_url` is the URL for a remote Selenium server. If you specify a value for a remote, you can also specify `desired_capabilities` to configure, for example, a proxy server for Internet Explorer or a browser and operating system when using Sauce Labs. Desired capabilities can be given either as a Python dictionary or as a string in format `key1:value1,key2:value2`. Selenium documentation lists possible capabilities that can be enabled. Optional `ff_profile_dir` is the path to the Firefox profile directory if you wish to overwrite the default profile Selenium uses. Notice that prior to Selenium2Library 3.0, the library contained its own profile that was used by default. Examples: *Open Browser* http://example.com Chrome *Open Browser* http://example.com Firefox alias=Firefox *Open Browser* http://example.com Edge remote_url=http://127.0.0.1:4444/wd/hub

The other keyword we have used is Maximize Browser Window which Maximizes current browser window.

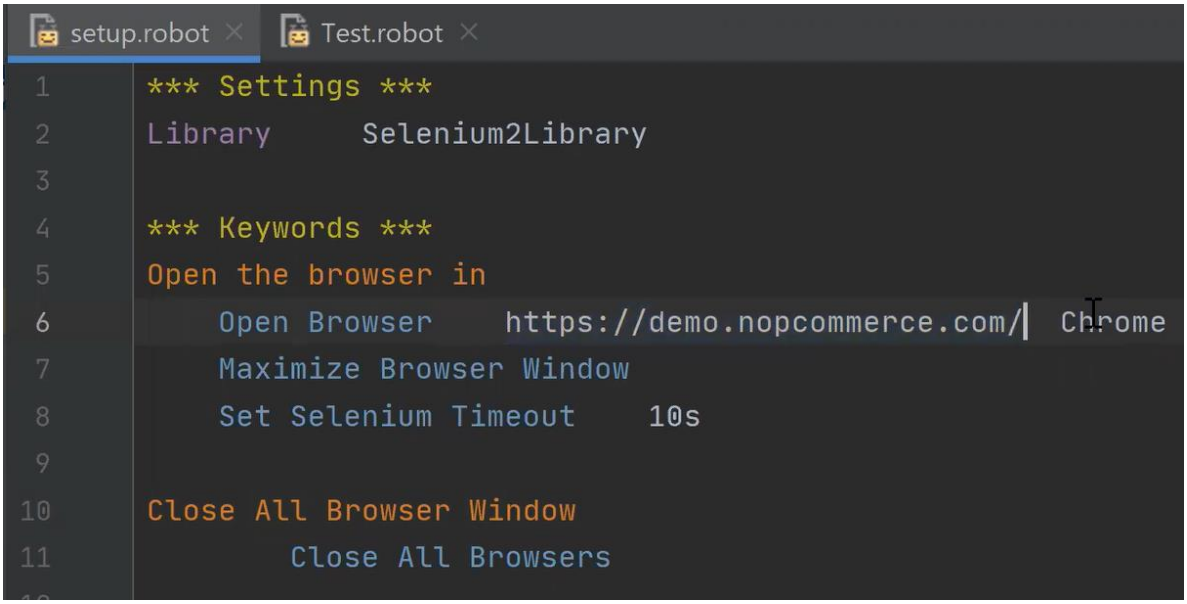
Maximize Browser Window		Maximizes current browser window.

You can see it doesn't take any parameters.

Next you will see keyword Close All Browsers

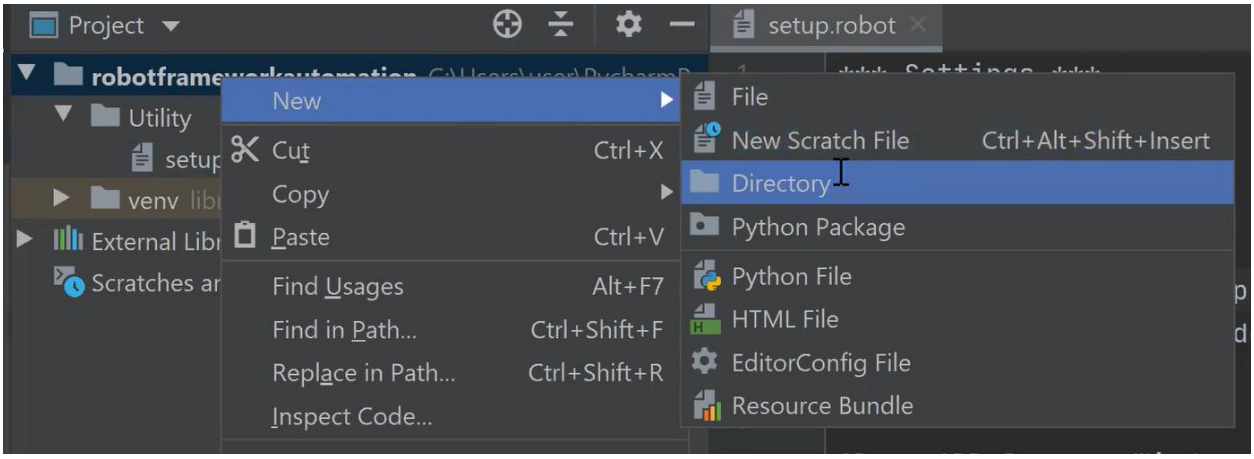
Close All Browsers		Closes all open browsers and resets the browser cache. After this keyword new indexes returned from *Open Browser* keyword are reset to 1. This keyword should be used in test or suite teardown to make sure all browsers are closed.

Now save this file here is how setup.robot will look

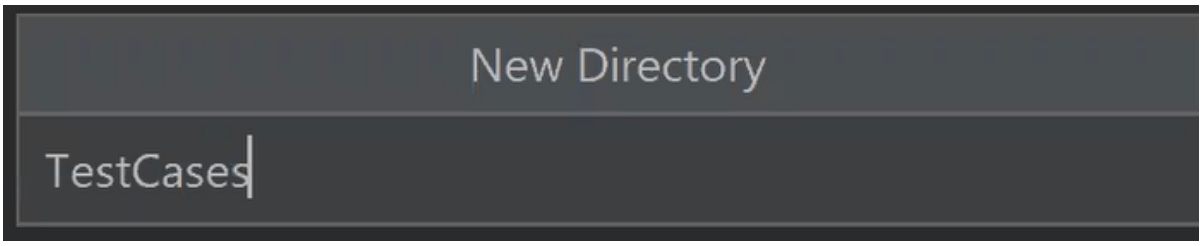


Next we wil see how we can use these keyswords in our test cases.

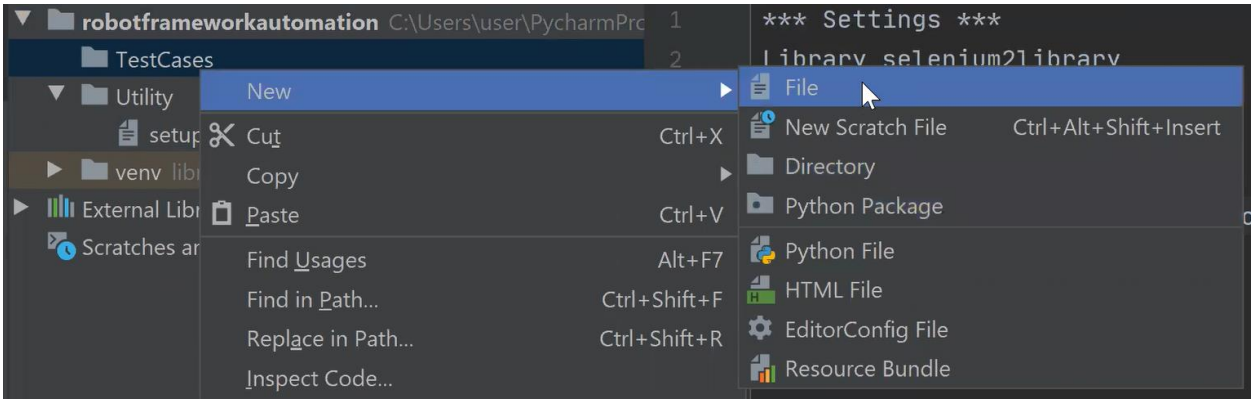
Create another folder right the project->New->Directory



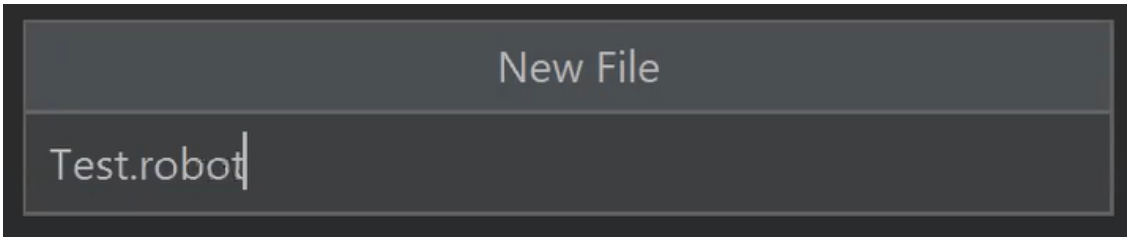
Name the directory TestCases. Under this directory we will create out test classes



Now create file under this TestCases directory



For now lets name it Test.robot



Lets add the script to Test.robot.

In order to use the keywords we just created we will need to import the setup.robot file in Settings section.

```
*** Settings ***
Resource    ../Utility/Setup.robot
```

Resource files are imported using the *Resource* setting in the Settings section. The path to the resource file is given as an argument to the setting.

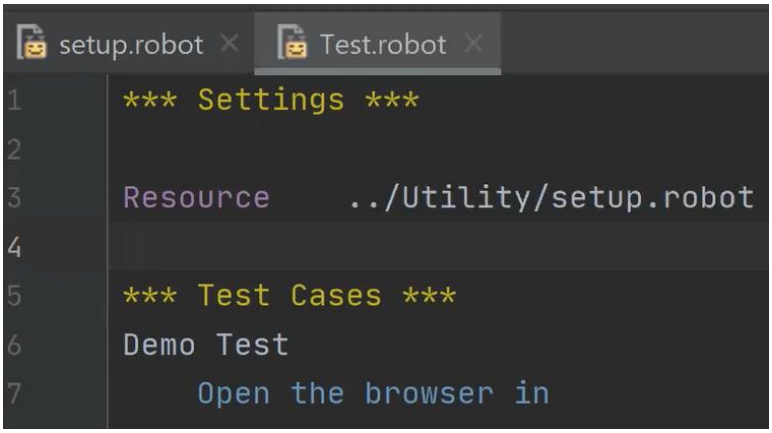
Now we will write test cases under Test Case section

```
*** Test Cases ***
```

We name our Test case as Demo Test and in second row we will call the keyword Open the browser in

```
*** Settings ***
Resource    ../Utility/Setup.robot
*** Test Cases ***
Demo Test
  Open the browser in
```

This is how test case file will look like

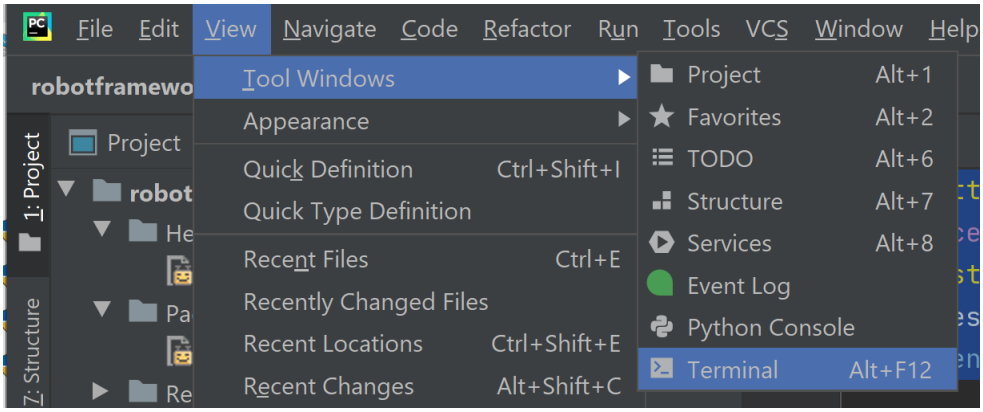


Test cases are constructed in test case tables from the available keywords. Keywords can be imported from test libraries or resource files, or created in the keyword table of the test case file itself.

The first column in the test case table contains test case names which in our case is Demo Test. A test case starts from the row with something in this column and continues to the next test case name or to the end of the table. It is an error to have something between the table headers and the first test.

The second column normally has keyword names which in our case is Open the browser in which we wrote in setup.robot

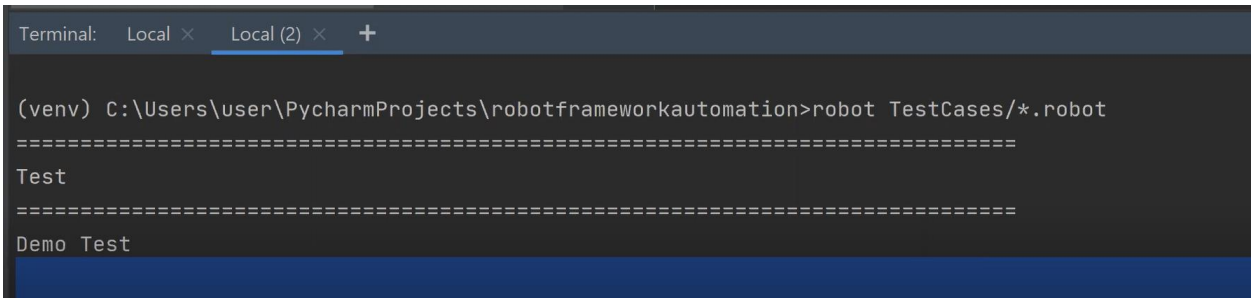
Now lets run this test. In order to run lets launch terminal view->Tool Windows->Terminal



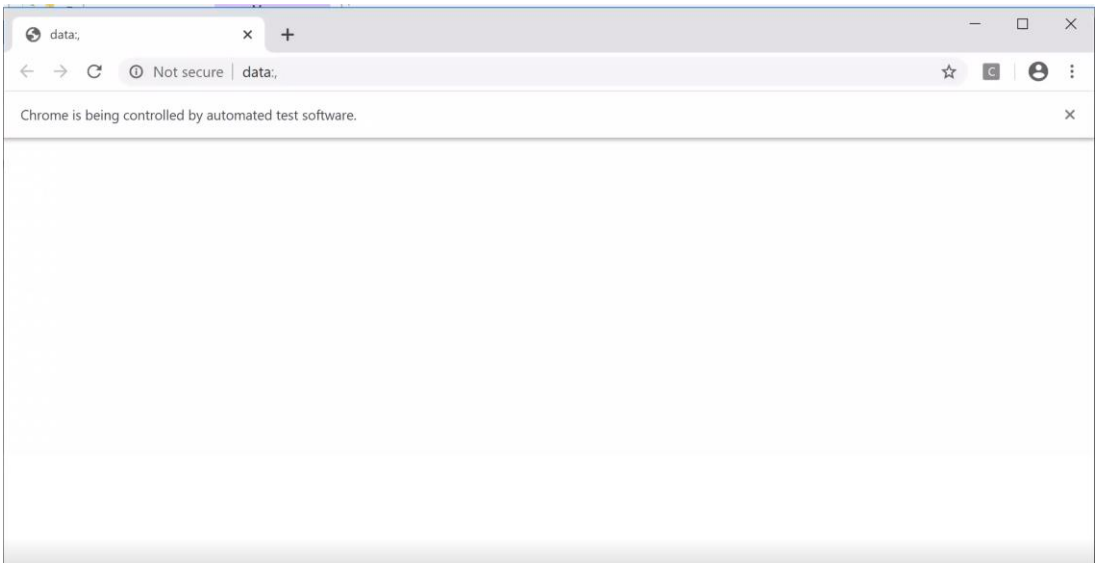
Than in Terminal Window type following command robot TestCases/*.robot

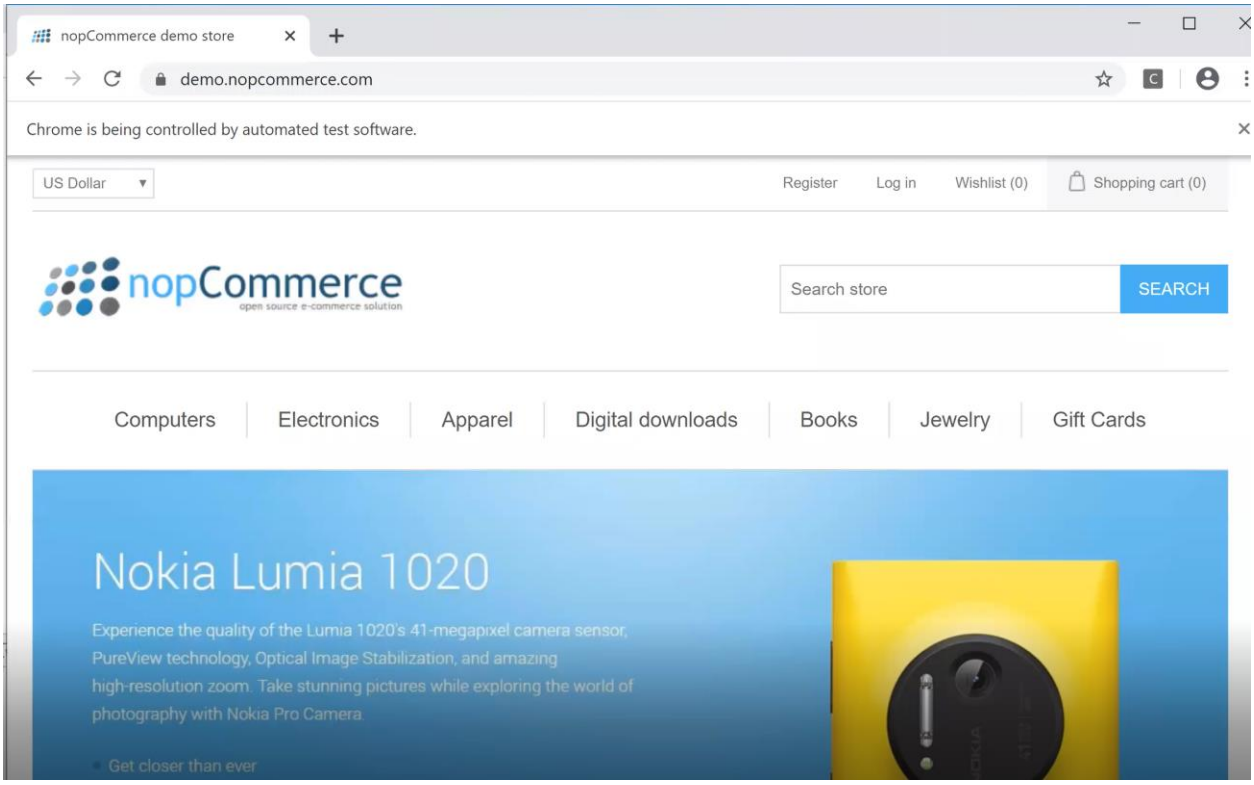
```
(venv) C:\Users\user\PycharmProjects\robotframeworkautomation>robot TestCases/*.robot
```

It will start executing the test case



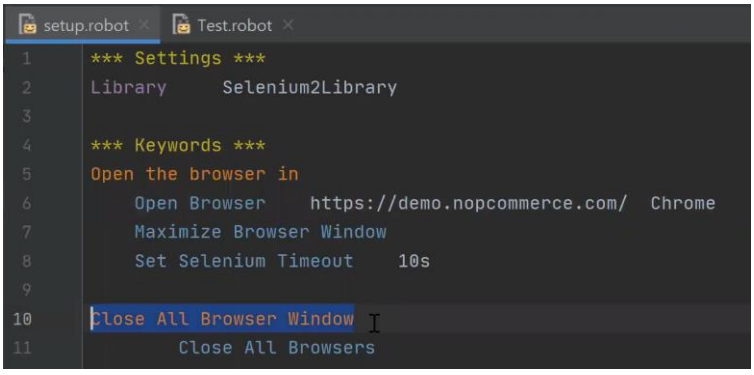
As per keywords under function Open the browser in declared in setup.robot it will launch the browser and navigate to URL, you will see chrom browser opening and navigating and finally maximize the browser





Since we didn't call Close All Browser Window function so thus it wont close browser as keyword Close All Browsers is under it.

Now lets copy function Close All Browser Window from setup.robot and paste it in Test.robot



Test.robot:

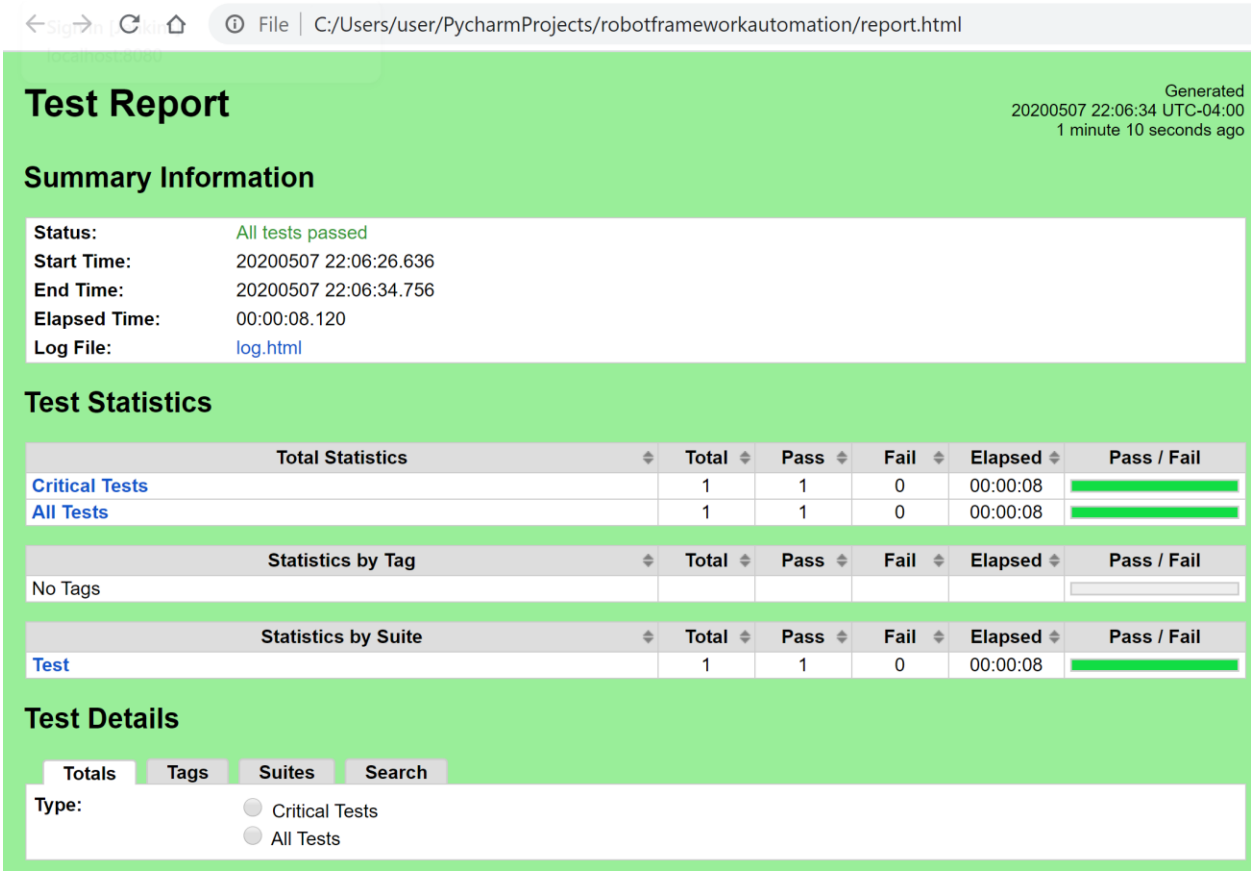
```
*** Settings ***
Resource    ../Utility/Setup.robot
*** Test Cases ***
Demo Test
   Open the browser in
   Close All Browser Window
```

Now save and run again you will see browser closed . Now let see terminal output test result

```
Terminal:  Local  Local (2)  +
DevTools listening on ws://127.0.0.1:63522/devtools/browser/c2f49cfe-dcab-45c0-8790-04250621b4bf
[23300:16000:0507/220628.076:ERROR:browser_switcher_service.cc(238)] XXX Init()
Demo Test                                                              | PASS |
------------------------------------------------------------------------------
Test                                                                   | PASS |
1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
==============================================================================
Output:  C:\Users\user\PycharmProjects\robotframeworkautomation\output.xml
Log:     C:\Users\user\PycharmProjects\robotframeworkautomation\log.html
Report:  C:\Users\user\PycharmProjects\robotframeworkautomation\report.html
```

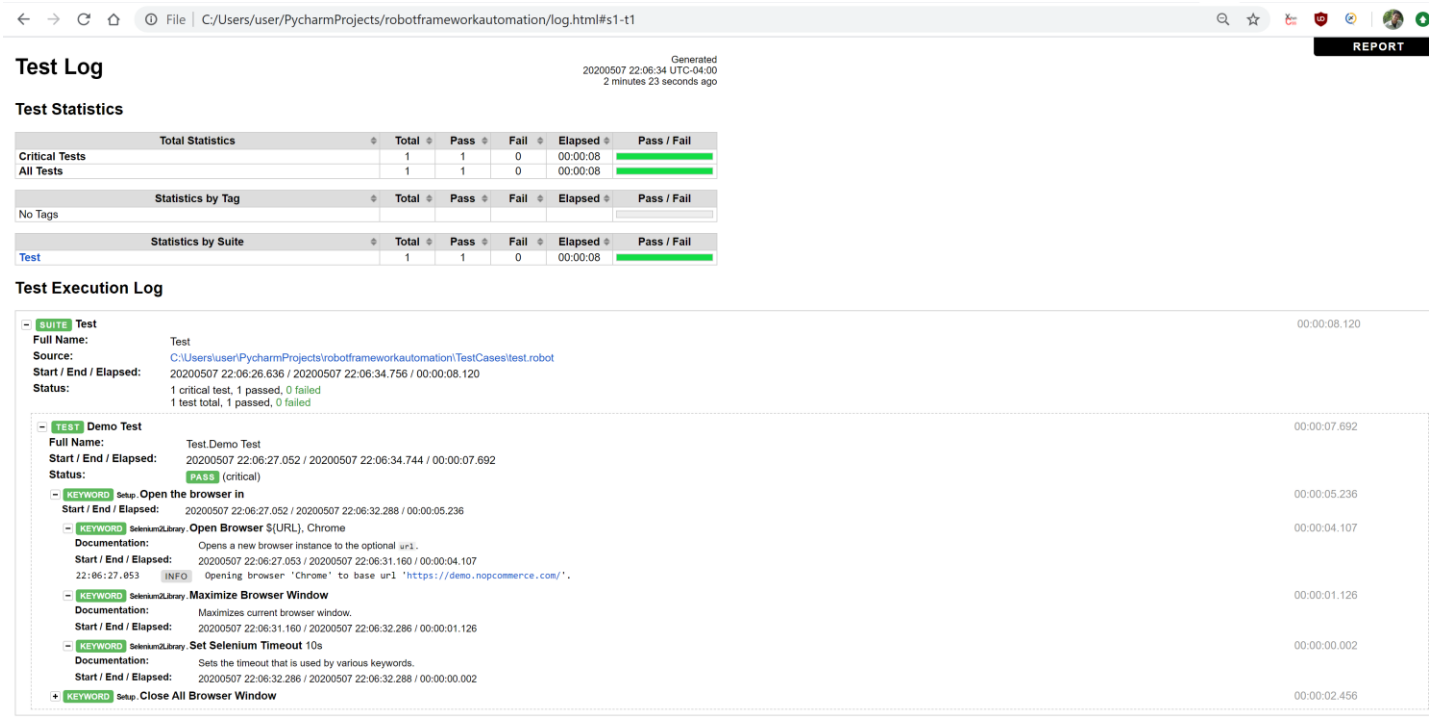
You can see test passed and it shows the path where the test report is saved. Lets open report.html

You will see Test Report showing details about test we ran Test



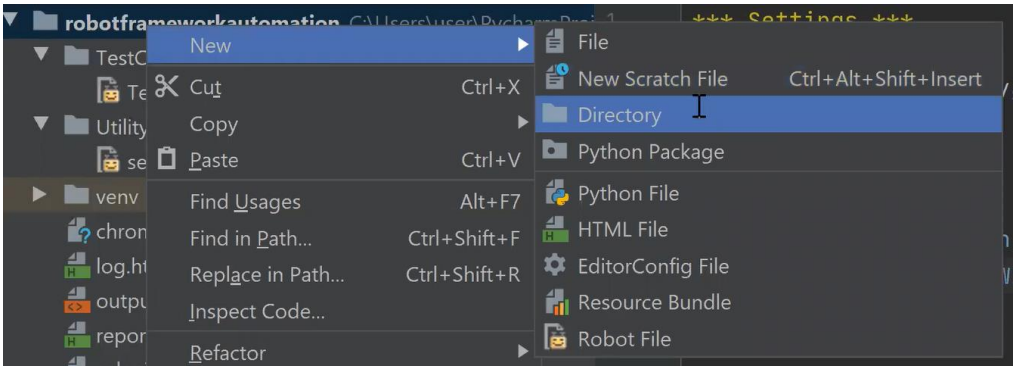
Click on Test, it will take you to Test Details page

Now click on Demo Test to see the steps in the test executed

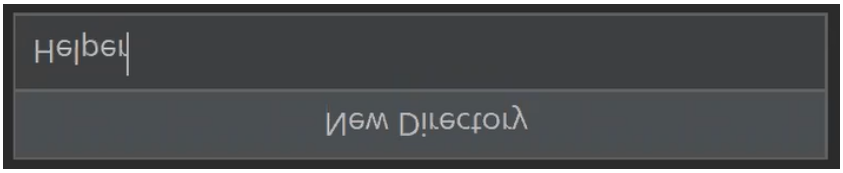


Now lets start writing steps where we will interact with UI elements like text fields or clicking button. In order to proceed with that we will create two folders Helper and Identifier .

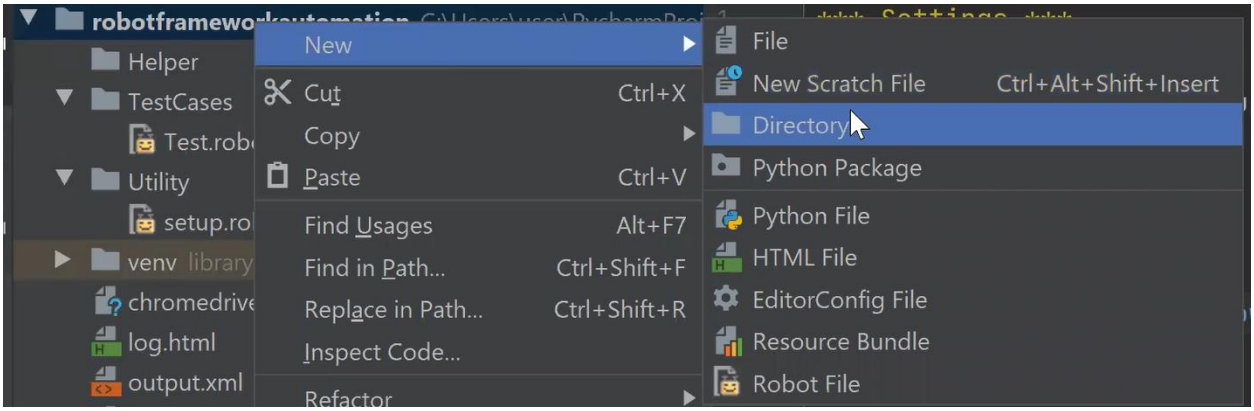
Right click the project->New->Directory



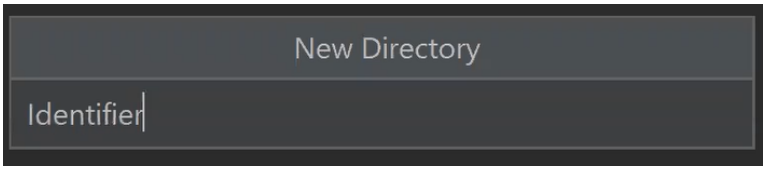
Now name directory as Helper



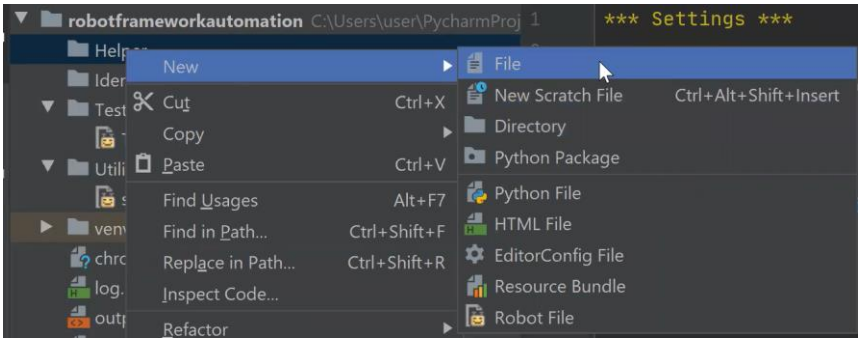
Now lets create identifier directory. Right click project->New->Directory

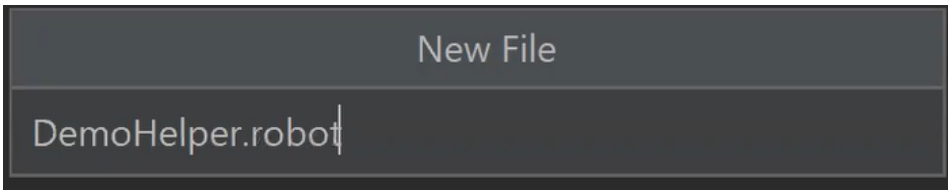


Now name the directory as Identifier



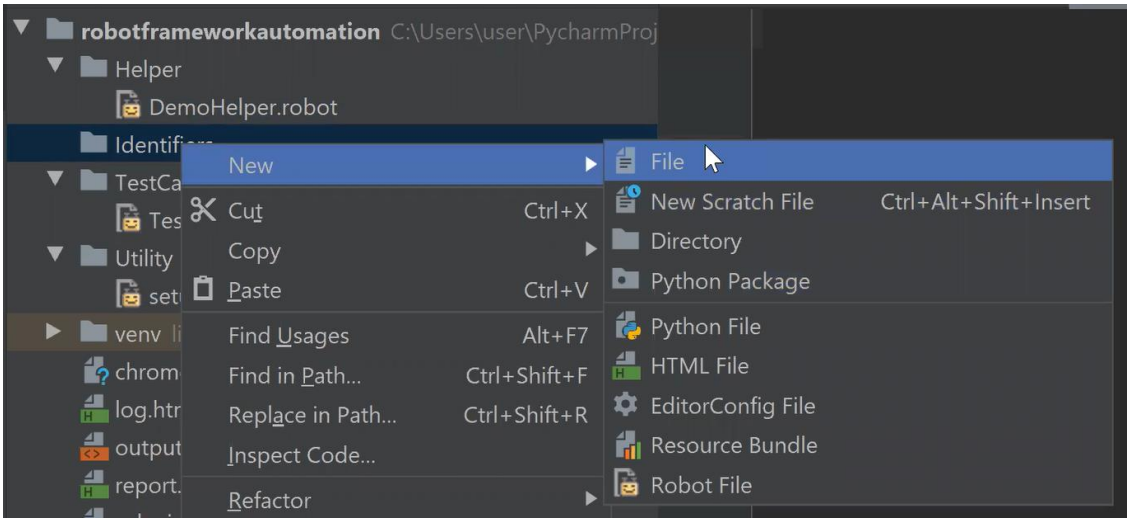
Under Helper directory we will create file DemoHelper.robot

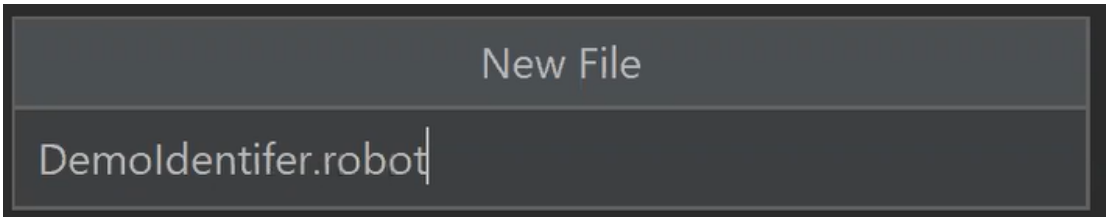




In this DemoHelper.robot we will create helper functions which will be reusable as per best practice to perform clicks on button , setting text in input fields etc. This way we wont have to re-write those function in each and every test class or wherever we perform those action.

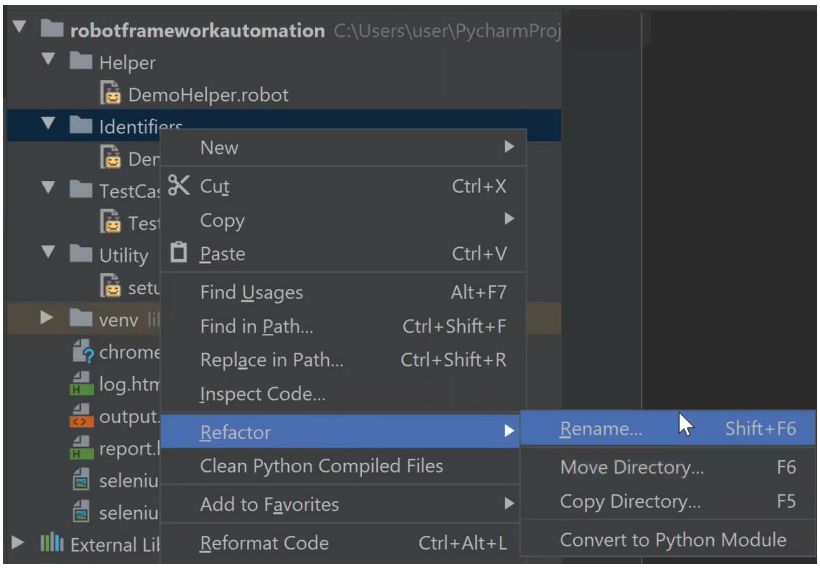
Now lets create file under Identifier folder DemoIdentifer.robot



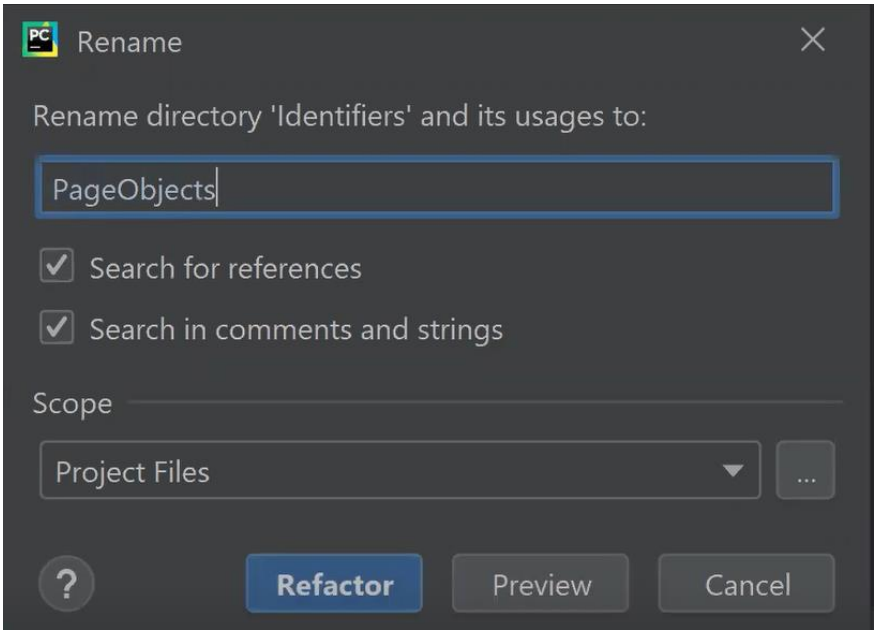


This file DemoIdentier.robot will contains the elements/obecjts(buttons,text fields etc.) locator either their xpaths or css or id

Lets rename directory name from Identifer to PageObjects. Right click Identifier directory->Refactor->Rename



Set PageObjects and click on Refactor



Here is scenario:

User clicks on login button/link

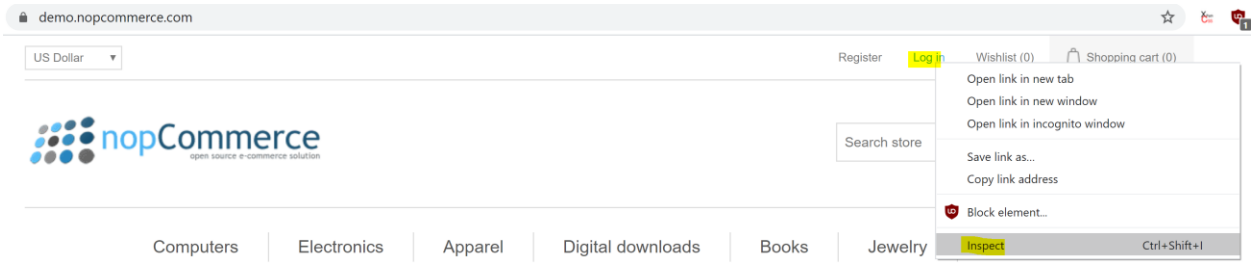
User enter username

User enter password

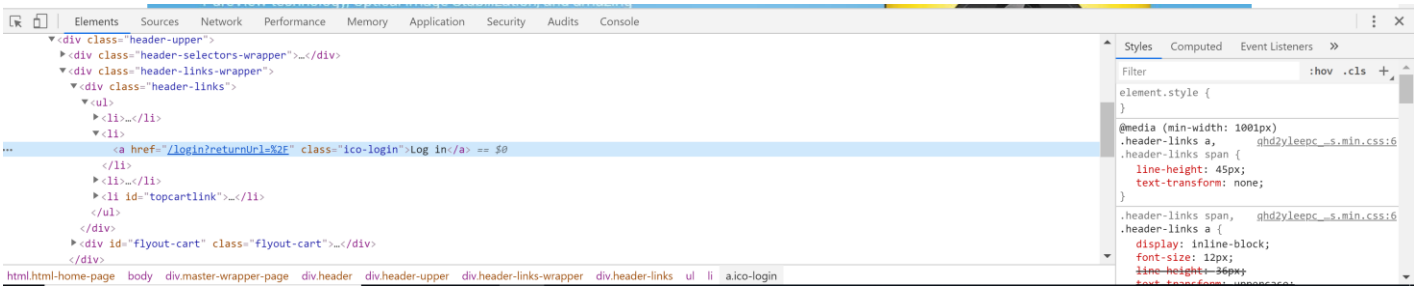
User clicks login button

Now all the objects of we will store here. Lets get locator for login from website https://demo.nopcommerce.com/

Right click login->Inspect



It will launch the window below highlight login html code



Press ctrl+f it will launch small editor below

I will be using css selector to locate it below is syntax for :

css=*tag*[*attribute*=*value*]

- tag = the HTML tag of the element being accessed
- [and] = square brackets within which a specific attribute and its corresponding value will be placed
- attribute = the attribute to be used. It is advisable to use an attribute that is unique to the element such as a name or ID.
- value = the corresponding value of the chosen attribute.

Now lets look at locator highlighted

```
<li>
    <a href="/login?returnUrl=%2F" class="ico-login">Log in</a> == $0
</li>
<li>…</li>
```

tag = a

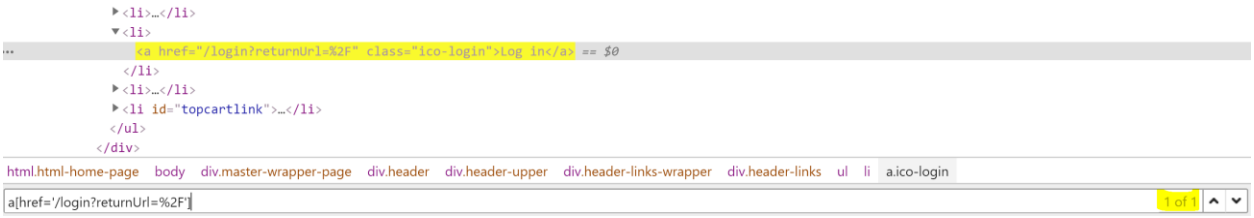
attribute = href

value = /login?returnUrl=%2F

Thus its locator will be : a[href='/login?returnUrl=%2F']

Now paste this in that small editor at the bottom of page

As soon as you enter you will see the login html code gets highlighted for login link and at same time it shows how many are located with same in entire DOM



Now lets add this to DemoIdentifer.robot.

Frist will add variable section as below

```
*** Variables ***
```

The most common source for variables are Variable tables in test case files and resource files. Variable tables are convenient, because they allow creating variables in the same place as the rest of the test data, and the needed syntax is very simple.

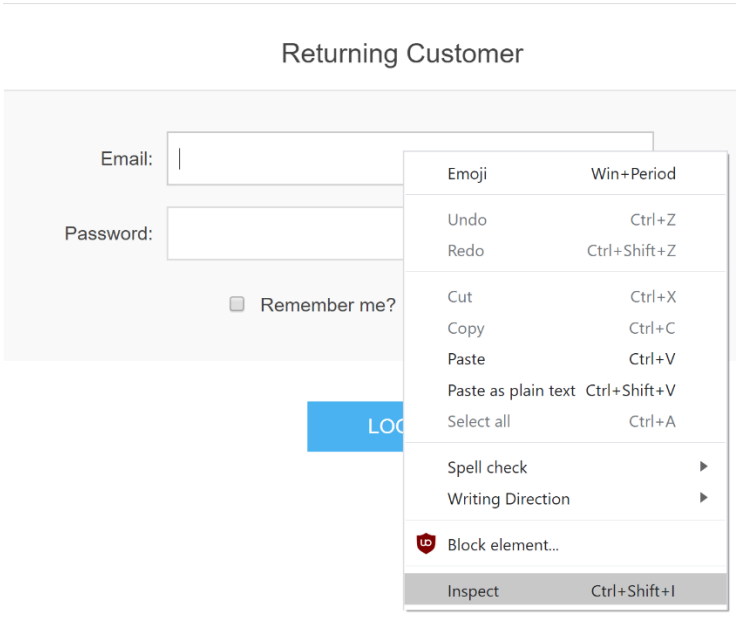
Now let's add locator:

```
*** Variables ***
${LoginLink}    css=a[href='/login?returnUrl=%2F']
```

In order to use the or get value of locator we will calling ${LoginLink} that's variable in which its stored.

Now lets add locators for username field, password field and Login button. All three are present once you click on login from website.

Right click in the email field and select Inspect



The HTML code will be highlithed for email field:

```
▼<div class="inputs">
    <label for="Email">Email:</label>
    <input class="email" autofocus type="email" data-val="true" data-val-email="Wrong email" data-val-required="Please enter your email" id="Email" name=
    "Email"> == $0
    <span class="field-validation-valid" data-valmsg-for="Email" data-valmsg-replace="true"></span>
    ::after
  </div>
```

This time I will use id attribute to locate the field their unique and easy to use

Syntax: id=*id of the element*

In our case id=Email for our Email field

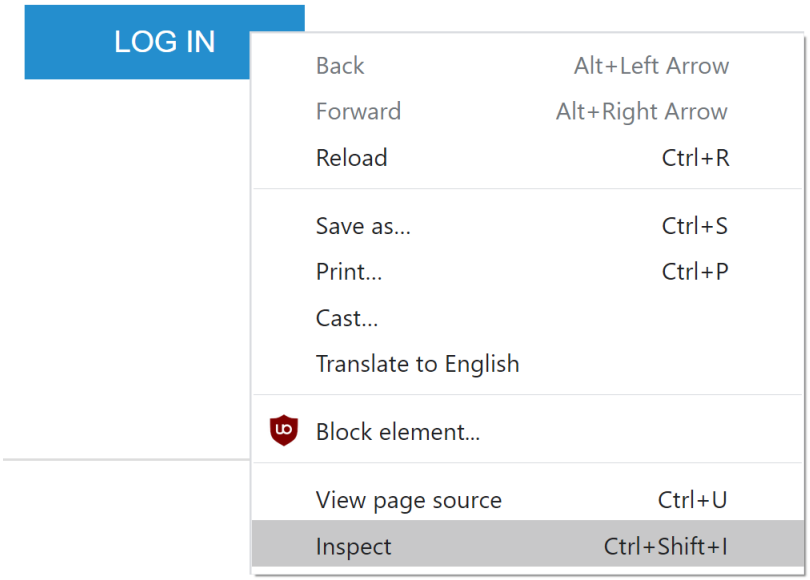
Now lets add this to DemoIdentifer.robot.

```
*** Variables ***
${LoginLink}    css=a[href='/login?returnUrl=%2F']
${EmailField}   id=Email
```

Similarily we will use id for Password field as above syntax will remain same and we add to DemoIdentifer.robot file

```
*** Variables ***
${LoginLink}    css=a[href='/login?returnUrl=%2F']
${EmailField}   id=Email
${PasswordField}    id=Password
```

Last but not the least we will add locator for LOG IN button. Right click and select Inspect



You will see highlighted locator

```
▼<div class="buttons">
    <input class="button-1 login-button" type="submit" value="Log in"> == $0
  </div>
```

Now here we don't have id which is unique identifier thus we will use css selector

css=*tag*[*attribute*=*value*]

tag = input

attribute = type

value = submit

Thus its locator will be : input[type='submit']

Now lets check whether we can locate LOG IN button using this css locator. Paste this in small editor (launch using ctrl+f)

Once I pasted it showed two results

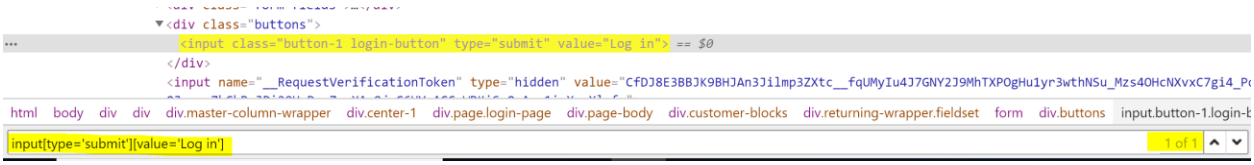
In above figure you will see the first one but if you hover your mouse on it this is for Search button to see the 2nd click on down arrow from small editor

```
▼<div class="buttons">
    <input class="button-1 login-button" type="submit" value="Log in"> == $0
  </div>
```

Hover the mouse on the locator it will highlight the LOG IN button. The key difference between two is the attribute value for both are different thus we can use that attribute as well. There our locator will be come

input[type='submit'][value='Log in']

Lets put this again in small editor and you will see it locates it and this time its shows 1 of 1 result



tag = input

attribute = type , value

value = submit , Log in

Thus its locator will be : input[type='submit'][value='Log in']

Lets add this to DemoIdentifier.robot file

```
*** Variables ***
${LoginLink}    css=a[href='/login?returnUrl=%2F']
${EmailField}   id=Email
${PasswordField}    id=Password
${LoginButton}  css=input[type='submit'][value='Log in']
```

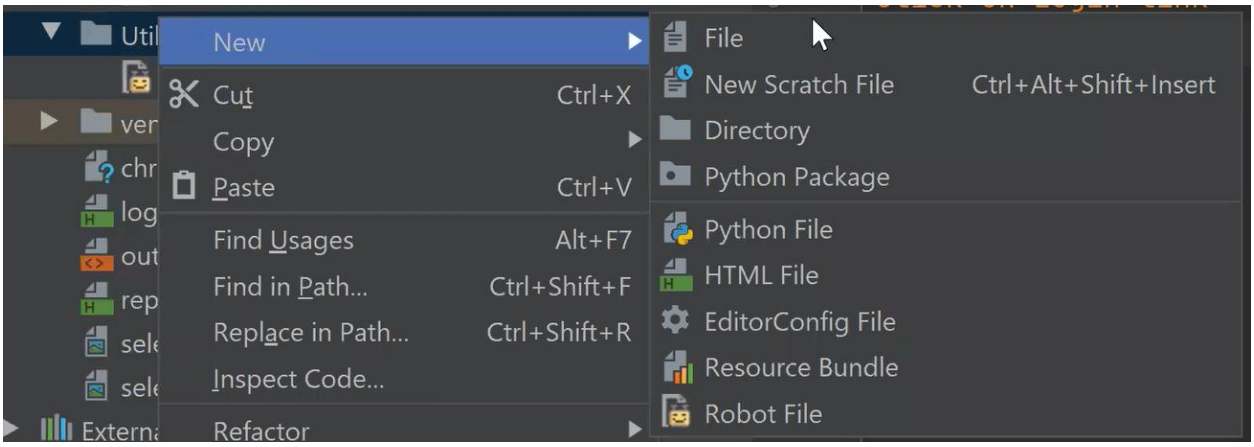

```
*** Variables ***
${LoginLink}    css=a[href='/login?returnUrl=%2F']
${EmailField}   id=Email
${PasswordField}    id=Password
${LoginButton}  css=input[type='submit'][value='Log in']
```


Save the file.

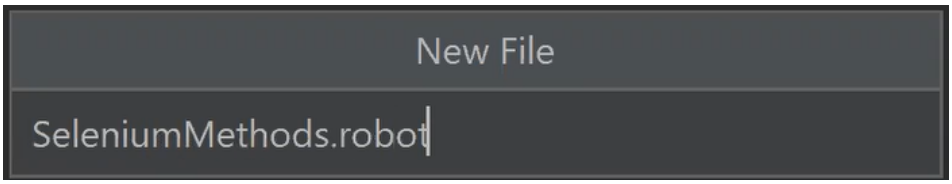
Now let me explain reason why we created DemoIdentifer.robot to save all page objects in that file because its as per best practices we keep in centralized location and save them page by page meaning every page will have locator file which will contains locator of that particular page. If you don't use this approach and use locators directly in test cases in various test files and developer changes or modify locator thus we will have to manually go and check where its used and change, however in case of approach using POM (Page Object Model) we will need to change at one place thus maintainability and re-usability is achieved.

In order to use these object locators to perform certain actions like clicks or set text on them lets create functions and use selenium methods.

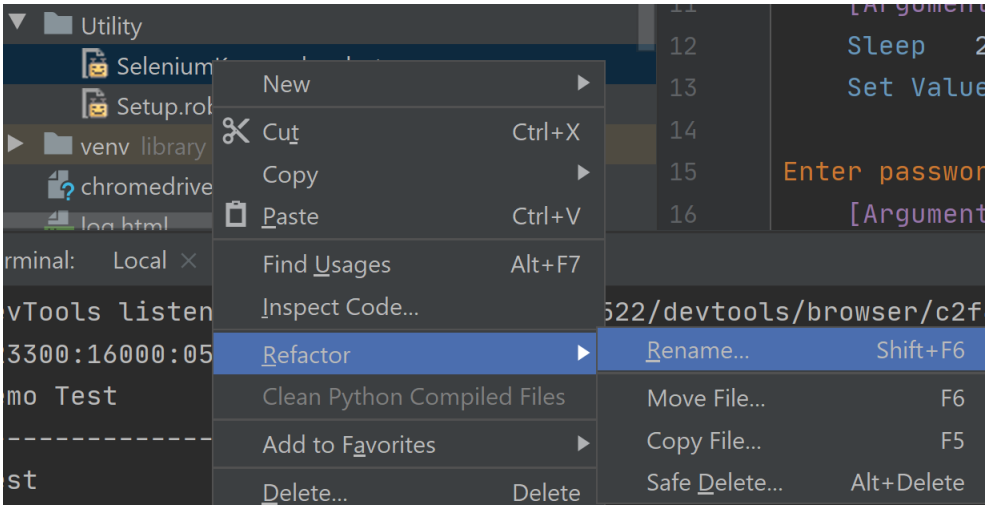
Lets create SeleniumMethods.robot file under Utility folder in this we will write selenium methods



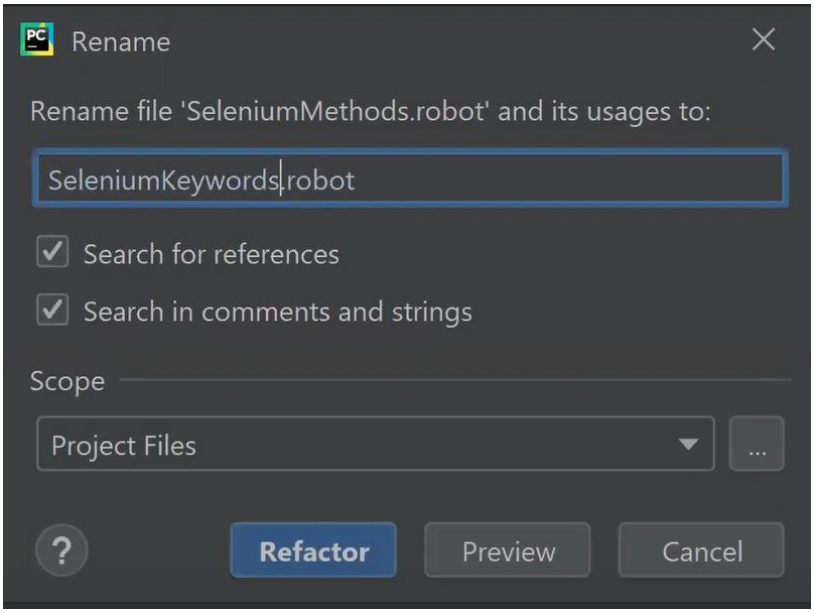
This is file in which we will create re-usable methods to perform clicks, set text etc



Lets re-name the file to SeleniumKewords.robot . To do so right click on SeleniumMethods.robot->Refactor->Rename



Change the name and click Refactor button



Lets write code in this SeleniumKeywords.robot. In order to use selenium methods we will need to import the selenium library under Settings section as you can see in code below:

```
*** Settings ***
Library    Selenium2Library
```

Now we have imported Selenium2Library let see write functions under Keywords section:

```
*** Settings ***
Library    Selenium2Library

*** Keywords ***
Click on Element
  [Arguments]    ${locator}
  Click Element    ${locator}
```

Lets examine the above code so we have written method Click on Element in which we are passing parameter ${locator} and in last step we are preforming click on the page object using its locator with help selenium method Click Element

As per selenium2library it takes argument/parameter locator

Click Element	*locator*	Click element identified by `locator`. See the *Locating elements* section for details about the locator syntax.

That's why in second line we have [Arguments] ${locator}

Next in scenario similarly we have set text in user name field we will write method which will help us to set text .

```
*** Settings ***
Library    Selenium2Library

*** Keywords ***
Click on Element
  [Arguments]    ${locator}
  Click Element  ${locator}

Set Value For Input Field
  [Arguments] ${locator}    ${value}
  Input Text      ${locator}  ${value}
```

So you can see we have written Set Value For Input Field method which takes parameter ${locator} and using Input Text perform set text that will be passed as parameter ${value}

```
*** Settings ***
Library      Selenium2Library

*** Keywords ***
Click on Element
    [Arguments]      ${locator}
    Click Element    ${locator}

Set Value For Input Field
    [Arguments]      ${locator}      ${value}
    Input Text    ${locator}     ${value}
```

Go to DemoHelper.robot file which we created under Helper directory . First things we will do is import the locators so we can perform actions in script

```
*** Settings ***
Resource    ../PageObjects/DemoIdentifer.robot
```

Then we will import SeleniumKeywords.robot since we will be using methods we wrote to perfom click and set text

```
*** Settings ***
Resource    ../PageObjects/LoginObjetcs.robot
Resource    ../Utility/SeleniumKeywords.robot
```

Now we will create functions under Keywords section and each function will be used in our test case file test.robot

Keeping scenario in mind lets write functions, thus first thing we need to do after browser is launched is click on login link

Lets write code for it

```
*** Keywords ***
Click on Login link
   Click on Element    ${LoginLink}
```

So we have written function Click on Login link and we have used Click on Element function we wrote in SeleniumKeywords.robot and passed ${LoginLink} as parameter . ${LoginLink} is page locator for login link on website and Click on Element function in SeleniumKeywords.robot we needs paratmer ${locator}.

So this locator of login link is passed so it can perform click on it.

Now lets write function to set username in username field

```
*** Settings ***
Resource    ../PageObjects/LoginObjetcs.robot
Resource    ../Utility/SeleniumKeywords.robot

*** Keywords ***
Click on Login link
   Click On Page Element    ${LoginLink}

Enter username
   [Arguments]    ${value}
   Set Value For Input Field  ${EmailField}   ${value}
```

You see function Enter Username take parameter ${value} passed to it from test.robot and in second line we can see we called function Set Value For Input Field from SeleniumKeywords.robot which we wrote to set text in the username field thus we are passing locator and value (text we want to set) to this function as it takes both.

Similarliy now we will write function for setting password in that field

```
*** Settings ***
Resource    ../PageObjects/LoginObjetcs.robot
Resource    ../Utility/SeleniumKeywords.robot

*** Keywords ***
Click on Login link
   Click On Page Element    ${LoginLink}

Enter username
   [Arguments]    ${value}
   Set Value For Input Field  ${EmailField}   ${value}
```

```
Enter password
  [Arguments]    ${value}
  Set Value For Input Field  ${PasswordField}   ${value}
```

Just like setting username field we are setting password field .

Now last but no least we have to perform click on login button thus we will need function lets write that

```
*** Settings ***
Resource   ../PageObjects/LoginObjetcs.robot
Resource   ../Utility/SeleniumKeywords.robot
Library    BuiltIn

*** Keywords ***
Click on Login link
  Click On Page Element   ${LoginLink}

Enter username
  [Arguments]    ${value}
  sleep  10s
  Set Value For Input Field  ${EmailField}   ${value}

Enter password
  [Arguments]    ${value}
  Set Value For Input Field  ${PasswordField}   ${value}

Click on Login button
  Click On Page Element   ${LoginButton}
```

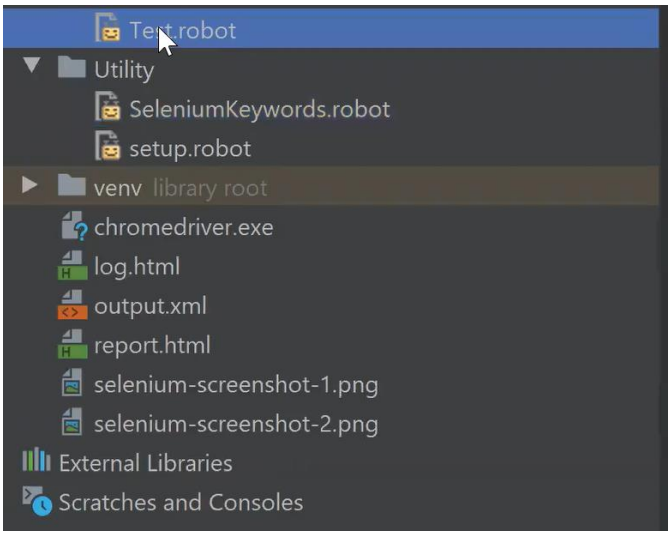
You can see Click on Login button is similar to Click on Login link function we wrote earlier and just that we will be using LOG IN buttons locator.

```
*** Settings ***
Resource    ../PageObjects/LoginObjetcs.robot
Resource    ../Utility/SeleniumKeywords.robot
Library     BuiltIn

*** Keywords ***
Click on Login link
    Click on Element    ${LoginLink}

Enter username
    [Arguments]     ${value}
    sleep   10s
    Set value in the field  ${EmailField}   ${value}

Enter password
    [Arguments]     ${value}
    Set value in the field  ${PasswordField}    ${value}

Click on Login button
    Click on Element    ${LoginButton}
```

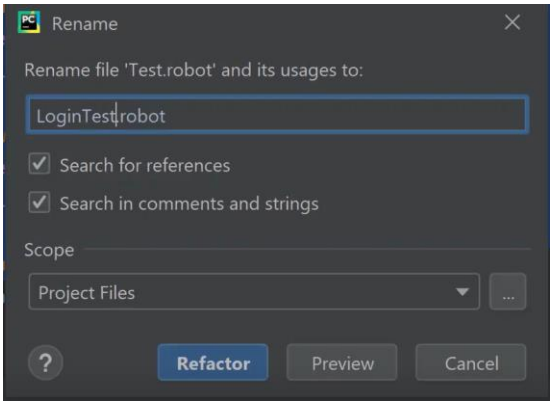
Now save the file .

Before we proceed further I would like to rename some of files . Change the name of test case file from test.robot to LoginTest.robot

Right click Test.robot->Refactor-Rename

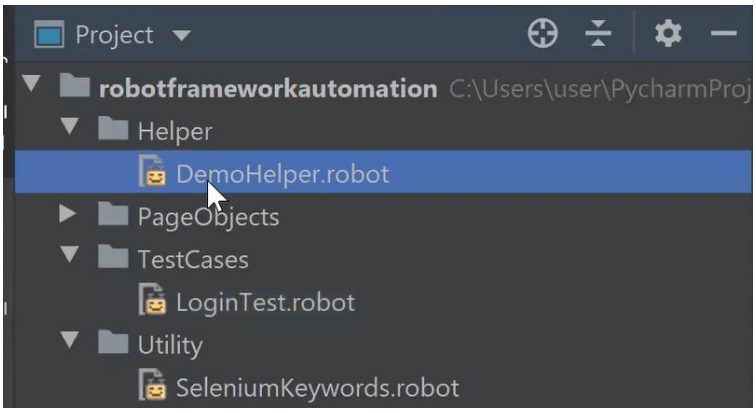


Change it to LoginTest.robot and click Refactor

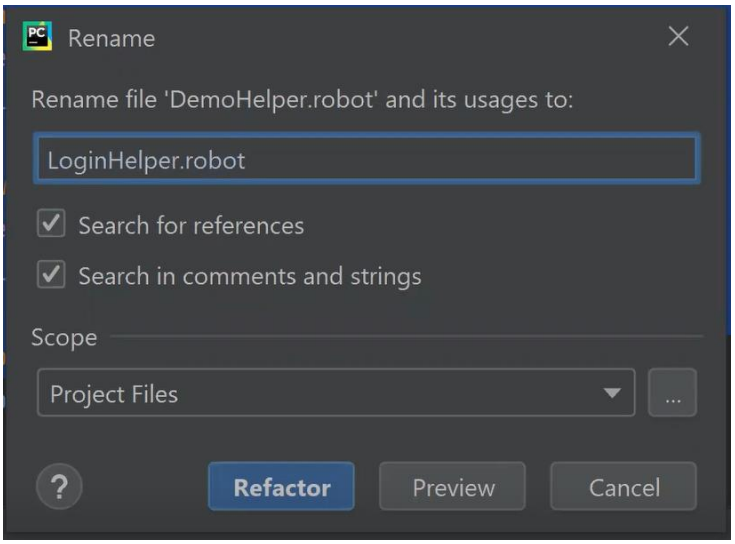


Simiarlily we will change DemoHelper.robot to LoginHelper.robot

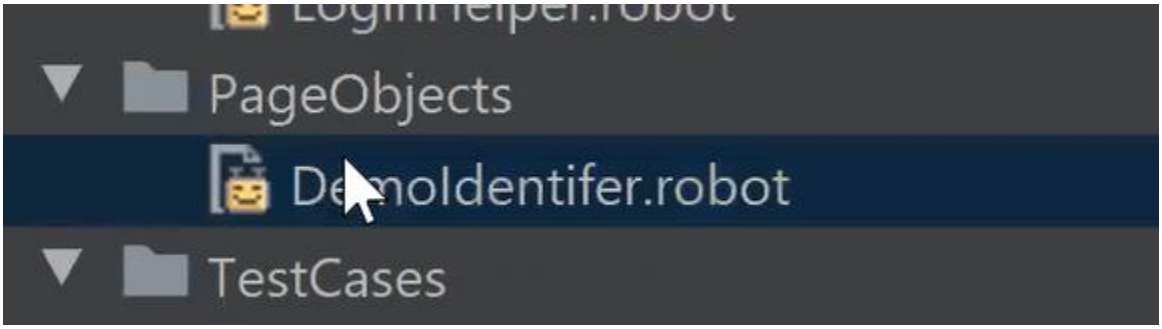
Right click DemoHelper.robot -> Refactor -> Rename



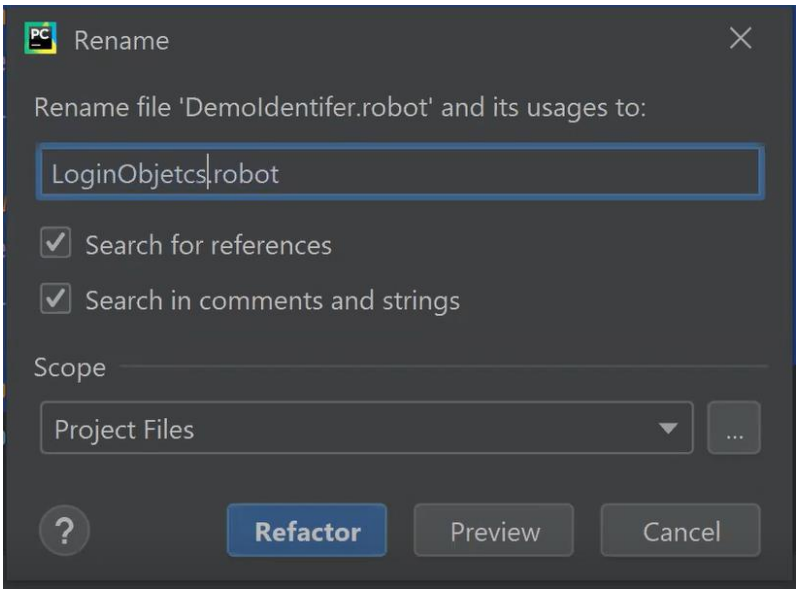
Set to LoginHelper.robot and click Refactor



Next we will change name of DemoIdentifer.robot file right click->Refactor-Rename



Change to LoginObjetcs.robot and click Refactor button



Now lets go to our LoginTest.robot file where we will steps from scenario by calling these functions.

First thing we will do is import LoginHelper.robot because we will using helper methods

```
*** Settings ***
Resource    ../Utility/Setup.robot
Resource    ../Helper/LoginHelper.robot

*** Test Cases ***
Demo Test
    Open the browser in
    Close All Browser Window
```

Next we will add the login helper methods to perform scenario under Test Cases section, earlier we were just opening and closing browser using seleniumkeywords.robot

```
*** Settings ***
Resource    ../Utility/Setup.robot
Resource    ../Helper/LoginHelper.robot
*** Test Cases ***
Demo Test
    Open the browser in
    Click on Login link
    Enter username  waheedahmed55@gmail.com
    Enter password  test1
    Click on Login button
    Close All Browser Window
```

Now you can see have called functions Click on Login link , Enter username with paratmer waheedahmed55@gmail.com , Enter password with parameter test1 and finally Click on Login button

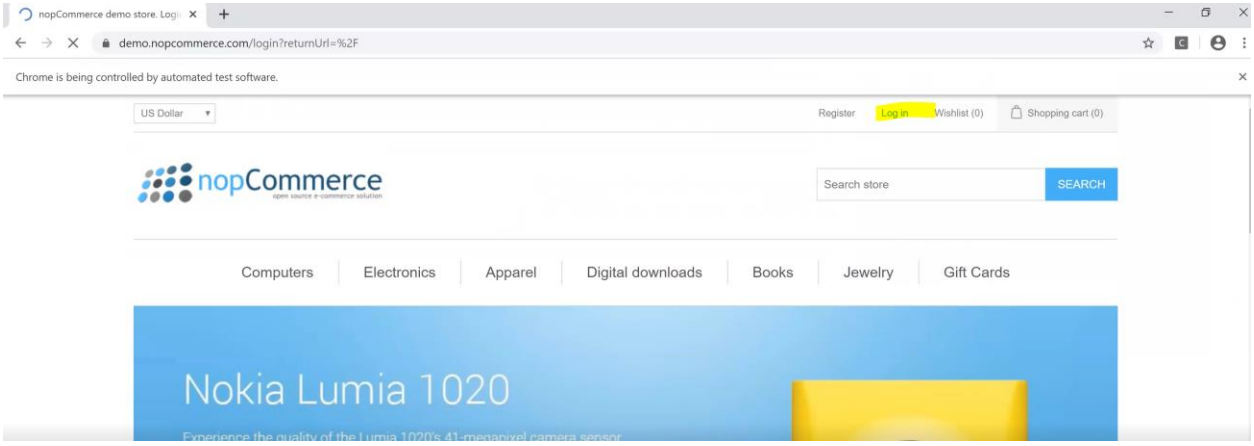
```
*** Settings ***
Resource    ../Utility/setup.robot
Resource    ../Helper/LoginHelper.robot

*** Test Cases ***
Demo Test
    Open the browser in
    Click on Login link
    Enter username  waheedahmed55@gmail.com
    Enter password  test1
    Click on Login button
    Close All Browser Window
```

Save the file.

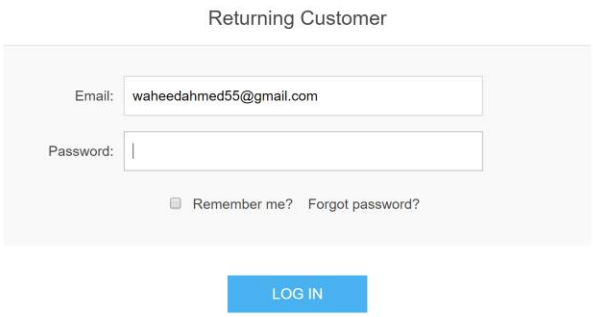
Run the test code :

```
Terminal:  Local  Local (2)  +

(venv) C:\Users\user\PycharmProjects\robotframeworkautomation>robot TestCases/*.robot
```

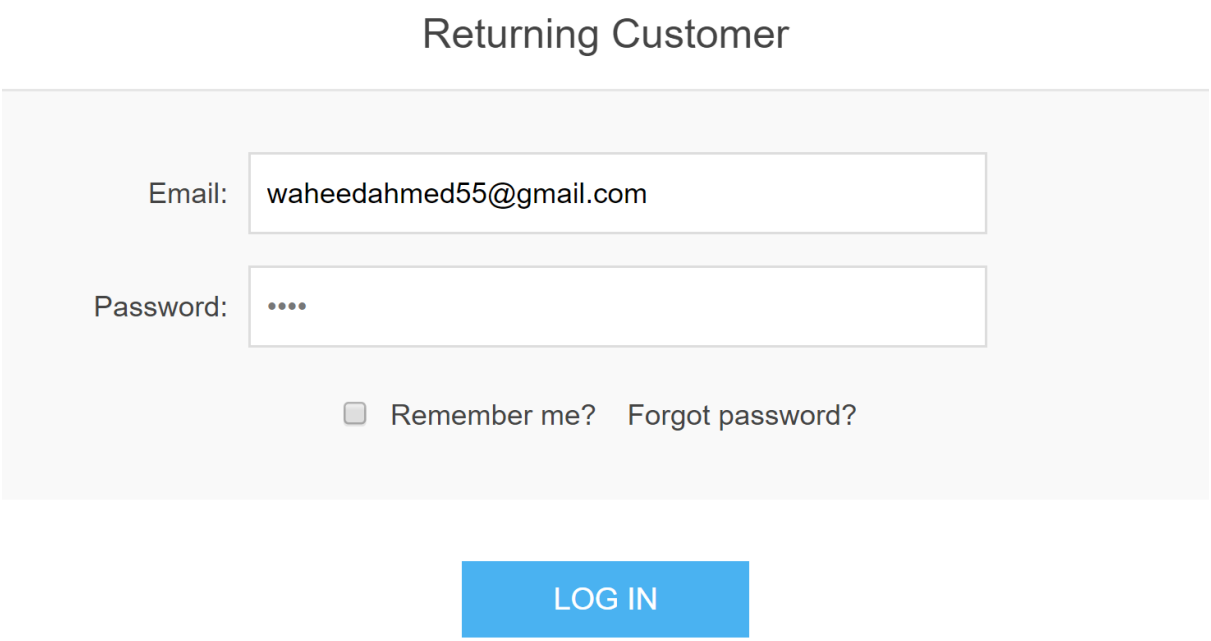
It launch the browser maximize and navigate to the website and click on login



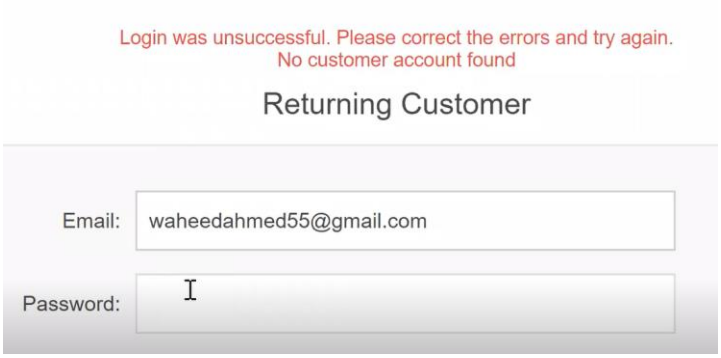
Then it enters Email



Then it enters password and clicks LOG IN button



This invalid password so login wont be successful



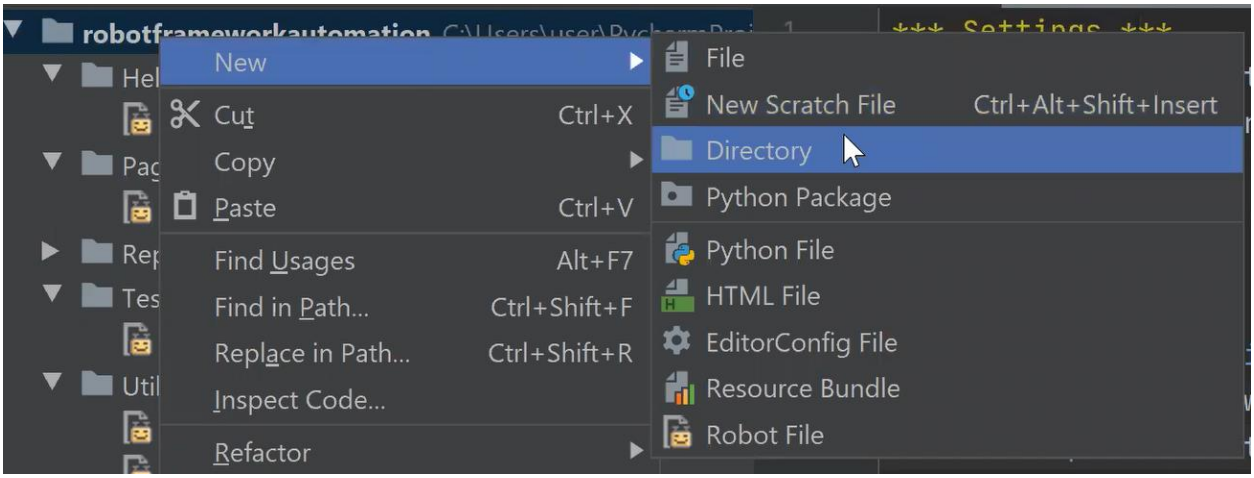
Then it closes browser

You can see output on console :

```
Terminal:  Local  Local (2)  +
LoginTest                                                              | PASS |
1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
==============================================================================
Output:  C:\Users\user\PycharmProjects\robotframeworkautomation\output.xml
Log:     C:\Users\user\PycharmProjects\robotframeworkautomation\log.html
Report:  C:\Users\user\PycharmProjects\robotframeworkautomation\report.html
```

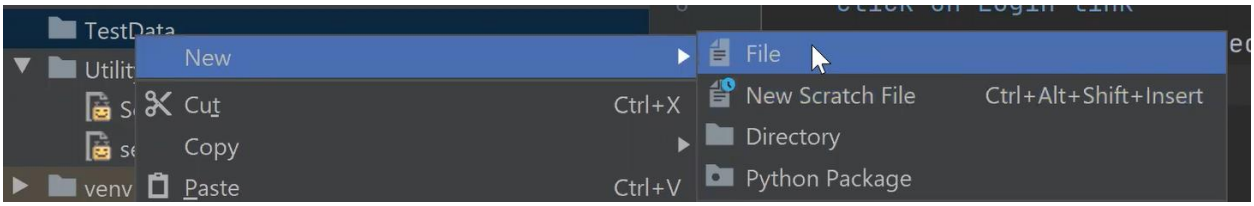
Now let see how we can manage test data currently we are using hard-coded values which Is not good practise and code should look clean and reusable and maintainable data.

Lets create new directory, Right click Project->New ->Directory

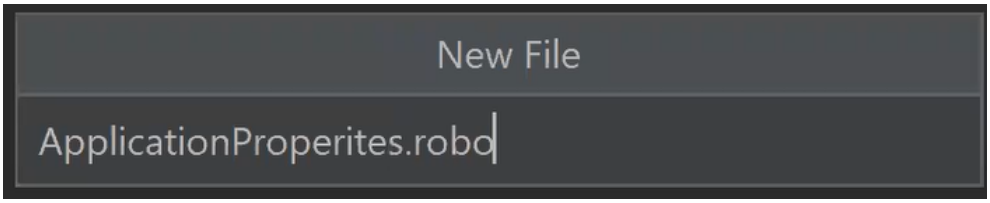


Set directory name as TestData

Under this directory we will create file to store test data. Right click TestData->New->File



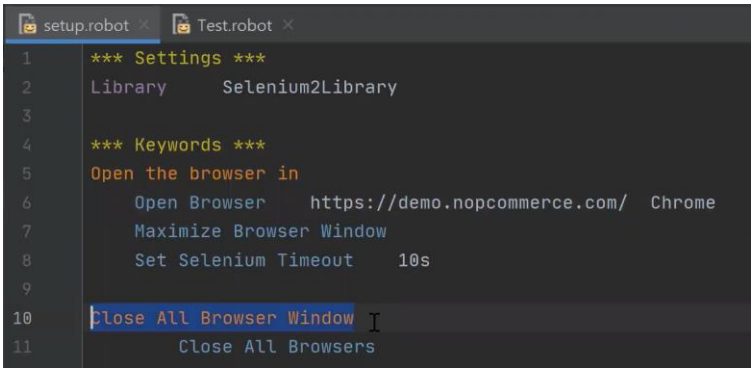
Set name to file name to ApplicationProperties.robot



Now we take all hard-coded values for example , in LoginTest.robot we are passing email address and password as you can see below

```
*** Settings ***
Resource    ../Utility/setup.robot
Resource    ../Helper/LoginHelper.robot

*** Test Cases ***
Demo Test
    Open the browser in
    Click on Login link
    Enter username  waheedahmed55@gmail.com
    Enter password  test1
    Click on Login button
    Close All Browser Window
```

& URL in setup.robot



This is how ApplicationProperties.robot file will look

ApplicationProperties.robot

```
*** Variables ***
${URL}    https://demo.nopcommerce.com/
${UserName}    waheedahmed55@gmail.com
${InvalidPassword}    test1
${Password}    test123
${SearchItem}    Laptop
```

Code:

```
*** Variables ***
${URL}  https://demo.nopcommerce.com/
${UserName}  waheedahmed55@gmail.com
${InvalidPassword}  test1
${Password}  test123
${SearchItem}  Laptop
```

Now we will replace wherever we used them directly by their keys for example for url we will use ${URL}

Replaced in setup.robot you can see ${URL} , also in order to use it will import using Resource under settings

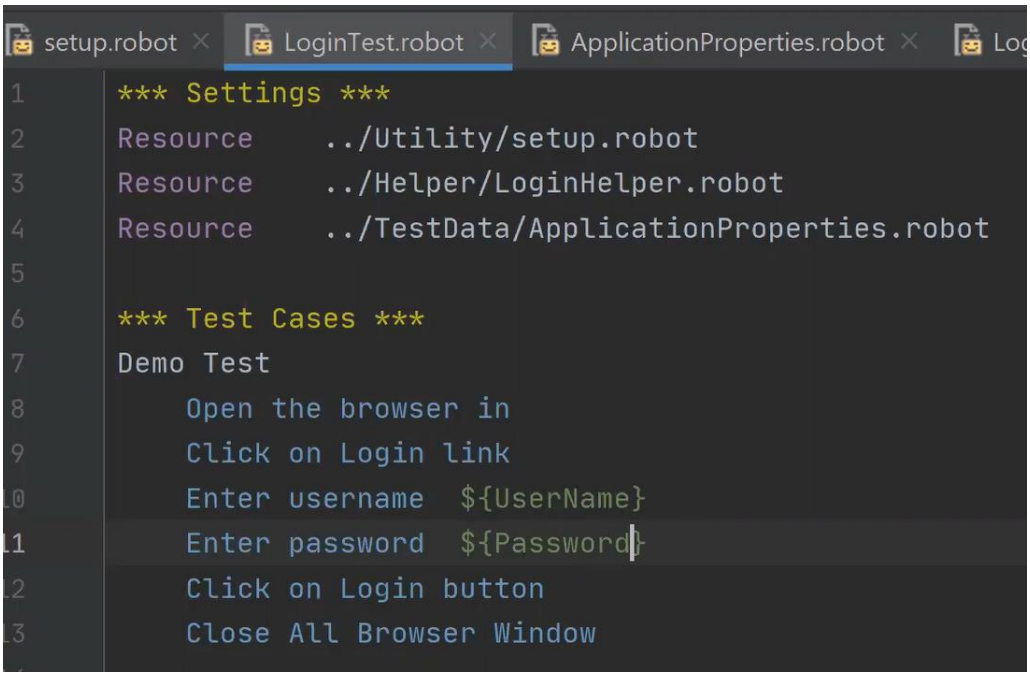
setup.robot | LoginTest.robot | ApplicationProperites.robot

```
*** Settings ***
Library     Selenium2Library
Resource    ../TestData/ApplicationProperties.robot

*** Keywords ***
Open the browser in
    Open Browser    ${URL}  Chrome
    Maximize Browser Window
    Set Selenium Timeout    10s

Close All Browser Window
        Close All Browsers
```

Code:

```
*** Settings ***
Library     Selenium2Library
Resource    ../TestData/ApplicationProperties.robot
Library  OperatingSystem
Library  BuiltIn
Library  String

*** Keywords ***
Open the browser in
   Open Browser    ${URL}  Chrome
   Maximize Browser Window
   Set Selenium Timeout    10s

Close All Browser Window
      Close All Browsers
```

Similarly we will replace username and password in LoginTest.robot and import ApplicationProperties.robot under settings. As you can see below :

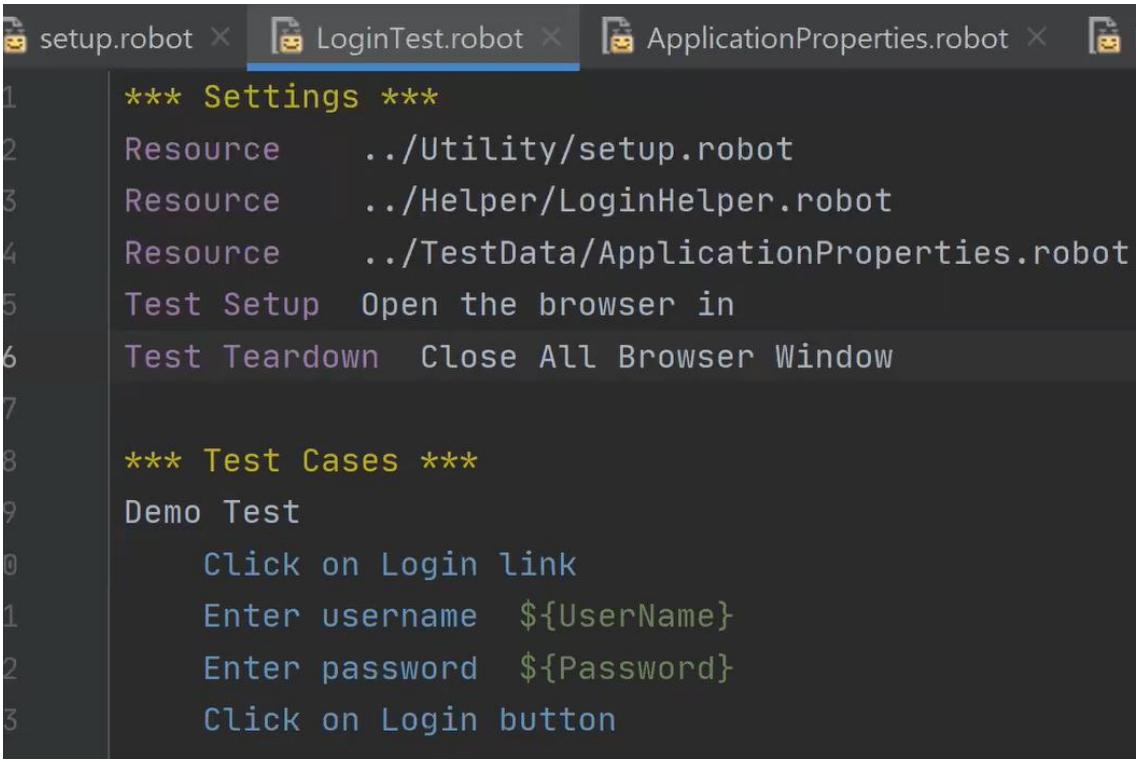


Now anytime you have to change the test data you wont have to change in test case or in different files you can change directly from ApplicationProperties.robot this will centralized it resulting in maintainability and clean code practice.

Lets improve the code around Test case . Currently we have only one Test file LoginTest.robot and 1 test cases in it Demo Test lets assume we have multiple test cases and for each of test you need to open browser navigate to website and close the browser that means you will be calling following

```
Open the browser in
Close All Browser Window
```

In each test case. As per best practices we use Test Setup and Test Tear down functions

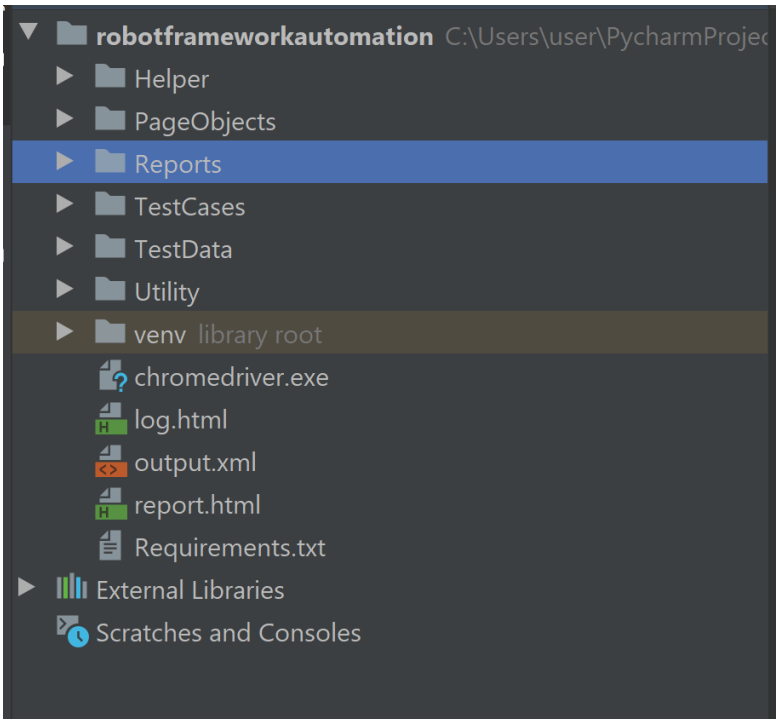


You can see above moved those lines out from Demo Test and placed under Settings

In short, a **test setup** is something that is executed before a **test** case, and **a test teardown** is executed after a **test** case. In Robot Framework **setups** and **teardowns** are just normal keywords with possible arguments. Setup and teardown are always a single keyword.

Update the LoginTest.robot and save.

Lets create directory where we will store the test reports with name Reports . Right click project->New->Directory . Set name to Reports



Now lets run using the following command

robot -d “Reports” TestCases/*.robot

It will save test reports under this folder.

```
(venv) C:\Users\user\PycharmProjects\robotframeworkautomation>robot -d "Reports" TestCases/*.robot
==============================================================================
LoginTest
==============================================================================
Demo Test
```

Once execution completes it will show path for Report

```
==============================================================================
Output:  C:\Users\user\PycharmProjects\robotframeworkautomation\Reports\output.xml
Log:     C:\Users\user\PycharmProjects\robotframeworkautomation\Reports\log.html
Report:  C:\Users\user\PycharmProjects\robotframeworkautomation\Reports\report.html

(venv) C:\Users\user\PycharmProjects\robotframeworkautomation>
```

Now we will add some more methods in SeleniumKeywords.robot, Pleas see below code. Remove the old code and replace with this one

```
*** Settings ***
Library    Selenium2Library

*** Keywords ***
Clear Input Field
   [Arguments]   ${locator}
   Wait For Element Present   ${locator}  5   Element not found in 5 seconds
   Clear Element Text   ${locator}
   ${value}=    Get Element Attribute   ${locator}    value
   ${backspaces count}=   Get Length    ${value}
   Run Keyword If   """${value}""" != ''
   ...    Repeat Keyword  ${backspaces count +1}  Press Key  ${locator}   \\08

Set Value For Input Field
   [Arguments]   ${locator}   ${inputval}
   sleep   2s
   Clear Input Field   ${locator}
   Input Text   ${locator}   ${inputval}

Click On Page Element
   [Arguments]   ${locator}
   Wait For Element Present   ${locator}  5   Element not found in 5 seconds
   Click Element   ${locator}

Wait For Page Load
   [Arguments]   ${pgloadlocator}
   Wait Until Page Contains Element   ${pgloadlocator}
   Sleep   5s

Wait For Element Present
   [Arguments]   ${locator}   ${timeout}   ${err_msg}
   #Wait For Element Present   ${locator}  5   Element not found in 5 seconds
   Wait Until Page Contains Element   ${locator}   ${timeout}   ${err_msg}

Get Attribute Value
   [Arguments]   ${locator}   ${attribute}
   #Wait For Element Present   ${locator}  5   Element not found in 5 seconds
   ${string}=   Get Element Attribute   ${locator}   ${attribute}
   log to console   \n get attribute ${string}.\n
   [Return]   ${string}

Back On Screen
   Go Back

Wait For Text Present
   [Arguments]   ${text}   ${seconds}   ${message}
   Wait Until Page Contains   ${text}   ${seconds}   ${message}

Verify Text Present
   [Arguments]   ${text}
   Page Should Contain   ${text}
```

```
Verify Element Present
    [Arguments]    ${locator}
    #Wait For Element Present    ${locator}  5    Element not found in 5 seconds
    Page Should Contain Element    ${locator}

Generate Random String With Defined Size
    [Arguments]    ${size}    ${type}
    ${str}=    Generate Random String    ${size}    ${type}
    [Return]    ${str}

Generate Random Number With Defined Size
    [Arguments]    ${size}    ${type}
    ${num}=    Generate Random String    ${size}    ${type}
    [Return]    ${num}

Page Scroll
    [Arguments]    ${st_loc}    ${en_loc}    ${duration}
    Scroll    ${st_loc}    ${en_loc}
    Sleep    ${duration}

Match Value
    [Arguments]    ${locator}    ${attr_name}    ${match_pattern}
    #Wait For Element Present    ${locator}  5    Element not found in 5 seconds
    Element Attribute Should Match    ${locator}    ${attr_name}    ${match_pattern}

Verify Text Not Present
    [Arguments]    ${text}
    Page Should Not Contain    ${text}

Select List Option By visible Text
    [Arguments]    ${locator}    ${value}
    Wait For Element Present    ${locator}  5    Element not found in 5 seconds
    Select From List By Label    ${locator}    ${value}

Verify Page title
    [Arguments]    ${title}
    Title Should Be    ${title}

Navigate to the page
    [Arguments]    ${URL}
    Go To    ${URL}

Verify Element Displayed
    [Arguments]    ${locator}
    Wait For Element Present    ${locator}  5    Element not found in 5 seconds
    Element Should Be Visible    ${locator}

Verify Element not displayed
    [Arguments]    ${locator}
    Element Should Not Be Visible    ${locator}

Verify Current URL should contains
    [Arguments]    ${URL}
    Location Should Contain    ${URL}
```

```
Static wait for
    [Arguments]    ${Second}
    Sleep    ${Second}s

Get all element count
    [Arguments]    ${locator}
    ${num}=    Get Element Count    ${locator}
    [Return]    ${num}

Verify both values are equal
    [Arguments]    ${val1}  ${val2}
    Should Be Equal    ${val1}    ${val2}

Switch to window
    [Arguments]    ${type}
    Select Window  ${type}

Close current window
    Close Window

Get Current URL
    return from keyword    Get Location

Press keyboard key
    [Arguments]    ${locator}   ${Key}
    Wait For Element Present    ${locator}  5    Element not found in 5 seconds
    Press Key  ${locator}   \\08

Mouse Over on the element
    [Arguments]    ${locator}
    Wait For Element Present    ${locator}  5    Element not found in 5 seconds
    Mouse Over  ${locator}

URL Should Contain
    [Arguments]    ${expected}
    Location Should Contain    ${expected}

Scroll for element
    [Arguments]    ${locator}
    Scroll Element Into View    ${locator}

Scroll at the bottom of the page
    Execute Javascript    window.scrollTo(0,document.body.scrollHeight)

Scroll at the top of the page
    Execute Javascript    window.scrollTo(0,0)
```

Update SeleniumKeywords.robot file and Save.

Let see some of them, Verify Page Title function its calling Title Should be SeleniumLibrary keyword

```
Verify Page title
    [Arguments]    ${title}
    Title Should Be    ${title}
```

Which takes parameter title from user is same page title.

You can find this on http://robotframework.org/SeleniumLibrary/SeleniumLibrary.html

Title Should Be	*title*, *message: NoneType=None*	Verifies that the current page title equals `title`. The `message` argument can be used to override the default error message. `message` argument is new in SeleniumLibrary 3.1.

Lets look at one more function Verify Element Displayed

```
Verify Element Displayed
    [Arguments]    ${locator}
    Wait For Element Present    ${locator}  5    Element not found in 5 seconds
    Element Should Be Visible    ${locator}
```

Here Wait for Element Present is another function defined in SeleniumKeywords.robot as you can see below which is calling Wait Until Page Contains Element SeleniumLibrary keyword

```
Wait For Element Present
    [Arguments]    ${locator}    ${timeout}    ${err_msg}
    #Wait For Element Present    ${locator}  5    Element not found in 5 seconds
    Wait Until Page Contains Element    ${locator}    ${timeout}    ${err_msg}
```

It can be found here

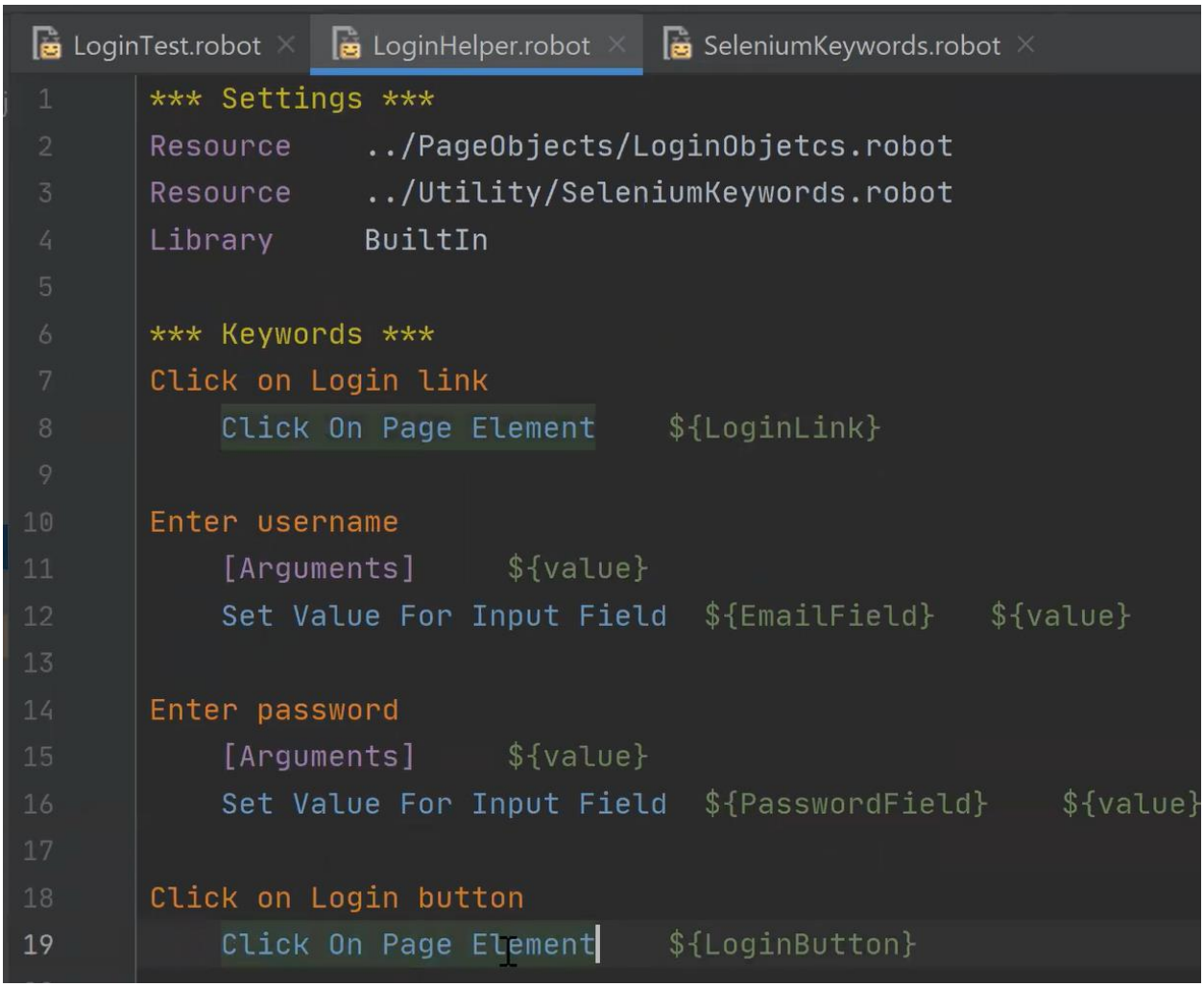
Wait Until Page Contains	*text*, *timeout: NoneType=None*, *error: NoneType=None*	Waits until `text` appears on the current page. Fails if `timeout` expires before the text appears. See the *Timeouts* section for more information about using timeouts and their default value. `error` can be used to override the default error message.

Now you see we from Verify Element Present we are passing ${locator} , 5 and Element not found in 5 seconds as parameter into this function Wait For Element Present which than is passed to Wait Until Page Contains Element $(locator) ${timeour} and ${err_msg} .

Next in Verify Element Displayed we have Element Should be Visible ${locator} which is SeleniumLibrary keyword which takes parameter ${locator} . It verifies that the element identified by locator is visible. You can find it here http://robotframework.org/SeleniumLibrary/SeleniumLibrary.html#Element%20Should%20Be%20Visible

Element Should Be Visible	*locator, message: NoneType=None*	Verifies that the element identified by `locator` is visible. Herein, visible means that the element is logically visible, not optically visible in the current browser viewport. For example, an element that carries `display:none` is not logically visible, so using this keyword on that element would fail. See the *Locating elements* section for details about the locator syntax.

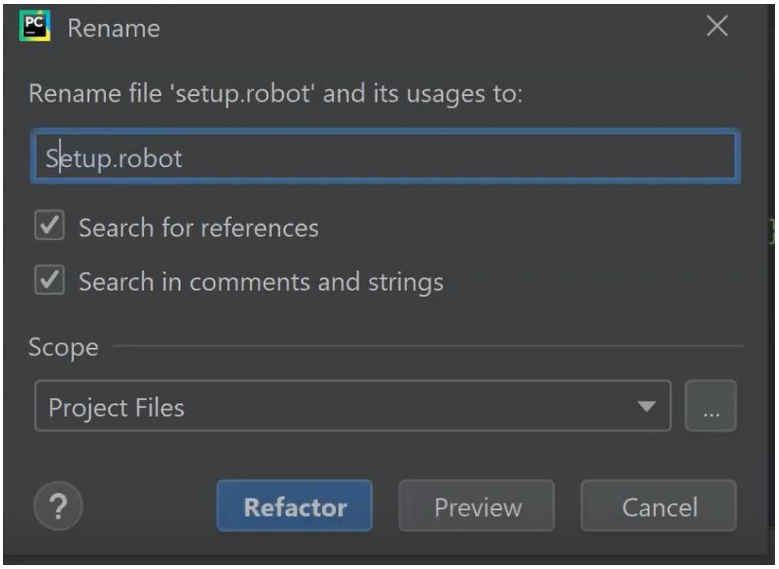
Since we updated the SeleniumKeywords.robot we will update LoginHelper.robot functions. I have replaced line 12 and 16 in below screenshot . Similarliy line 8 and 9 with new functions headers



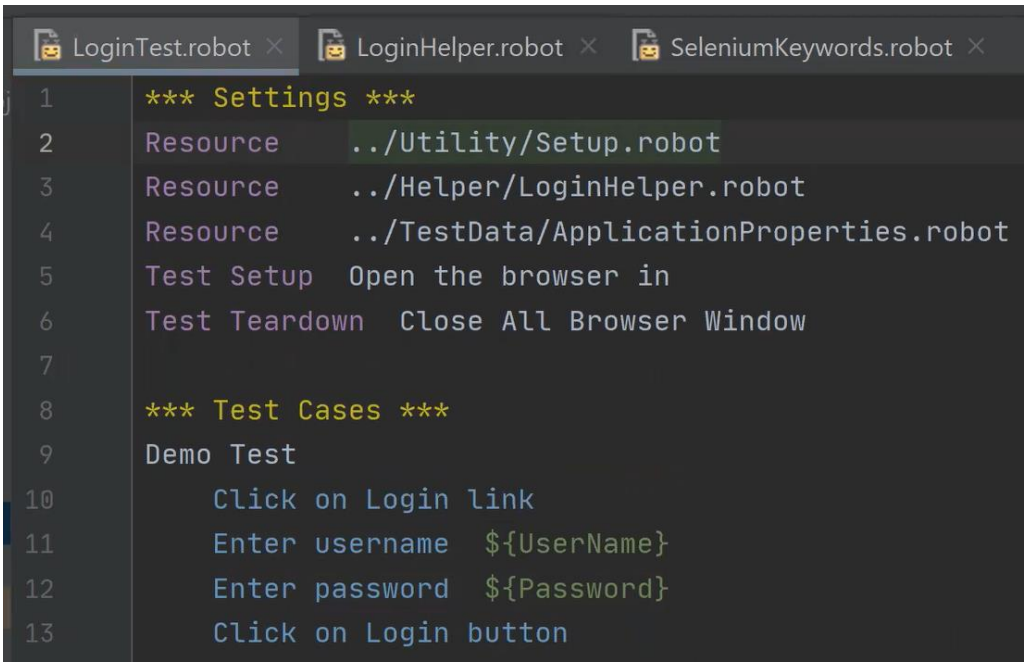
Code:

```
*** Settings ***
Resource    ../PageObjects/LoginObjetcs.robot
Resource    ../Utility/SeleniumKeywords.robot
Library     BuiltIn

*** Keywords ***
Click on Login link
    Click On Page Element    ${LoginLink}

Enter username
    [Arguments]    ${value}
    Sleep  2s
    Set Value For Input Field  ${EmailField}   ${value}

Enter password
    [Arguments]    ${value}
    Set Value For Input Field  ${PasswordField}    ${value}

Click on Login button
    Click On Page Element    ${LoginButton}
```

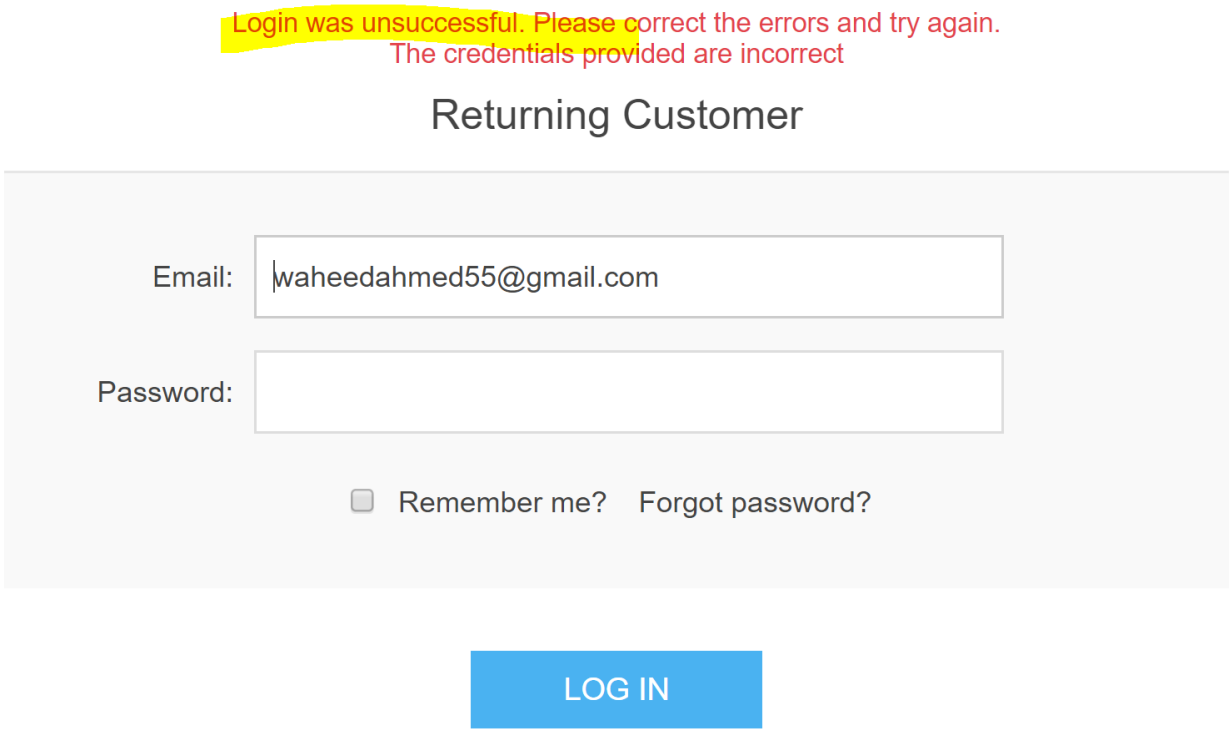
Before we proceed further change the file name from setup.robot to Setup.robot and click Refactor button



After this name change go to LoginTest.robot and change it from Resource where we have imported it as you can see on line 2.



Now let say we want to verify error message which gets displayed post entering wrong password



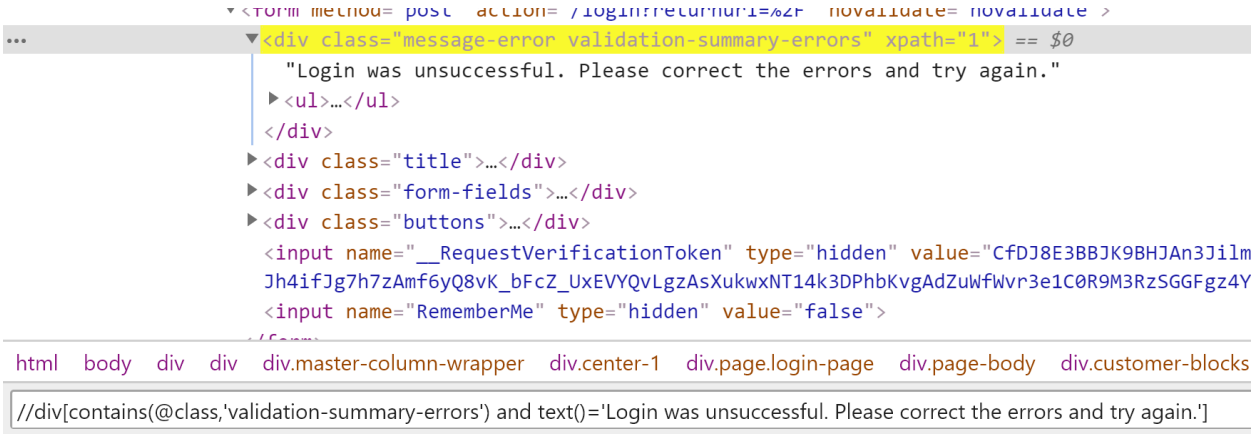
So we will first try to locate the path of that error message

I was able to locate the element using xpath

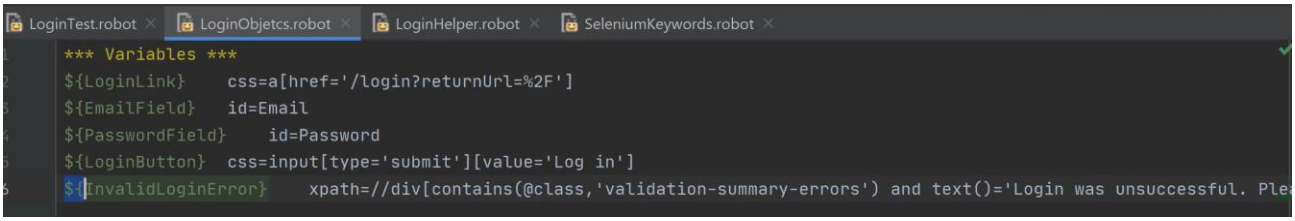
//div[contains(@class,'validation-summary-errors') and text()='Login was unsuccessful. Please correct the errors and try again.']

Syntax : tag[contains(@class),'name of class' and text()='text you want to validate within the tag']

In our case tag is div and class 'validation-summary-errors' and text is 'Login was unsuccessful. Please correct the errors and try again.



Thus we will add this locator to LoginPageObject.robot, as you can see below:



Now we will write new function in LoginHelper.robot which will help us verifying the error message. In that function we will call function from SelelniumKeywords.robot which will use SeleniumLibrary keyword to locate the object which displays error message.

Lets see from code, go to LoginHelper.robot

First we will wait for element to be present before we validate so we called function Wait For Element Present from SeleniumKeywords.robot and passed three paramters ${InvalidLoginError} , 5 , and Error not displayed

```
Verify error message displayed
    Wait For Element Present    ${InvalidLoginError}    5    Error not displayed
```

Lets look at that function Wait For Element Present in SeleniumKeywords.robot now this function calls SeleniumLibrary keyword Wait Until Page Contains Element and needs ${locator} which we are passing of ${InvalidLoginError} and ${timeout} which we are passing as 5 and ${err_msg} which we are passing as Error not displayed

```
Wait For Element Present
    [Arguments]    ${locator}    ${timeout}    ${err_msg}
    #Wait For Element Present    ${locator}  5    Element not found in 5 seconds
    Wait Until Page Contains Element    ${locator}    ${timeout}    ${err_msg}
```

You can more information here
http://robotframework.org/SeleniumLibrary/SeleniumLibrary.html#Wait%20Until%20Page%20Contains%20Element

Wait Until Page Contains Element	*locator*, *timeout: NoneType=None*, *error: NoneType=None*, *limit: NoneType=None*	Waits until the element `locator` appears on the current page. Fails if `timeout` expires before the element appears. See the *Timeouts* section for more information about using timeouts and their default value and the *Locating elements* section for details about the locator syntax. `error` can be used to override the default error message. The `limit` argument can used to define how many elements the page should contain. When `limit` is *None* (default) page can contain one or more elements. When limit is a number, page must contain same number of elements. `limit` is new in SeleniumLibrary 4.4

Now we will call that function Verify error message displayed in LoginTest.robot as step in the test case.

LoginTest.robot | LoginObjetcs.robot | LoginHelper.robot | Seleniu

```
1   *** Settings ***
2   Resource    ../Utility/Setup.robot
3   Resource    ../Helper/LoginHelper.robot
4   Resource    ../TestData/ApplicationProperties.robot
5   Test Setup  Open the browser in
6   Test Teardown  Close All Browser Window
7
8   *** Test Cases ***
9   Demo Test
10      Click on Login link
11      Enter username  ${UserName}
12      Enter password  ${Password}
13      Click on Login button
14      Verify error message displayed
```

Now save everything. Lets run

```
(venv) C:\Users\user\PycharmProjects\robotframeworkautomation>robot -d "Reports" TestCases/*.robot
```

You will see browser will launch go to login page and enter user name and password , since password is invalid it will display error message and it will get verified once done will close browser

```
LoginTest                                                             | PASS |
1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
==============================================================================
Output:  C:\Users\user\PycharmProjects\robotframeworkautomation\Reports\output.xml
Log:     C:\Users\user\PycharmProjects\robotframeworkautomation\Reports\log.html
Report:  C:\Users\user\PycharmProjects\robotframeworkautomation\Reports\report.html
```

You can see the report below all test steps in Demo Test case passed. One thing I wanted to show here is now you can ${UserName} and ${Password} instead of actual email address and password since we have stored that in ApplicationProperties.robot

LoginTest Log

Generated
20200506 14:11:36 UTC-04:00
17 seconds ago

Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	1	1	0	00:00:11	
All Tests	1	1	0	00:00:11	

Statistics by Tag	Total	Pass	Fail	Elapsed	Pass / Fail
No Tags					

Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
LoginTest	1	1	0	00:00:12	

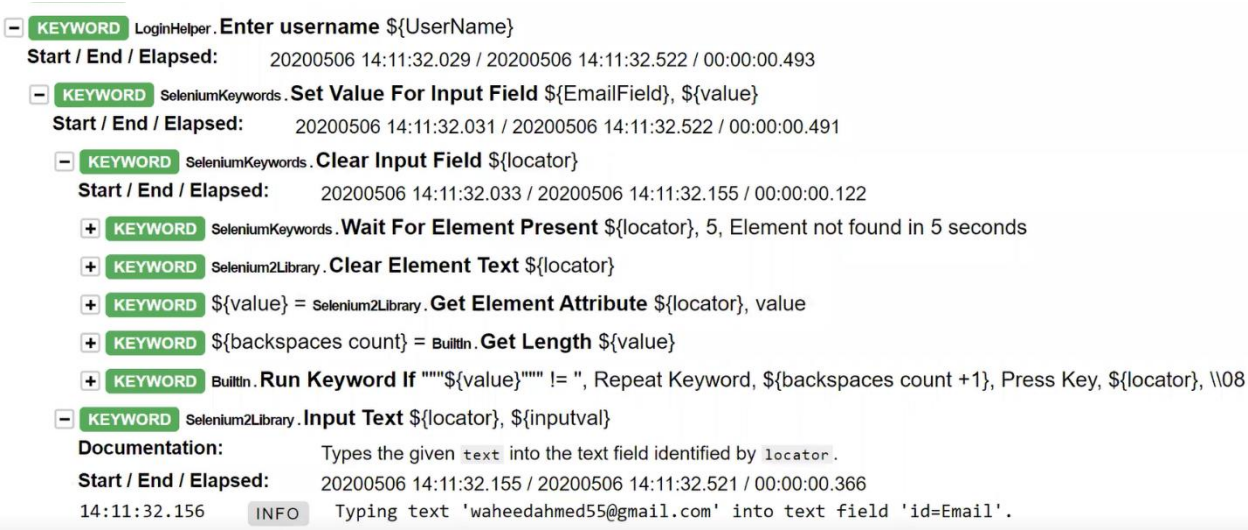
Test Execution Log

SUITE LoginTest
Full Name: LoginTest
Source: C:\Users\user\PycharmProjects\robotframeworkautomation\TestCases\LoginTest.robot
Start / End / Elapsed: 20200506 14:11:24.499 / 20200506 14:11:36.143 / 00:00:11.644
Status: 1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed

TEST Demo Test
LoginTest.Demo Test
LoginTest.Demo Test
Start / End / Elapsed: 20200506 14:11:24.836 / 20200506 14:11:36.140 / 00:00:11.304
Status: PASS (critical)

SETUP Setup . Open the browser in
KEYWORD LoginHelper . Click on Login link
KEYWORD LoginHelper . Enter username ${UserName}
KEYWORD LoginHelper . Enter password ${Password}
KEYWORD LoginHelper . Click on Login button
KEYWORD LoginHelper . Verify error message displayed

Now if you expand Enter username step you will email address



Let say we want to verify exact error message text only 'Login was unsuccessful. Please correct the errors and try again.' Lets add locator for it in LoginObjects.robot

```
*** Variables ***
${LoginLink}    css=a[href='/login?returnUrl=%2F']
${EmailField}   id=Email
${PasswordField}    id=Password
${LoginButton}  css=input[type='submit'][value='Log in']
${InvalidLoginError}    xpath=//div[contains(@class,'validation-summary-errors') and text()='Login was unsuccessful. Please correct the errors and try again.']
${InvaliLoginText}      Login was unsuccessful. Please correct the errors and try again.
```

We added it in ${InvalidLoginText} . Now lets add step in LoginHelper.robots function

```
Verify error message displayed
    Wait For Element Present    ${InvalidLoginError}    5    Error message not displayed
    Verify Text Present     ${InvaliLoginText}
```

So we are calling function Verify Text Present from SeleniumKeywords.robot and passing the ${InvalidLoginText} as paramter lets see in SeleniumKeywords.robot

```
Verify Text Present
    [Arguments]     ${text}
    Page Should Contain     ${text}
```

So its calling Page Should Contain SeleniumLibrary keyword which needs parameter text which we are passing from LoginHelper .

Here you can find more information
http://robotframework.org/SeleniumLibrary/SeleniumLibrary.html#Page%20Should%20Contain

Page Should Contain	*text*, *loglevel=TRACE*	Verifies that current page contains `text`. If this keyword fails, it automatically logs the page source using the log level specified with the optional `loglevel` argument. Valid log levels are `DEBUG`, `INFO` (default), `WARN`, and `NONE`. If the log level is `NONE` or below the current active log level the source will not be logged.

Save all. Now lets run the test

```
(venv) C:\Users\user\PycharmProjects\robotframeworkautomation>robot -d "Reports" TestCases/*.robot
==============================================================================
LoginTest
==============================================================================
Demo Test
```

In report, you can see error message.

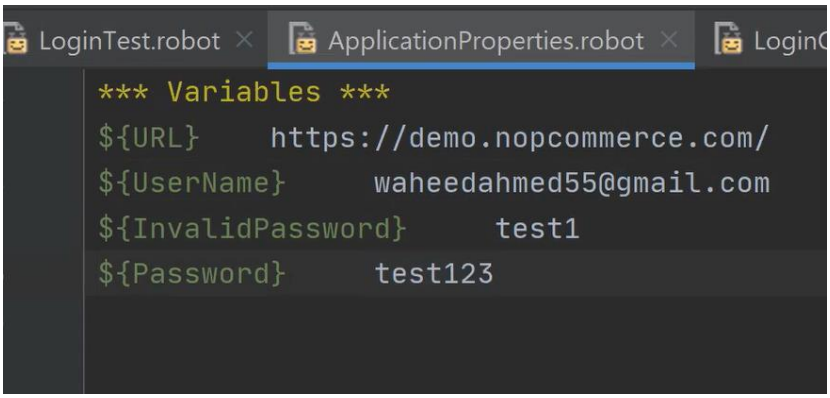
```
KEYWORD LoginHelper . Verify error message displayed
  Start / End / Elapsed:  20200511 14:38:56.593 / 20200511 14:38:56.637 / 00:00:00.044
  KEYWORD SeleniumKeywords . Wait For Element Present ${InvalidLoginError}, 5, Error message not displayed
  KEYWORD SeleniumKeywords . Verify Text Present ${InvaliLoginText}
    Start / End / Elapsed:  20200511 14:38:56.620 / 20200511 14:38:56.637 / 00:00:00.017
    KEYWORD Selenium2Library . Page Should Contain ${text}
      Documentation:  Verifies that current page contains text .
      Start / End / Elapsed:  20200511 14:38:56.620 / 20200511 14:38:56.637 / 00:00:00.017
      14:38:56.636  INFO  Current page contains text 'Login was unsuccessful. Please correct the errors and try again.'.
```

Lets rename this test case from Demo Test to Invalid Login in LoginTest.robot

LoginTest.robot | LoginObjetcs.robot | LoginHelper.robot | Sele

```
1  *** Settings ***
2  Resource    ../Utility/Setup.robot
3  Resource    ../Helper/LoginHelper.robot
4  Resource    ../TestData/ApplicationProperties.robot
5  Test Setup  Open the browser in
6  Test Teardown  Close All Browser Window
7
8  *** Test Cases ***
9  Invalid Login
10     Click on Login link
11     Enter username  ${UserName}
12     Enter password  ${Password}
13     Click on Login button
14     Verify error message displayed
```

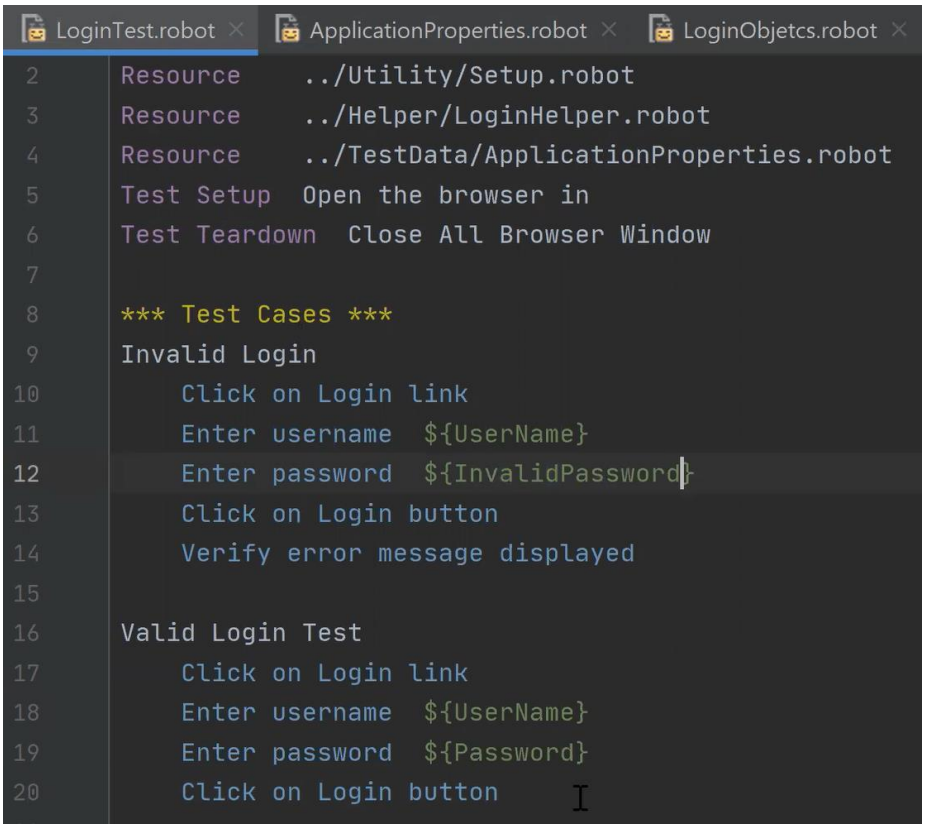
Now lets create another test case in LoginTest.robot Valid Login Test . For that lets update the ApplicationProperties.robot with valid and invlaid password. As you can see below :



Code:

```
*** Variables ***
${URL}  https://demo.nopcommerce.com/
${UserName}  waheedahmed55@gmail.com
${InvalidPassword}  test1
${Password}  test123
```

Now we can go back to LoginTest.robot and write steps for Valid Login Test case steps will be same expect it will use valid password. As you can see below:



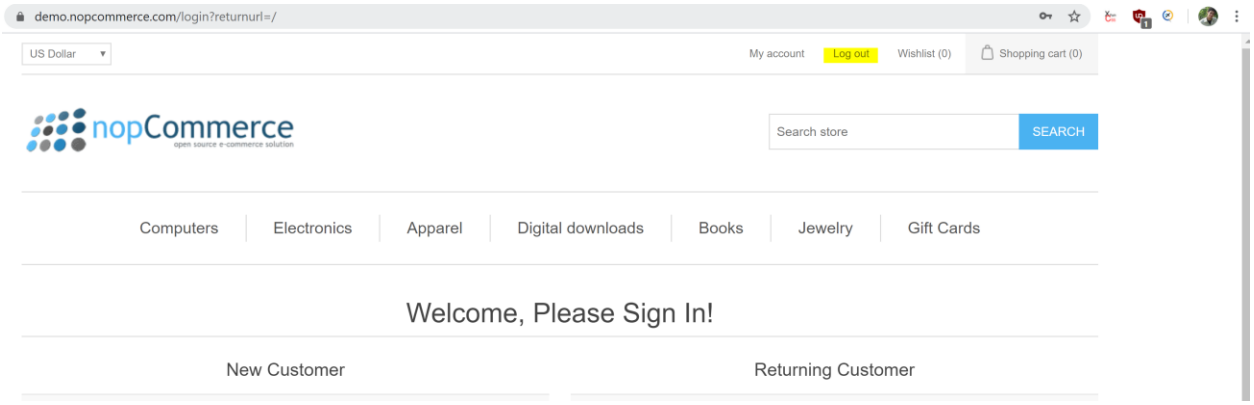
Now let say after successful login I want to add a step in Valid Login Test to verify I am on Home page.

Lets call step Verify User is at Home Page , as you can see in below code

Code: LoginTest.robot

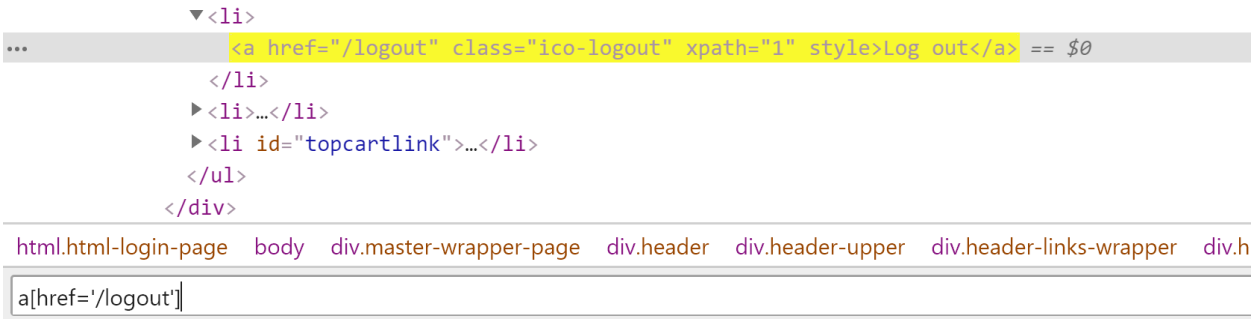
```
*** Settings ***
Resource    ../Utility/Setup.robot
Resource    ../Helper/LoginHelper.robot
Resource    ../TestData/ApplicationProperties.robot
Test Setup  Open the browser in
Test Teardown  Close All Browser Window

*** Test Cases ***
Invalid Login
    Click on Login link
    Enter username  ${UserName}
    Enter password  ${InvalidPassword}
    Click on Login button
    Verify error message displayed

Valid Login Test
    Click on Login link
    Enter username  ${UserName}
    Enter password  ${Password}
    Click on Login button
    Verify User is at Home page
```

Now lets implement this step in LoginHelper.robot. After successful login I will verify Logout is present on Home page



Before we implement that function lets find locator of Logout.

I was able to locate the Logout using css



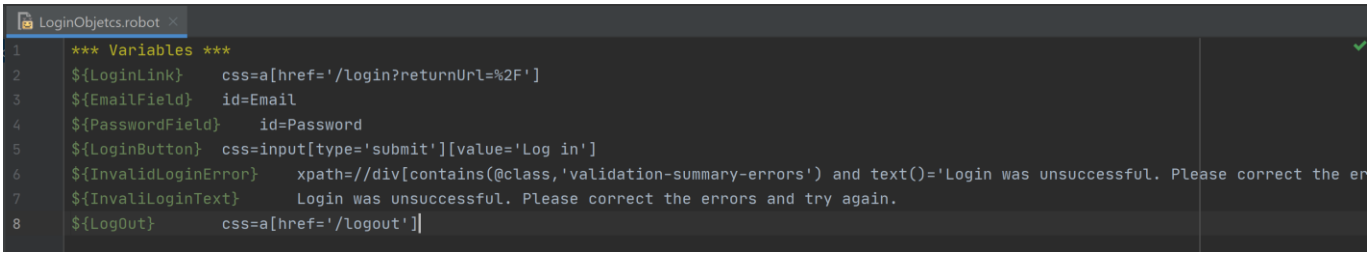
Syntax: tag[attribute='value']

Tag = a

Attribute = href

Value = '/logout'

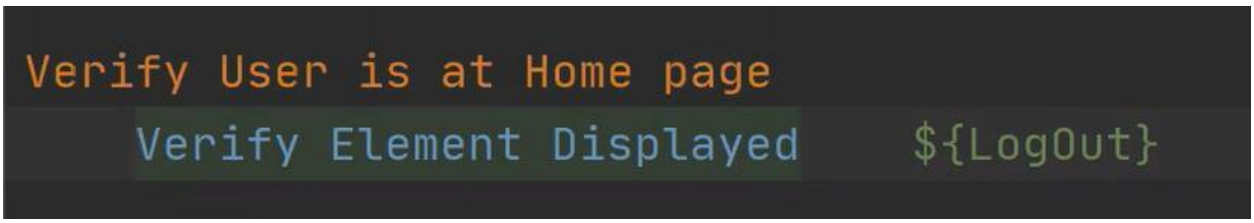
So we will create the locator for Logout in LoginObjects.robot as you can see below:



Code: LoginObjects.robot

```
*** Variables ***
${LoginLink}    css=a[href='/login?returnUrl=%2F']
${EmailField}   id=Email
${PasswordField}    id=Password
${LoginButton}  css=input[type='submit'][value='Log in']
${InvalidLoginError}    xpath=//div[contains(@class,'validation-summary-errors') and text()='Login was unsuccessful. Please correct the errors and try again.']
${InvaliLoginText}      Login was unsuccessful. Please correct the errors and try again.
${LogOut}       css=a[href='/logout']
```

Now lets implement Verify User is at Home page in LoginHelper.robot class as shown below



Code: LoginHelper.robot

```
*** Settings ***
Resource    ../PageObjects/LoginObjetcs.robot
Resource    ../Utility/SeleniumKeywords.robot
Library     BuiltIn

*** Keywords ***
Click on Login link
    Click On Page Element    ${LoginLink}

Enter username
    [Arguments]    ${value}
    Sleep  2s
    Set Value For Input Field  ${EmailField}  ${value}

Enter password
    [Arguments]    ${value}
    Set Value For Input Field  ${PasswordField}   ${value}

Click on Login button
    Click On Page Element    ${LoginButton}

Verify error message displayed
    Wait For Element Present    ${InvalidLoginError}    5    Error message not displayed
    Verify Text Present    ${InvaliLoginText}

Verify User is at Home page
    Verify Element Displayed    ${LogOut}
```

I am calling Verify Element Dsiplayed function of SeleniumKeywords and passing locator of Logout to it

Lets look at Verify Element Displayed

```
Verify Element Displayed
    [Arguments]    ${locator}
    Wait For Element Present    ${locator}  5    Element not found in 5 seconds
    Element Should Be Visible    ${locator}
```

Now here we see call to Wait For Element Present function which is implmeneted in same SeleniumKeywords.robot. We are passing locator , 5 and Element not found in 5 seconds .

```
Wait For Element Present
    [Arguments]    ${locator}    ${timeout}    ${err_msg}
    #Wait For Element Present    ${locator}  5    Element not found in 5 seconds
    Wait Until Page Contains Element    ${locator}    ${timeout}    ${err_msg}
```

Here Wait Until Page Contains Element is SeleniumLibrary keyword which takes locator , timeout and err_msg as parameter which we are passing to it.

You can find information here
http://robotframework.org/SeleniumLibrary/SeleniumLibrary.html#Wait%20Until%20Page%20Contains

Wait Until Page Contains	*text*, *timeout: NoneType=None*, *error: NoneType=None*	Waits until `text` appears on the current page. Fails if `timeout` expires before the text appears. See the *Timeouts* section for more information about using timeouts and their default value. `error` can be used to override the default error message.

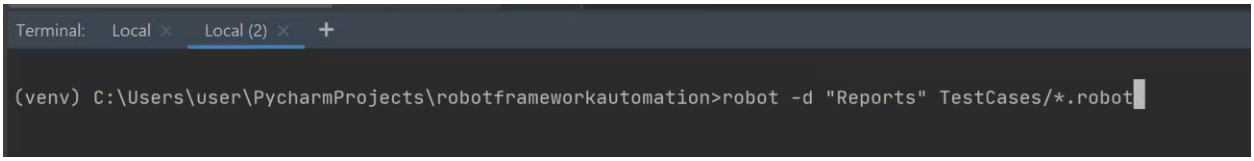
Now in Verify Element Displayed next we see is Element Should Be Visible which is SeleniumLibrary keyword it takes locator of logout .

You can find information here:
http://robotframework.org/SeleniumLibrary/SeleniumLibrary.html#Element%20Should%20Be%20Visible

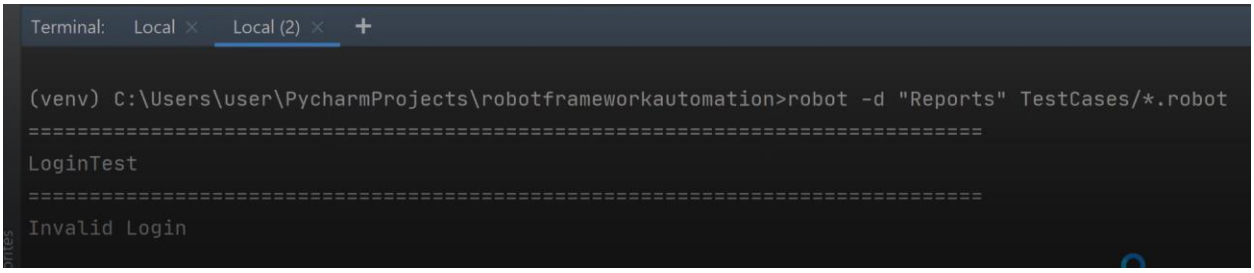
Element Should Be Visible	*locator*, *message: NoneType=None*	Verifies that the element identified by `locator` is visible. Herein, visible means that the element is logically visible, not optically visible in the current browser viewport. For example, an element that carries `display:none` is not logically visible, so using this keyword on that element would fail. See the *Locating elements* section for details about the locator syntax.

Save all the files and run the tests.

This time it will execute both test cases

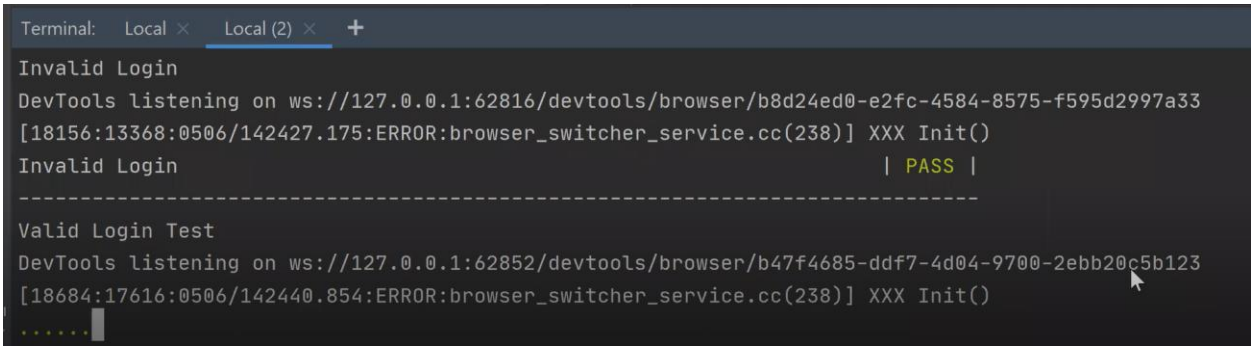


First you will see Invalid Login test

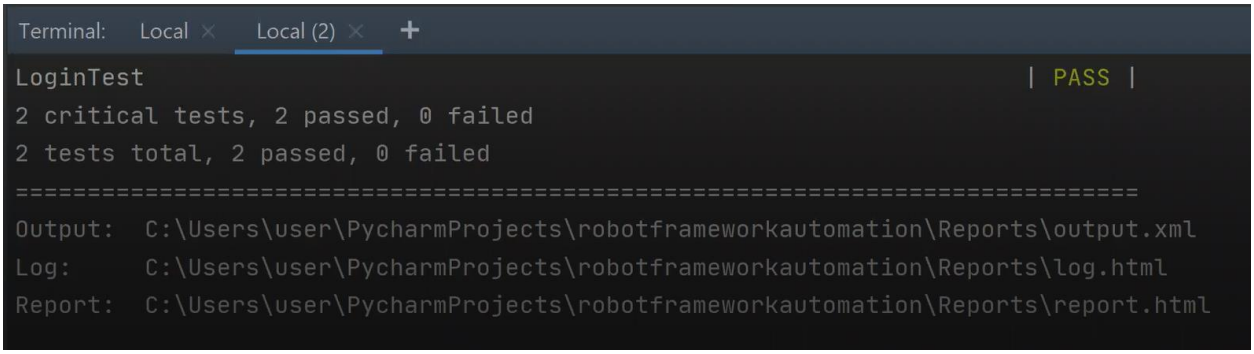


Note: Since we have Test Setup and Test Tear Down so before each test it will launch browser and navigate to website and after that particularly is completed it will close all browser as its part of Test Tear Down.

Now you will see second test executed which is Valid Login Test



Finally you will see Test result



Now what if I want to verify URL after login is successful. The SeleniumLibrary keyword whichI want to use to verify URL isn't implemented in our SeleniumKeywords.robot.

Here is keyword I will be using from SeleniumLibrary you can find more information here

http://robotframework.org/SeleniumLibrary/SeleniumLibrary.html#Location%20Should%20Contain

Location Should Contain	*expected*, *message: NoneType=None*	Verifies that the current URL contains `expected`. The `expected` argument contains the expected value in url. The `message` argument can be used to override the default error message. `message` argument is new in SeleniumLibrary 3.2.0.

Now this keyword takes argument expected which we will pass and it will verify against current URL of browser.

Lets implement it in SeleniumKeywords.robot

```
URL Should Contain
    [Arguments]    ${expected}
    Location Should Contain    ${expected}
```

As you can see I wrote function URL Should Contain in which I am calling Location Should Contain with arugment ${expected}.

Please append this method at the end of SeleniumKeywords.robot rest of file remain as it is.

Now lets use this function in our LoginHelper.robot . What I will do is I will call this function in Verify user is at home page function like this

```
Verify User is at Home page
    Verify Element Displayed    ${LogOut}
    URL Should Contain  demo.nopcommerce.com
```

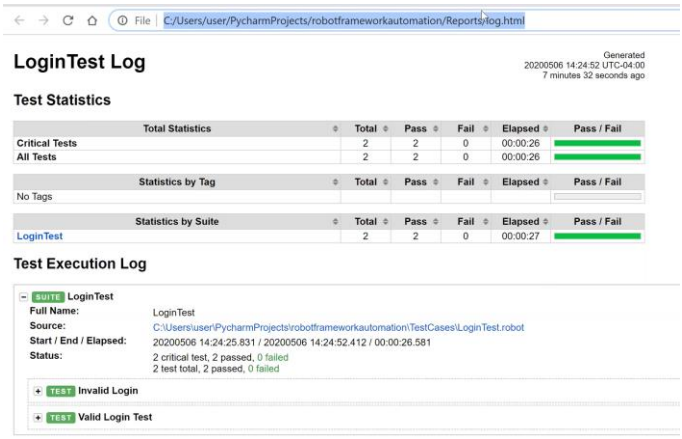
See I called the function URL Should Contain and I passed the expected URL which is demo.nocommerce.com as parameter.

Code: LoginHelper.robot

```
*** Settings ***
Resource    ../PageObjects/LoginObjetcs.robot
Resource    ../Utility/SeleniumKeywords.robot
Library     BuiltIn

*** Keywords ***
Click on Login link
    Click On Page Element    ${LoginLink}

Enter username
    [Arguments]    ${value}
    Sleep  2s
    Set Value For Input Field  ${EmailField}  ${value}

Enter password
    [Arguments]    ${value}
    Set Value For Input Field  ${PasswordField}  ${value}

Click on Login button
    Click On Page Element    ${LoginButton}

Verify error message displayed
    Wait For Element Present    ${InvalidLoginError}    5    Error message not displayed
    Verify Text Present    ${InvaliLoginText}

Verify User is at Home page
    Verify Element Displayed    ${LogOut}
    URL Should Contain  demo.nopcommerce.com
```

Save all files.

Run the test and once both test cases are executed launch the report, you will see both tests executed.

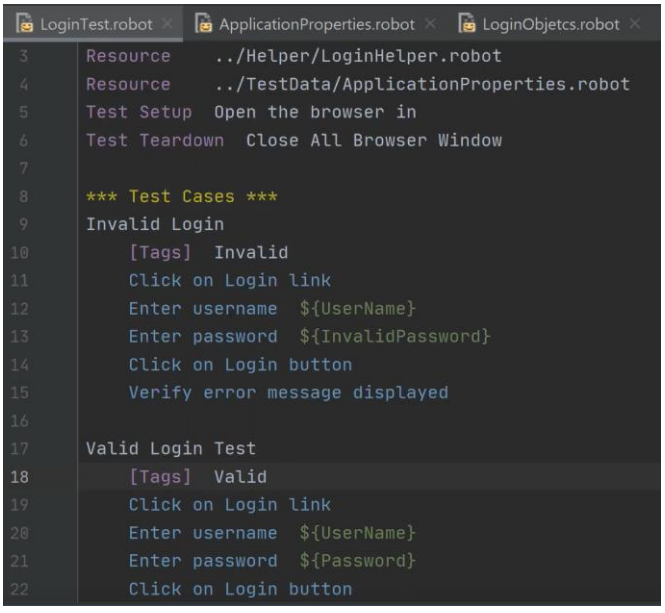


Now let see how we can control execution of test cases. For example you want to run only 1 not the other test case in this case we will Tags.

Using tags in Robot Framework is a simple, yet powerful mechanism for classifying test cases. Tags are free text and they can be used at least for the following purposes:

- Tags are shown in test reports, logs and, of course, in the test data, so they provide metadata to test cases.
- Statistics about test cases (total, passed, failed are automatically collected based on tags).
- With tags, you can **include or exclude** test cases to be executed.
- With tags, you can specify which test cases are considered critical.

We will add Tags in LoginTest.robot. As you can see in below screenshot I added Tag Invalid for first test case and valid for 2nd



Code: LoginTest.robot

```
*** Settings ***
Resource    ../Utility/Setup.robot
Resource    ../Helper/LoginHelper.robot
Resource    ../TestData/ApplicationProperties.robot
Test Setup  Open the browser in
Test Teardown  Close All Browser Window

*** Test Cases ***
Invalid Login
    [Tags]  Invalid
    Click on Login link
    Enter username  ${UserName}
    Enter password  ${InvalidPassword}
    Click on Login button
    Verify error message displayed

Valid Login Test
    [Tags]  Valid
    Click on Login link
    Enter username  ${UserName}
    Enter password  ${Password}
    Click on Login button
    Verify User is at Home page
```

Save all changes and lets run it using following command. Here you can see we are using –include “Valid” what this will do is it will find all test cases with Tag Valid will execute them rest ignore.

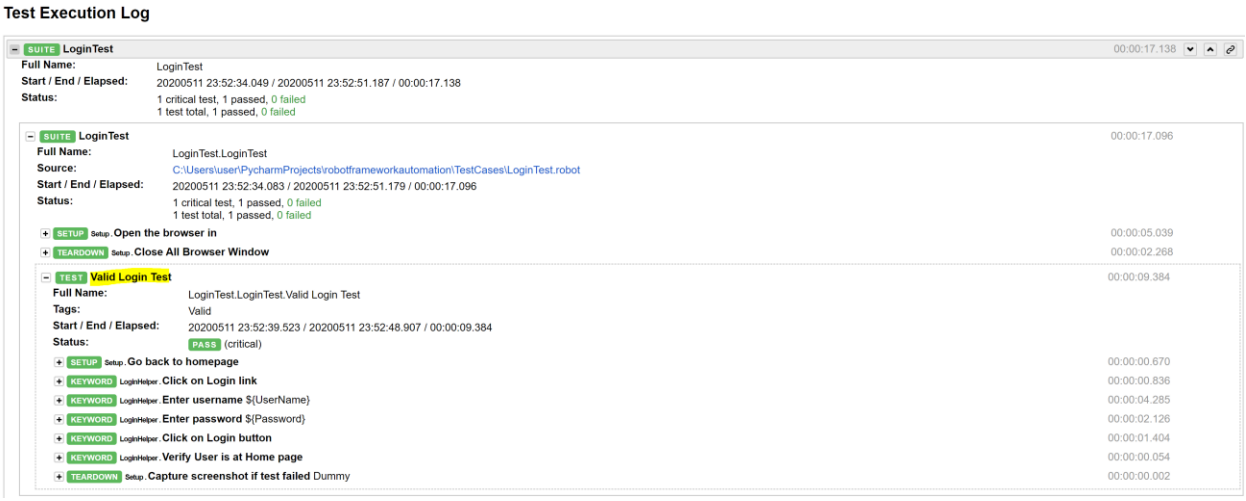
robot -d "Reports" --include "Valid" TestCases/*.robot

```
(venv) C:\Users\user\PycharmProjects\robotframeworkautomation>robot -d "Reports" --include "Valid" TestCases/*.robot
```

You will see only Valid Login Test got executed

```
Terminal:  Local  Local (2)  +
==============================================================================
LoginTest
==============================================================================
LoginTest.LoginTest
==============================================================================

DevTools listening on ws://127.0.0.1:50497/devtools/browser/f635cfdf-1748-4cb0-b747-511684f501ec
[4952:8132:0511/235235.410:ERROR:browser_switcher_service.cc(238)] XXX Init()
Valid Login Test                                                      | PASS |
------------------------------------------------------------------------------
```

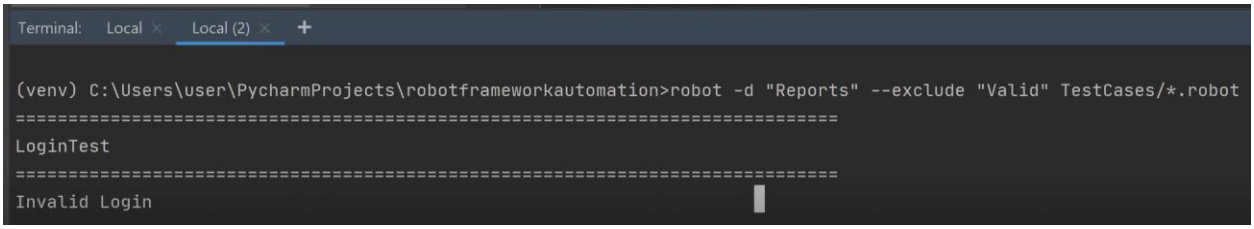
You will see in Test report as that it executed Valid Login Test



Similarly, we can exclude as well by running command like this

robot -d "Reports" --exclude "Valid" TestCases/*.robot

Now what this will do is it wont run all test cases which has Valid tag . As you run the command you will see It execute Invalid Login



So you will see it ran only Inavlid Login test

Error Handling/Screenshots

Now if any step fails wont execut next step and program stops which is good but we need to capture where it failed via screenshot. Let see how it will be done.

In order to capture screenshot we will implement function in Setup.robot which will capture screenshot and save in particular location.

1.Capture Screenshot If any test fails

Go to Setup.robot and create function Capture screenshot if test failed as you can see below

Code:

```
Capture screenshot if test failed
  [Arguments] ${FileName}
  ${now}   Evaluate
'{dt.second}{dt.minute}{dt.hour}{dt.day}{dt.month}{dt.year}'.format(dt=datetime.datetime.now())
modules=datetime
  Run Keyword If Test Failed     Capture Page Screenshot
${OUTPUTDIR}/ScreenShot/${FileName}_${now}.png
```

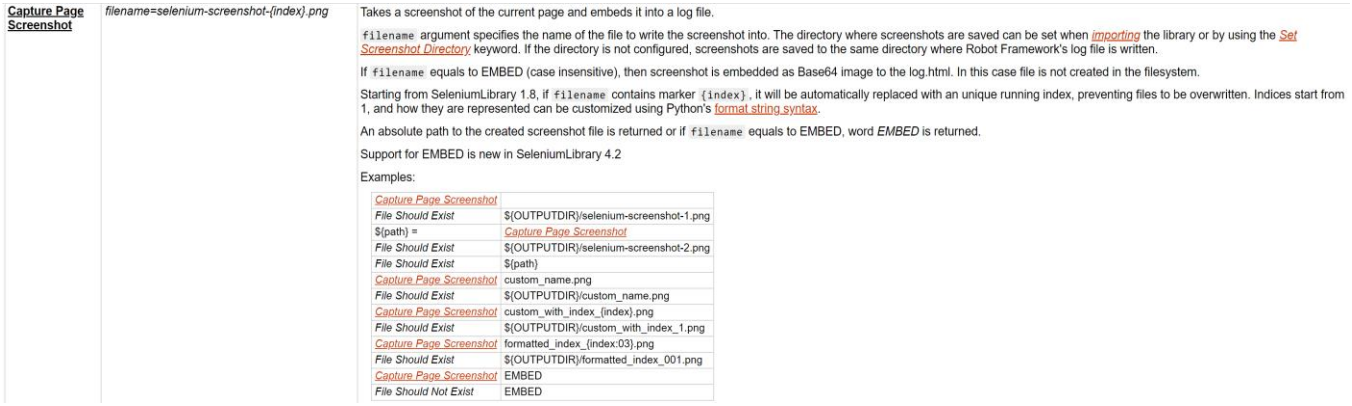
Here you can see we are using Run Keyword If Test Failed which is part of BuiltIn library what this keyword does is it will trigger another keyword if any testfails , you can see it will trigger Capture Page Screenshot which is SeleniumLibrary keyword.

Run Keyworf If Test Failed: you can find more information here https://robotframework.org/robotframework/latest/libraries/BuiltIn.html#Run%20Keyword%20If%20Test%20Failed

Run Keyword If Test Failed	*name*, **args*	Runs the given keyword with the given arguments, if the test failed. This keyword can only be used in a test teardown. Trying to use it anywhere else results in an error. Otherwise, this keyword works exactly like *Run Keyword*, see its documentation for more details.

This can only be used in Test Teardown .

Capture Page Screenshot: you can find more information here
http://robotframework.org/SeleniumLibrary/SeleniumLibrary.html#Capture%20Page%20Screenshot



Capture Page Screenshot | *filename=selenium-screenshot-{index}.png*

Takes a screenshot of the current page and embeds it into a log file.

`filename` argument specifies the name of the file to write the screenshot into. The directory where screenshots are saved can be set when *importing* the library or by using the *Set Screenshot Directory* keyword. If the directory is not configured, screenshots are saved to the same directory where Robot Framework's log file is written.

If `filename` equals to EMBED (case insensitive), then screenshot is embedded as Base64 image to the log.html. In this case file is not created in the filesystem.

Starting from SeleniumLibrary 1.8, if `filename` contains marker `{index}`, it will be automatically replaced with an unique running index, preventing files to be overwritten. Indices start from 1, and how they are represented can be customized using Python's format string syntax.

An absolute path to the created screenshot file is returned or if `filename` equals to EMBED, word *EMBED* is returned.

Support for EMBED is new in SeleniumLibrary 4.2

Examples:

Capture Page Screenshot	
File Should Exist	${OUTPUTDIR}/selenium-screenshot-1.png
${path} =	*Capture Page Screenshot*
File Should Exist	${OUTPUTDIR}/selenium-screenshot-2.png
File Should Exist	${path}
Capture Page Screenshot	custom_name.png
File Should Exist	${OUTPUTDIR}/custom_name.png
Capture Page Screenshot	custom_with_index_{index}.png
File Should Exist	${OUTPUTDIR}/custom_with_index_1.png
Capture Page Screenshot	formatted_index_{index:03}.png
File Should Exist	${OUTPUTDIR}/formatted_index_001.png
Capture Page Screenshot	EMBED
File Should Not Exist	EMBED

Lets go to LoginTest.robot . Now as we found Run Keyword If Test Failed works only in Test Teardown, however we have Test Teardown Close All Browser Window which is calling Close All Browser Window function of Setup.robot.

Now we will have to change the flow which means we will introduce Suite Teardown and Suite Setup.

Not only test cases but also test suites can have a setup and a teardown. A suite setup is executed before running any of the suite's test cases or child test suites, and a test teardown is executed after them. All test suites can have a setup and a teardown; with suites created from a directory they must be specified in a test suite initialization file.

Similarly as with test cases, a suite setup and teardown are keywords that may take arguments. They are defined in the Setting table with Suite Setup and Suite Teardown settings, respectively. Keyword names and possible arguments are located in the columns after the setting name.

If a suite setup fails, all test cases in it and its child test suites are immediately assigned a fail status and they are not actually executed. This makes suite setups ideal for checking preconditions that must be met before running test cases is possible.

A suite teardown is normally used for cleaning up after all the test cases have been executed. It is executed even if the setup of the same suite fails. If the suite teardown fails, all test cases in the suite are marked failed, regardless of their original execution status. Note that all the keywords in suite teardowns are executed even if one of them fails.

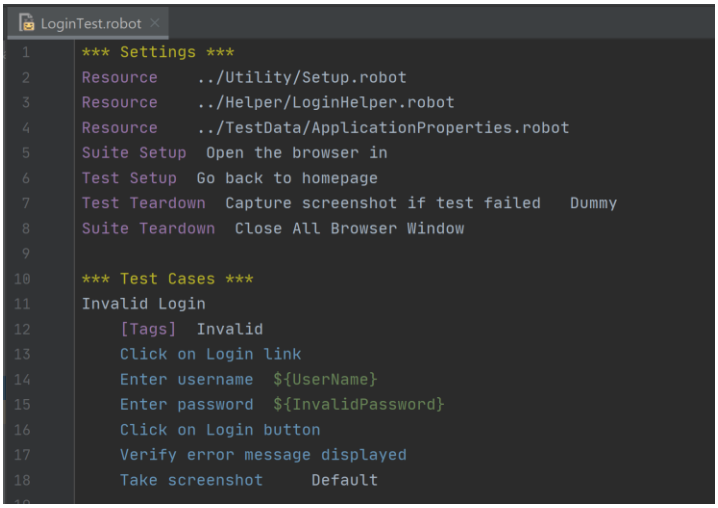
The name of the keyword to be executed as a setup or a teardown can be a variable. This facilitates having different setups or teardowns in different environments by giving the keyword name as a variable from the command line.

So what we will do we will move Open the browser in from Test Setup to Suite Setup which we would me we will be opening it once , now question here is we have two cases for each of them we open browser navigate to url perform test steps and clos the browser , in order to re-architect the flow what

we will do is in Test Setup we will navigate back to home page this way before next test starts we will be on home page so rest of test steps can be executed.

Thus in Test Teardown we can add call to function Capture Screenshot which we implemented in Setup.robot . Finally in Suite Teardown we can close the browser.

This is how it will look in Settings section LoginTest.robot

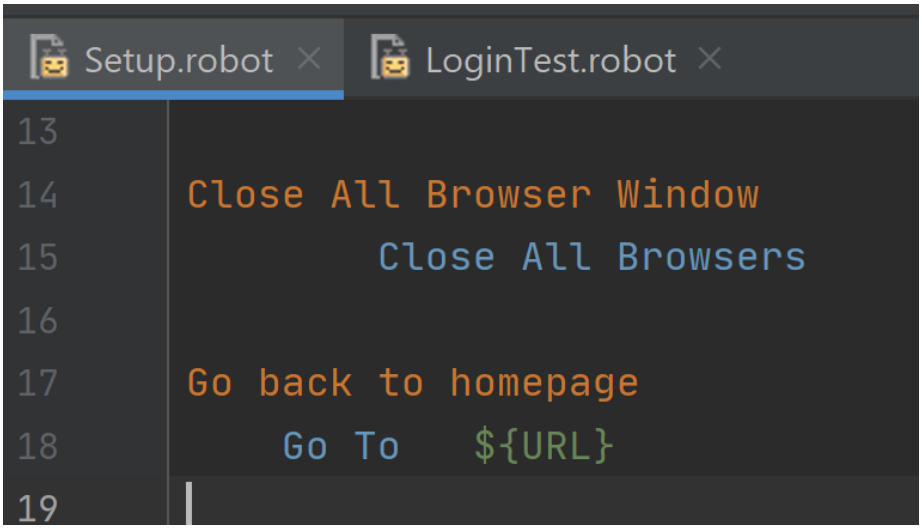


Code: LoginTest.robot

```
*** Settings ***
Resource    ../Utility/Setup.robot
Resource    ../Helper/LoginHelper.robot
Resource    ../TestData/ApplicationProperties.robot
Suite Setup  Open the browser in
Test Setup  Go back to homepage
Test Teardown  Capture screenshot if test failed   Dummy
Suite Teardown  Close All Browser Window

*** Test Cases ***
Invalid Login
   [Tags]  Invalid
   Click on Login link
   Enter username  ${UserName}
   Enter password  ${InvalidPassword}
   Click on Login button
   Verify error message displayed
   Take screenshot     Default

Valid Login Test
   [Tags]  Valid
   Click on Login link
   Enter username  ${UserName}
   Enter password  ${Password}
   Click on Login button
   Verify User is at Home page
```

We will have implement Go back to homepage function we are calling in Test Setup in Setup.robot .

As you can see below I added the function , here its calling Go To SeleniumLibrary keyword which helps navigating to URL so we are providing by ${URL} (from ApplicationProperties.robot)



Go To: http://robotframework.org/SeleniumLibrary/SeleniumLibrary.html#Go%20To

Go To	*url*	Navigates the current browser window to the provided `url`.

Code: Setup.robot

```
*** Settings ***
Library    Selenium2Library
Resource   ../TestData/ApplicationProperties.robot
Library  OperatingSystem
Library  BuiltIn
Library  String

*** Keywords ***
Open the browser in
   Open Browser   ${URL}  Chrome
   Maximize Browser Window
   Set Selenium Timeout   10s

Close All Browser Window
      Close All Browsers

Go back to homepage
   Go To  ${URL}

Capture screenshot if test failed
   [Arguments]  ${FileName}
   ${now}    Evaluate
'{dt.second}{dt.minute}{dt.hour}{dt.day}{dt.month}{dt.year}'.format(dt=datetime.datetime.now())
modules=datetime
   Run Keyword If Test Failed     Capture Page Screenshot
${OUTPUTDIR}/ScreenShot/${FileName}_${now}.png
```

Now save all files and run the test without using include or exclude , I want you to notice after first is executed it will navigate back to homepage (it wont open the new browser).

Run the command: robot -d "Reports" TestCases/*.robot

```
Terminal:  Local  Local (2)  +

(venv) C:\Users\user\PycharmProjects\robotframeworkautomation>robot -d "Reports" TestCases/*.robot
==============================================================================
LoginTest
==============================================================================
```

Now lets see whether it captures screenshot if test fails, in order to simulate this we can slightly change locator from ${InvalidPassword} to ${InvalidPasswor} (just removed the 'd') as you can see below. This will fail the test and capture the screenshot not only that it will embed in report as well.

```
*** Test Cases ***
Invalid Login
    [Tags]  Invalid
    Click on Login link
    Enter username  ${UserName}
    Enter password  ${InvalidPasswor}
    Click on Login button
    Verify error message displayed
```

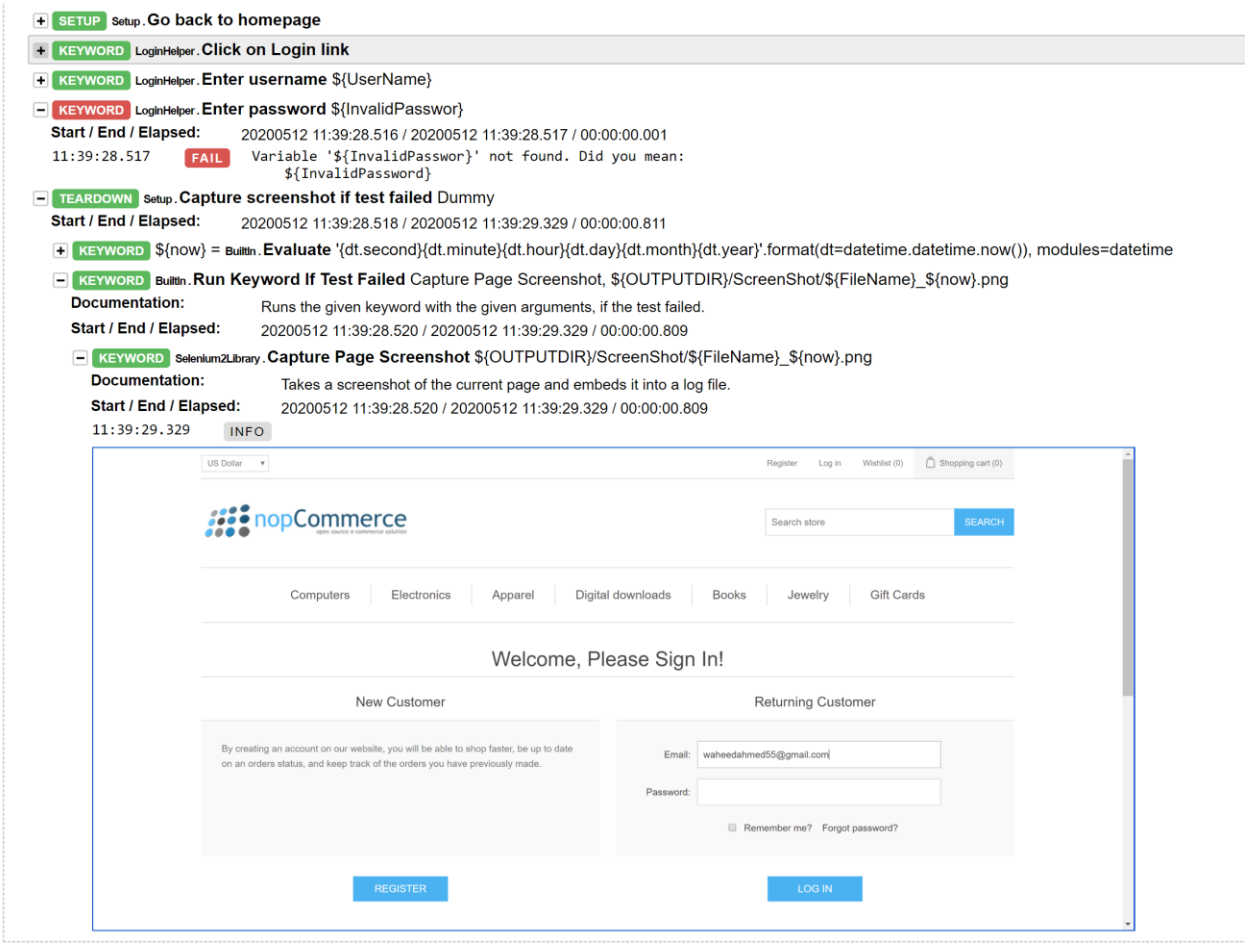
Save all files, and lets run the test

```
(venv) C:\Users\user\PycharmProjects\robotframeworkautomation>robot -d "Reports" TestCases/*.robot
```

You will see from output it failed at that step Enter Password of Invalid Login and didn't perform next steps and proceeded to next test and executed that. But before proceeding it took screenshot.

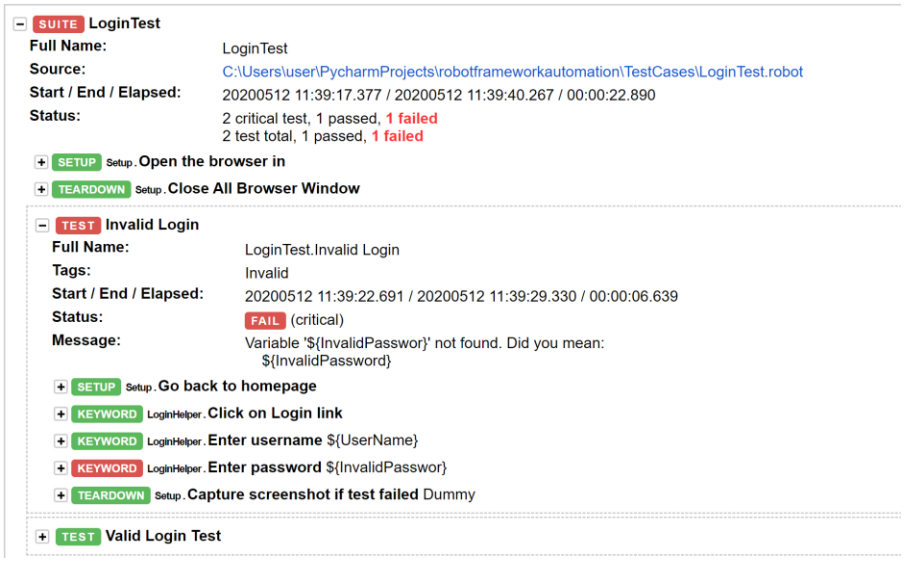
```
Terminal:  Local  Local (2)  +

DevTools listening on ws://127.0.0.1:55961/devtools/browser/70482261-b49e-4941-837c-b1d2b8d58248
[592:13524:0512/113918.648:ERROR:browser_switcher_service.cc(238)] XXX Init()
Invalid Login                                                         | FAIL |
Variable '${InvalidPasswor}' not found. Did you mean:
    ${InvalidPassword}
------------------------------------------------------------------------------
Valid Login Test                                                      | PASS |
------------------------------------------------------------------------------
LoginTest                                                             | FAIL |
2 critical tests, 1 passed, 1 failed
2 tests total, 1 passed, 1 failed
==============================================================================
Output:  C:\Users\user\PycharmProjects\robotframeworkautomation\Reports\output.xml
Log:     C:\Users\user\PycharmProjects\robotframeworkautomation\Reports\log.html
Report:  C:\Users\user\PycharmProjects\robotframeworkautomation\Reports\report.html

(venv) C:\Users\user\PycharmProjects\robotframeworkautomation>
```

Now lets examine the test report, as you can see below it has captured the screenshot and didn't execute the next step and proceeded to next test case which is Valid Login Test



Valid Login Test was executed and passed

Test Execution Log



Now lets fix the locator back to ${InvalidPassword} . Save all files.

Now lets say we want to take a screenshot after specific to test step , in that case what we will do?

In order to capture screenshot after at any given step we will implement same method and we will change it slightly. We will create new function Take Screenshot in Setup.robot and will remove Builtin keyword Run Keyword If Test Failed because now we don't want to have that as condition. As you can see below

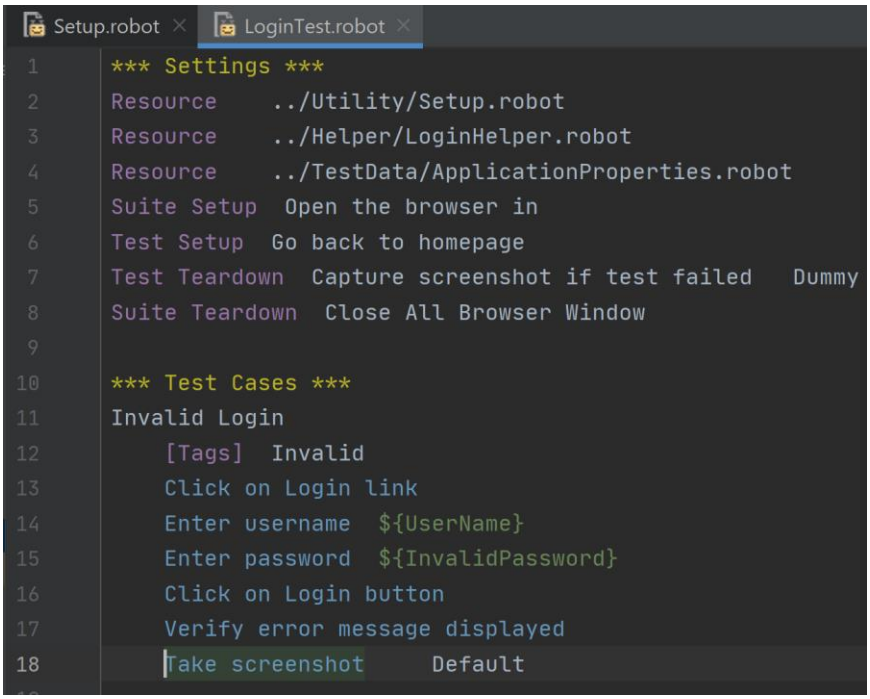
```
Take screenshot
    [Arguments]  ${FileName}
    ${now}    Evaluate  '{dt.second}{dt.minute}{dt.hour}{dt.day}{dt.month}{dt.year}'.format(dt=datetime.datetime.now())    modules=datetime
    Capture Page Screenshot    ${OUTPUTDIR}/ScreenShot/${FileName}_${now}.png
```

Here onething I forgot to explain earlier was it takes ${FileName} as arugment which we will pass from test step , so what it does it takes that and append it to ${now} which on second line gets complete date-timestamp.

Code : Setup.robot

```
*** Settings ***
Library    Selenium2Library
Resource   ../TestData/ApplicationProperties.robot
Library  OperatingSystem
Library  BuiltIn
Library  String

*** Keywords ***
Open the browser in
   Open Browser   ${URL}  Chrome
   Maximize Browser Window
   Set Selenium Timeout   10s

Close All Browser Window
      Close All Browsers

Go back to homepage
   Go To   ${URL}

Capture screenshot if test failed
   [Arguments]  ${FileName}
   ${now}    Evaluate
'{dt.second}{dt.minute}{dt.hour}{dt.day}{dt.month}{dt.year}'.format(dt=datetime.datetime.now())
modules=datetime
   Run Keyword If Test Failed     Capture Page Screenshot
${OUTPUTDIR}/ScreenShot/${FileName}_${now}.png

Take screenshot
   [Arguments]  ${FileName}
   ${now}    Evaluate
'{dt.second}{dt.minute}{dt.hour}{dt.day}{dt.month}{dt.year}'.format(dt=datetime.datetime.now())
modules=datetime
   Capture Page Screenshot    ${OUTPUTDIR}/ScreenShot/${FileName}_${now}.png
```

Now lets call Take screenshot in our LoginTest.robot . I would like to take screenshot in Invalid Login after error message being displayed .



So you can see Default is parameter I am passing to the function so when it creates screenshot file it will have file name Default_completedate.png

```
*** Settings ***
Resource    ../Utility/Setup.robot
Resource    ../Helper/LoginHelper.robot
Resource    ../TestData/ApplicationProperties.robot
Suite Setup  Open the browser in
Test Setup  Go back to homepage
Test Teardown  Capture screenshot if test failed   Dummy
Suite Teardown  Close All Browser Window

*** Test Cases ***
Invalid Login
   [Tags]  Invalid
   Click on Login link
   Enter username  ${UserName}
   Enter password  ${InvalidPassword}
   Click on Login button
   Verify error message displayed
   Take screenshot     Default

Valid Login Test
   [Tags]  Valid
   Click on Login link
   Enter username  ${UserName}
   Enter password  ${Password}
   Click on Login button
   Verify User is at Home page
```

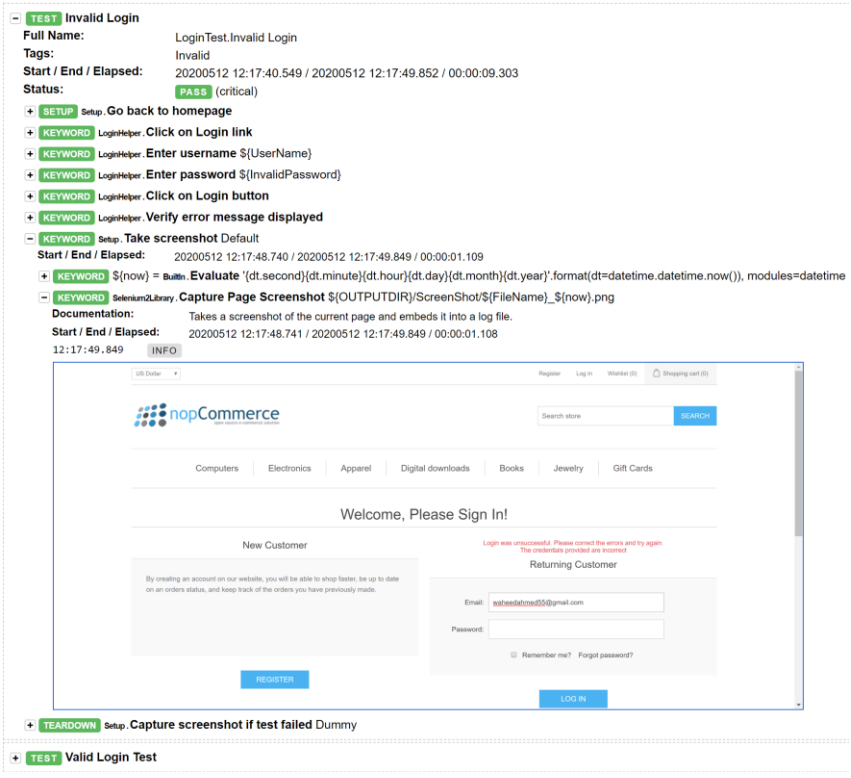
Save all files. Lets run the code

```
(venv) C:\Users\user\PycharmProjects\robotframeworkautomation>robot -d "Reports" TestCases/*.robot
```

You will see terminal output both test passed

```
Terminal:  Local ×  Local (2) ×  +
==============================================================================
LoginTest
==============================================================================

DevTools listening on ws://127.0.0.1:57977/devtools/browser/a90d4a75-3a12-4d23-ac74-389835bda901
[5756:8048:0512/121736.363:ERROR:browser_switcher_service.cc(238)] XXX Init()
Invalid Login                                                         | PASS |
------------------------------------------------------------------------------
Valid Login Test                                                      | PASS |
------------------------------------------------------------------------------
LoginTest                                                             | PASS |
2 critical tests, 2 passed, 0 failed
2 tests total, 2 passed, 0 failed
==============================================================================
Output:  C:\Users\user\PycharmProjects\robotframeworkautomation\Reports\output.xml
Log:     C:\Users\user\PycharmProjects\robotframeworkautomation\Reports\log.html
Report:  C:\Users\user\PycharmProjects\robotframeworkautomation\Reports\report.html
```

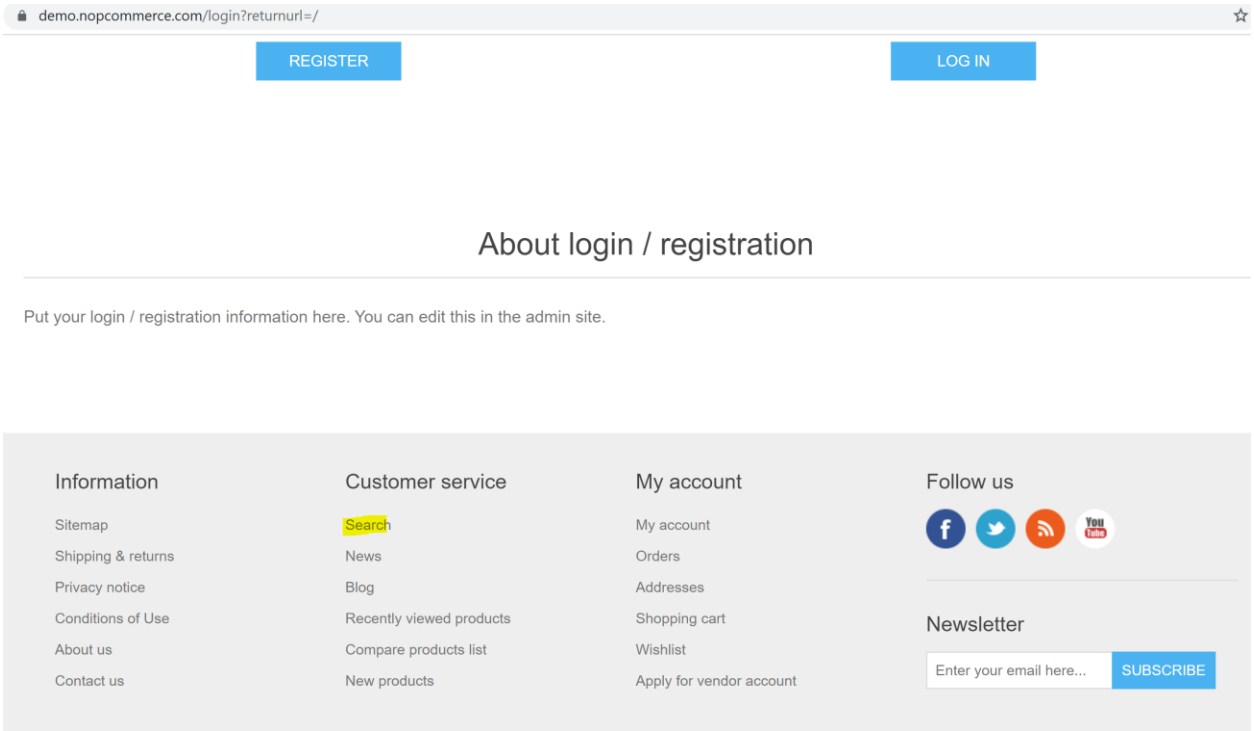
Now let see whether it took screenshot and embedded into report. As you can see it did took screenshot and embedded in the report.



Lets Recape we covered how to write simple tests by implementing helper functions which in turn calls SeleniumLibrary Keywords, we looked into how to locate objects (locators) , we looked into various SeleniumLibrary Kewords , also covered how to implement SeleniumLibrary keyword and use in our test cases, Simple Test Setup and Teardown and moved to Suite Setup and Teardown and looked into Error/failure handling if tests fails capture log/screenshot and how to take screen shot, learnt how to control the flow/execution of tests using [Tags], along with properly structured the framework using POM, Helper methods , and created ApplicationProperties.robot to manage test data. Most of the best practices and framework setup is complete here, in next sections I will show some more SeleniumLibrary keywords and how to use them.

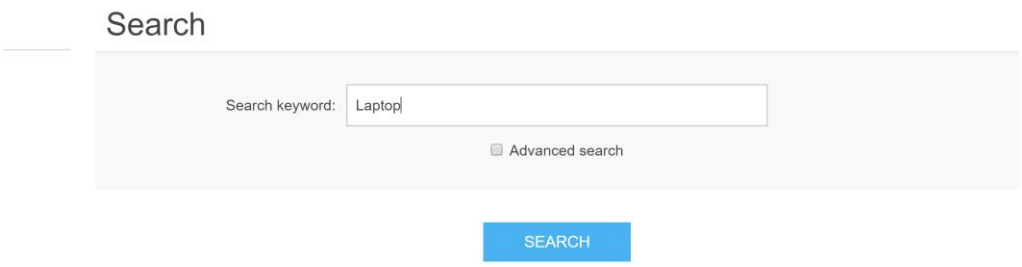
How to do Scroll on pages top to bottom or vice versa, drop-down lists and Mouse Over actions.

Lets create new Test file in which we cover Search item and add to cart functionality. As soon the we navigate to website scroll to Search at the bottom of page in footer

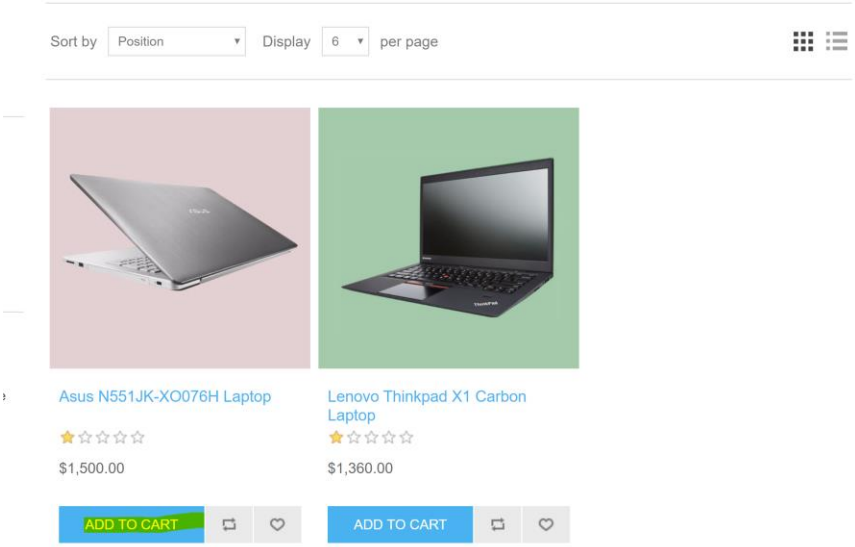


Click on it

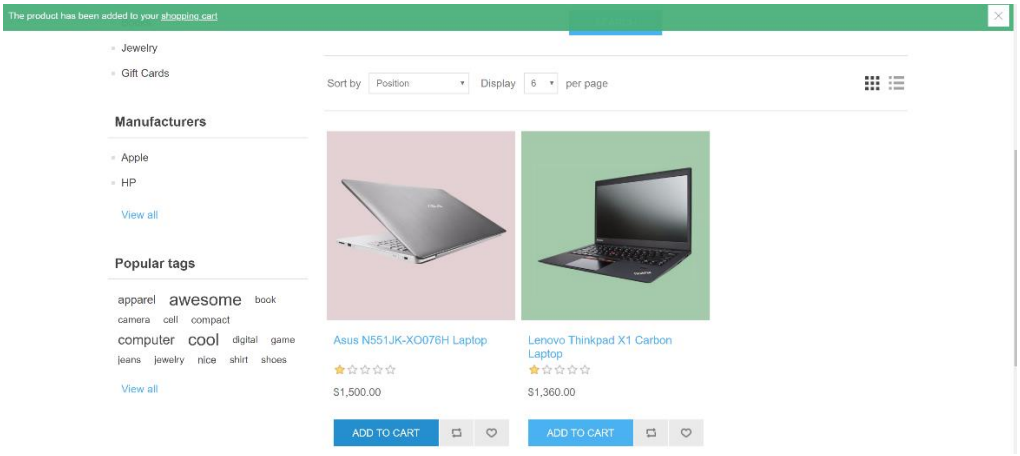
Then enter item to be searched and click on Search button



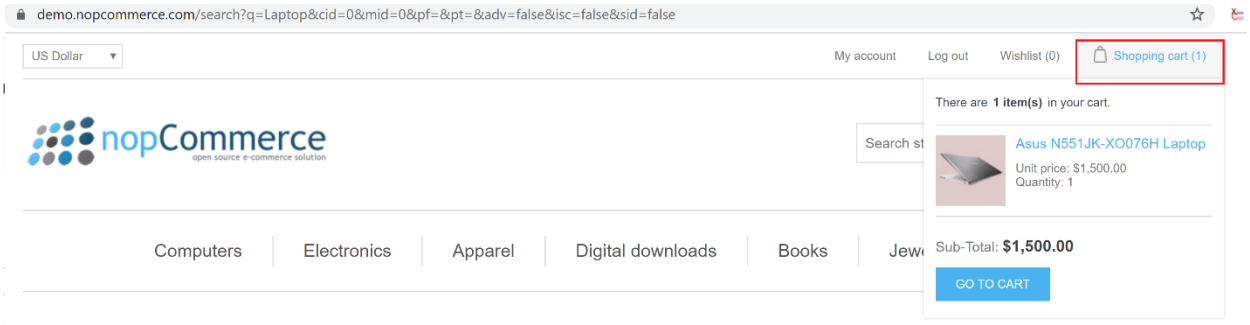
The sort item by High to Low from drop-down lists . Than click on Add to cart button of first item



Then we will scroll to top to take screenshot of successfully added to cart message.



Finally we will perform mouse over Shopping cart



Switch from Selenium2Library to SeleniumLibrary

Since the Library name has changed from Selenium2Library to SeleniumLibrary we will change it accordingly.

First lets uninstall it

Run the command : pip uninstall robotframework-selenium2library

And then install SeleniumLibrary using: pip install robotframework-seleniumlibrary

Go to SeleniumKeywords.robot and change Library to SeleniumLibrary

```
*** Settings ***
Library       SeleniumLibrary
```

Go to Setup.robot and make same change

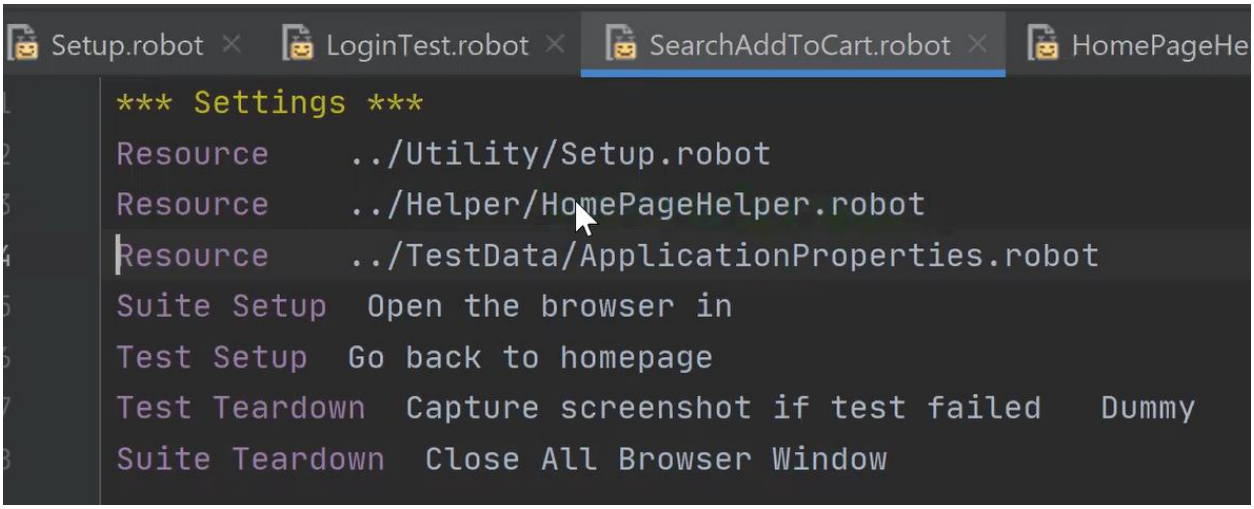
Setup.robot

```
1  *** Settings ***
2  Library     SeleniumLibrary
3  Resource    ../TestData/ApplicationProperties.robot
4  Library  OperatingSystem
5  Library  BuiltIn
6  Library  String
```

Lets get started I will skip some stuff/steps as they have been covered previously.

Create Test file SearchAddToCart.robot under TestCase directory .

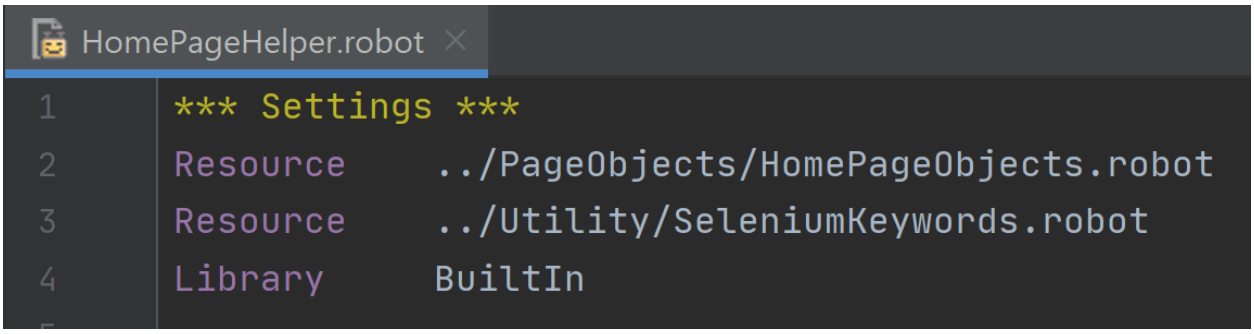
Lets write code in SearchAddToCart.robot . First section which is Settings will be same of LoginTest.robot



Code: SearchAddToCart.robot

```
*** Settings ***
Resource    ../Utility/Setup.robot
Resource    ../Helper/HomePageHelper.robot
Resource    ../TestData/ApplicationProperties.robot
Suite Setup  Open the browser in
Test Setup  Go back to homepage
Test Teardown  Capture screenshot if test failed   Dummy
Suite Teardown  Close All Browser Window
```

Similarly, create another file HomePageHelper.robot under Helper directory. First section in this file with be same as of LoginHelper.robot



Code: HomePagerHelper.robot

```
*** Settings ***
Resource    ../PageObjects/HomePageObjects.robot
Resource    ../Utility/SeleniumKeywords.robot
Library    BuiltIn
```

Next lets implement the scroll SeleniumLibrary keyword in out SeleniumKeyword.robot.

I added function Scroll for element which is calling Scroll Element into View SeleniumLibrary , what it does is it keeps scrolling until finds the locator of the object .

```
Scroll for element
    [Arguments]     ${locator}
    Scroll Element Into View    ${locator}
```

Scroll Element into View:
http://robotframework.org/SeleniumLibrary/SeleniumLibrary.html#Scroll%20Element%20Into%20View

Scroll Element Into View	*locator*	Scrolls the element identified by `locator` into view. See the *Locating elements* section for details about the locator syntax. New in SeleniumLibrary 3.2.0

Similarly added scroll to bottom and scroll to top using Execute Javascript Selenium Library keyword which allows to execute javascript code

```
Scroll at the bottom of the page
    Execute Javascript      window.scrollTo(0,document.body.scrollHeight)

Scroll at the top of the page
    Execute Javascript      window.scrollTo(0,0)
```

Here window.scrollTo(0, document.body.scrollHieght) is javascript function which helps to scroll , in our case we are scrolling here to bottom so we provided 0 based on scrollHeight of page.

Similarly if you can to move to the top of the page we use 0 , 0 .

Execute Javascript:
http://robotframework.org/SeleniumLibrary/SeleniumLibrary.html#Execute%20Javascript

Execute Javascript	**code*	Executes the given JavaScript code with possible arguments. `code` may be divided into multiple cells in the test data and `code` may contain multiple lines of code and arguments. In that case, the JavaScript code parts are concatenated together without adding spaces and optional arguments are separated from `code`. If `code` is a path to an existing file, the JavaScript to execute will be read from that file. Forward slashes work as a path separator on all operating systems. The JavaScript executes in the context of the currently selected frame or window as the body of an anonymous function. Use `window` to refer to the window of your application and `document` to refer to the document object of the current frame or window, e.g. `document.getElementById('example')`. This keyword returns whatever the executed JavaScript code returns. Return values are converted to the appropriate Python types. Starting from SeleniumLibrary 3.2 it is possible to provide JavaScript arguments as part of `code` argument. The JavaScript code and arguments must be separated with *JAVASCRIPT* and *ARGUMENTS* markers and must be used exactly with this format. If the Javascript code is first, then the *JAVASCRIPT* marker is optional. The order of *JAVASCRIPT* and *ARGUMENTS* markers can be swapped, but if *ARGUMENTS* is the first marker, then *JAVASCRIPT* marker is mandatory. It is only allowed to use *JAVASCRIPT* and *ARGUMENTS* markers only one time in the `code` argument. Examples:

Execute JavaScript	window.myFunc('arg1', 'arg2')			
Execute JavaScript	${CURDIR}/js_to_execute.js			
Execute JavaScript	alert(arguments[0]);	ARGUMENTS	123	
Execute JavaScript	ARGUMENTS	123	JAVASCRIPT	alert(arguments[0]);

Code: SeleniumKeywords.robot

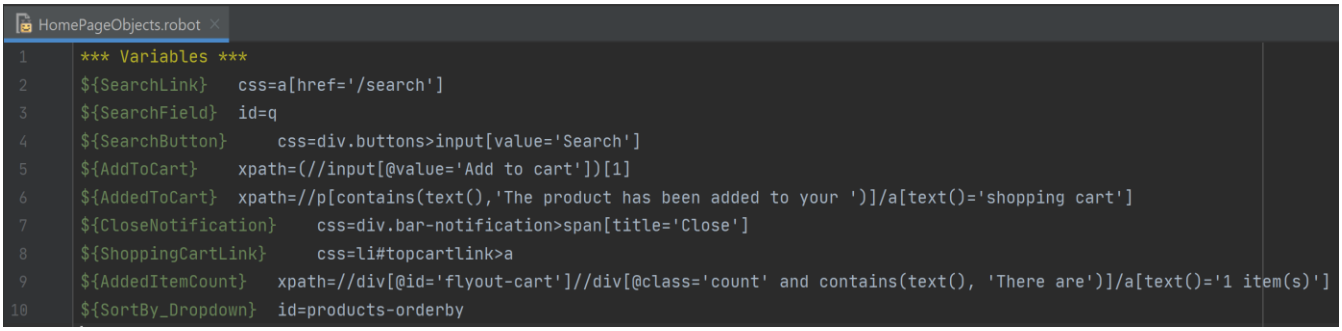
```
*** Settings ***
Library    SeleniumLibrary

*** Keywords ***
Clear Input Field
   [Arguments]   ${locator}
   Wait For Element Present   ${locator}  5   Element not found in 5 seconds
   Clear Element Text   ${locator}
   ${value}=    Get Element Attribute   ${locator}    value
   ${backspaces count}=   Get Length    ${value}
   Run Keyword If   """${value}""" != ''
   ...    Repeat Keyword  ${backspaces count +1}  Press Key  ${locator}   \\08

Set Value For Input Field
   [Arguments]   ${locator}   ${inputval}
   sleep   2s
   Clear Input Field   ${locator}
   Input Text   ${locator}   ${inputval}

Click On Page Element
   [Arguments]   ${locator}
   Wait For Element Present   ${locator}  5   Element not found in 5 seconds
   Click Element   ${locator}

Wait For Page Load
   [Arguments]   ${pgloadlocator}
   Wait Until Page Contains Element   ${pgloadlocator}
   Sleep   5s

Wait For Element Present
   [Arguments]   ${locator}   ${timeout}   ${err_msg}
   #Wait For Element Present   ${locator}  5   Element not found in 5 seconds
   Wait Until Page Contains Element   ${locator}   ${timeout}   ${err_msg}

Get Attribute Value
   [Arguments]   ${locator}   ${attribute}
   #Wait For Element Present   ${locator}  5   Element not found in 5 seconds
   ${string}=   Get Element Attribute   ${locator}   ${attribute}
   log to console   \n get attribute ${string}.\n
   [Return]   ${string}

Back On Screen
   Go Back

Wait For Text Present
   [Arguments]   ${text}   ${seconds}   ${message}
   Wait Until Page Contains   ${text}   ${seconds}   ${message}

Verify Text Present
   [Arguments]   ${text}
   Page Should Contain   ${text}

Verify Element Present
```

```
    [Arguments]    ${locator}
    #Wait For Element Present    ${locator}  5    Element not found in 5 seconds
    Page Should Contain Element    ${locator}

Generate Random String With Defined Size
    [Arguments]    ${size}    ${type}
    ${str}=    Generate Random String    ${size}    ${type}
    [Return]    ${str}

Generate Random Number With Defined Size
    [Arguments]    ${size}    ${type}
    ${num}=    Generate Random String    ${size}    ${type}
    [Return]    ${num}

Match Value
    [Arguments]    ${locator}    ${attr_name}    ${match_pattern}
    #Wait For Element Present    ${locator}  5    Element not found in 5 seconds
    Element Attribute Should Match    ${locator}    ${attr_name}    ${match_pattern}

Verify Text Not Present
    [Arguments]    ${text}
    Page Should Not Contain    ${text}

Select List Option By visible Text
    [Arguments]    ${locator}    ${value}
    Wait For Element Present    ${locator}  5    Element not found in 5 seconds
    Select From List By Label    ${locator}    ${value}

Verify Page title
    [Arguments]    ${title}
    Title Should Be    ${title}

Navigate to the page
    [Arguments]    ${URL}
    Go To   ${URL}

Verify Element Displayed
    [Arguments]    ${locator}
    Wait For Element Present    ${locator}  5    Element not found in 5 seconds
    Element Should Be Visible    ${locator}

Verify Element not displayed
    [Arguments]    ${locator}
    Element Should Not Be Visible    ${locator}

Verify Current URL should contains
    [Arguments]    ${URL}
    Location Should Contain    ${URL}

Static wait for
    [Arguments]    ${Second}
    Sleep   ${Second}s

Get all element count
    [Arguments]    ${locator}
```

```
    ${num}=    Get Element Count    ${locator}
    [Return]    ${num}

Verify both values are equal
    [Arguments]    ${val1}  ${val2}
    Should Be Equal    ${val1}    ${val2}

Switch to window
    [Arguments]    ${type}
    Select Window  ${type}

Close current window
    Close Window

Get Current URL
    return from keyword    Get Location

Press keyboard key
    [Arguments]    ${locator}   ${Key}
    Wait For Element Present    ${locator}  5    Element not found in 5 seconds
    Press Key  ${locator}   \\08

Mouse Over on the element
    [Arguments]    ${locator}
    Wait For Element Present    ${locator}  5    Element not found in 5 seconds
    Mouse Over  ${locator}

URL Should Contain
    [Arguments]    ${expected}
    Location Should Contain    ${expected}

Scroll for element
    [Arguments]    ${locator}
    Scroll Element Into View    ${locator}

Scroll at the bottom of the page
    Execute Javascript    window.scrollTo(0,document.body.scrollHeight)

Scroll at the top of the page
    Execute Javascript    window.scrollTo(0,0)
```

Lets create HomePageObjects.robot under PageObjects directory.

Lets add locators of all element we will be interacting like search link at the bottom of the page , search field, search button, add to cart button, added to cart, shop cart link etc.



Code: HomePageObjects.robot

```
*** Variables ***
${SearchLink}  css=a[href='/search']
${SearchField}  id=q
${SearchButton}  css=div.buttons>input[value='Search']
${AddToCart}  xpath=(//input[@value='Add to cart'])[1]
${AddedToCart}  xpath=//p[contains(text(),'The product has been added to your ')]/a[text()='shopping cart']
${CloseNotification}  css=div.bar-notification>span[title='Close']
${ShoppingCartLink}  css=li#topcartlink>a
${AddedItemCount}  xpath=//div[@id='flyout-cart']//div[@class='count' and contains(text(), 'There are')]/a[text()='1 item(s)']
${SortBy_Dropdown}  id=products-orderby
```

Now lets go to HomePagerHelper.robot and function which will help in clicking the search link

First we will scroll to that element which is search link and click on it

```
*** Settings ***
Resource  ../PageObjects/HomePageObjects.robot
Resource  ../Utility/SeleniumKeywords.robot
Library  BuiltIn

*** Keywords ***
Click on search link
  Scroll for element  ${SearchLink}
  Click On Page Element  ${SearchLink}
```

As you can see above I wrote function in HomePageHelper.robot Click on search link in which we are calling Scroll for element with paratmeter which is locator of search link and Click On Page Element with arugment ${SearchLink} which is locator of search link from SeleniumKeywords.robot.

Now lets go to SearchAddToCart.robot and step in our test case. I have create test case with name Serch and Add to cart and added first by calling helper function Click on search link implemented in HomePageHelper.robot

SearchAddToCart.robot

```
*** Settings ***
Resource    ../Utility/Setup.robot
Resource    ../Helper/HomePageHelper.robot
Resource    ../TestData/ApplicationProperties.robot
Suite Setup  Open the browser in
Test Setup  Go back to homepage
Test Teardown  Capture screenshot if test failed   Dummy
Suite Teardown  Close All Browser Window

*** Test Cases ***
Search and Add to cart
    Click on search link
```

Save all files and run.

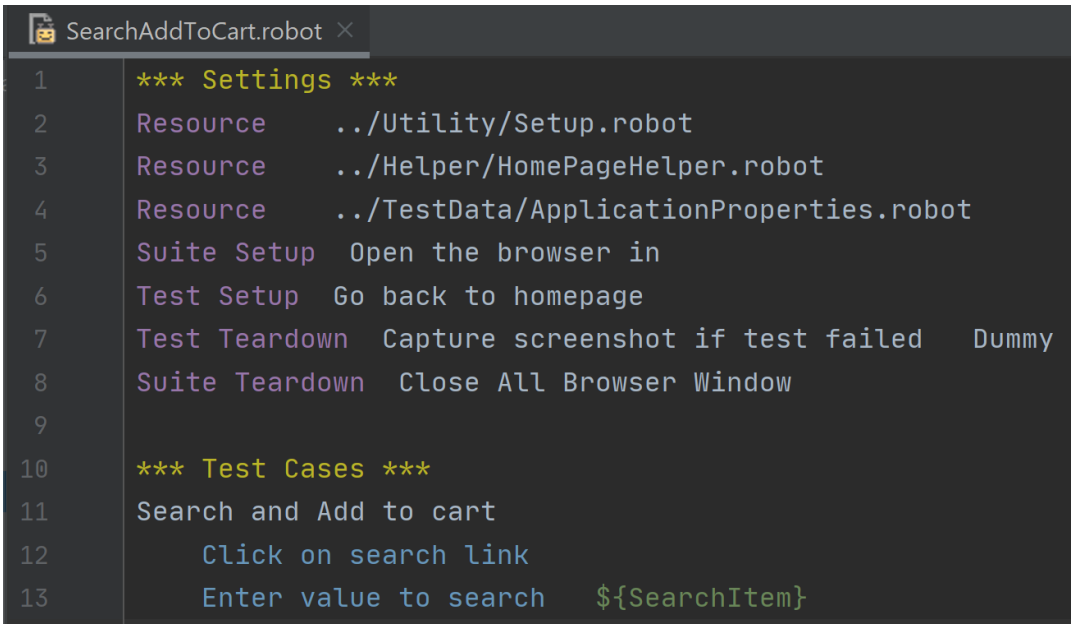
robot -d "Reports" TestCases/SearchAddToCart.robot

```
Terminal:  Local  Local (2)  +
Microsoft Windows [Version 10.0.17763.1098]
(c) 2018 Microsoft Corporation. All rights reserved.

(venv) C:\Users\user\PycharmProjects\robotframeworkautomation>robot -d "Reports" TestCases/SearchAddToCart.robot
```

You will see in terminal output it Passed

```
DevTools listening on ws://127.0.0.1:53901/devtools/browser/844f695c-1544-4eb6-839f-662f4f9a4e33
[1336:15296:0509/152338.706:ERROR:browser_switcher_service.cc(238)] XXX Init()
Search and Add to cart                                                PASS |
------------------------------------------------------------------------------
SearchAddToCart                                                     | PASS |
1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
```

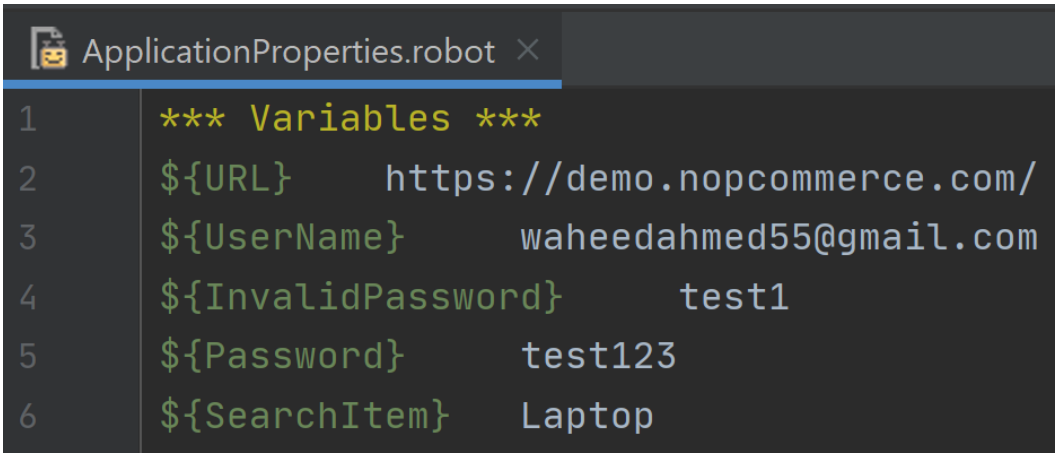
Now lets write step to enter text in search field Laptop as you can see below I added Enter value to search Laptop , here Laptop is passed as parameter



Code:SearchAddToCart.robot

```
*** Settings ***
Resource   ../Utility/Setup.robot
Resource   ../Helper/HomePageHelper.robot
Resource   ../TestData/ApplicationProperties.robot
Suite Setup  Open the browser in
Test Setup  Go back to homepage
Test Teardown  Capture screenshot if test failed   Dummy
Suite Teardown  Close All Browser Window

*** Test Cases ***
Search and Add to cart
   Click on search link
   Enter value to search   ${SearchItem}
```

${SearchItem} I have defined in ApplicationProperties.robot



Code: ApplicationProperties.robot

```
*** Variables ***
${URL}    https://demo.nopcommerce.com/
${UserName}    waheedahmed55@gmail.com
${InvalidPassword}    test1
${Password}    test123
${SearchItem}    Laptop
```

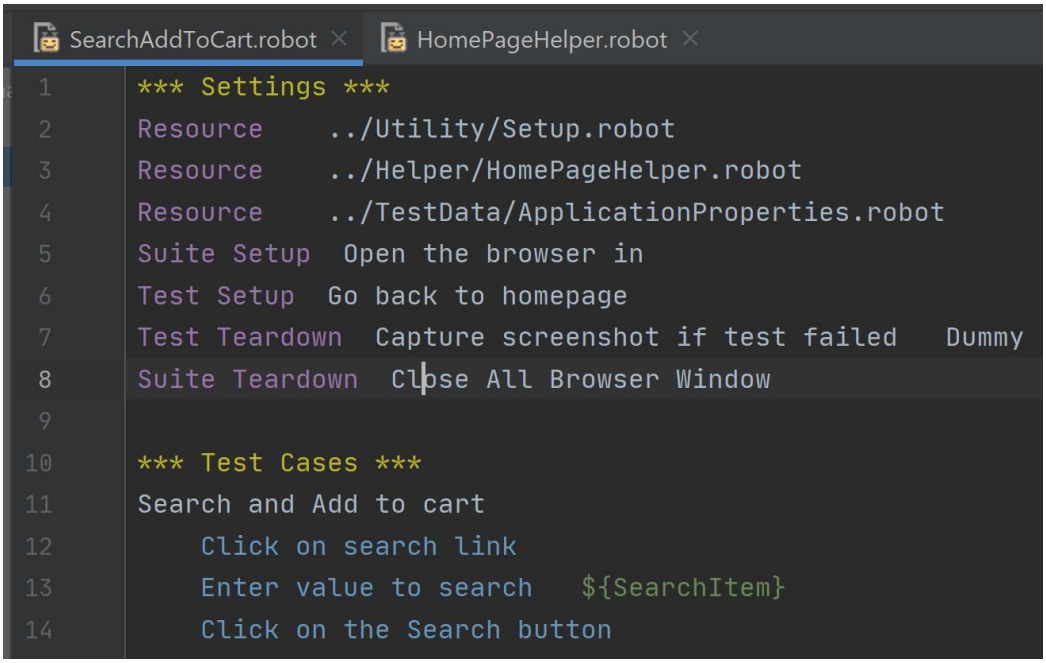
Now lets implement it in HomePageHelper.robot

Code : HomePageHelper.robot

```
*** Settings ***
Resource    ../PageObjects/HomePageObjects.robot
Resource    ../Utility/SeleniumKeywords.robot
Library    BuiltIn

*** Keywords ***
Click on search link
    Scroll for element  ${SearchLink}
    Click On Page Element    ${SearchLink}

Enter value to search
    [Arguments]    ${inputText}
    Set Value For Input Field    ${SearchField}  ${inputText}
```

Here we are calling Set Value For Input Field function from SeleniumKeyword.robot which in turn calls another function Clear Input Field which is implemented just above it which calls Clear Element Text SeleniumLibrary which clears the text fron input field .Set Value For Input Field function also call Input Text which is Selenium Library to set text.

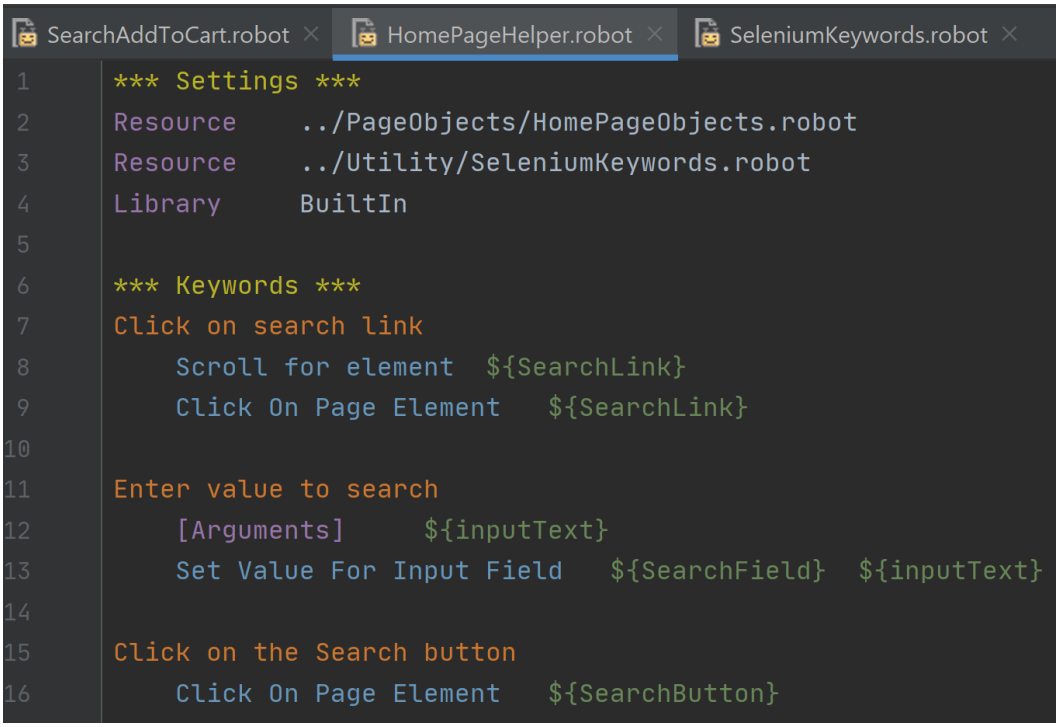
After entering Laptop in search field we will click on Search button . Lets add step in Test case



Code: SearchAddToCart.robot

```
*** Settings ***
Resource    ../Utility/Setup.robot
Resource    ../Helper/HomePageHelper.robot
Resource    ../TestData/ApplicationProperties.robot
Suite Setup  Open the browser in
Test Setup  Go back to homepage
Test Teardown  Capture screenshot if test failed   Dummy
Suite Teardown  Close All Browser Window

*** Test Cases ***
Search and Add to cart
   Click on search link
   Enter value to search   ${SearchItem}
   Click on the Search button
```

Now lets implement this in HomePageHelper.robot



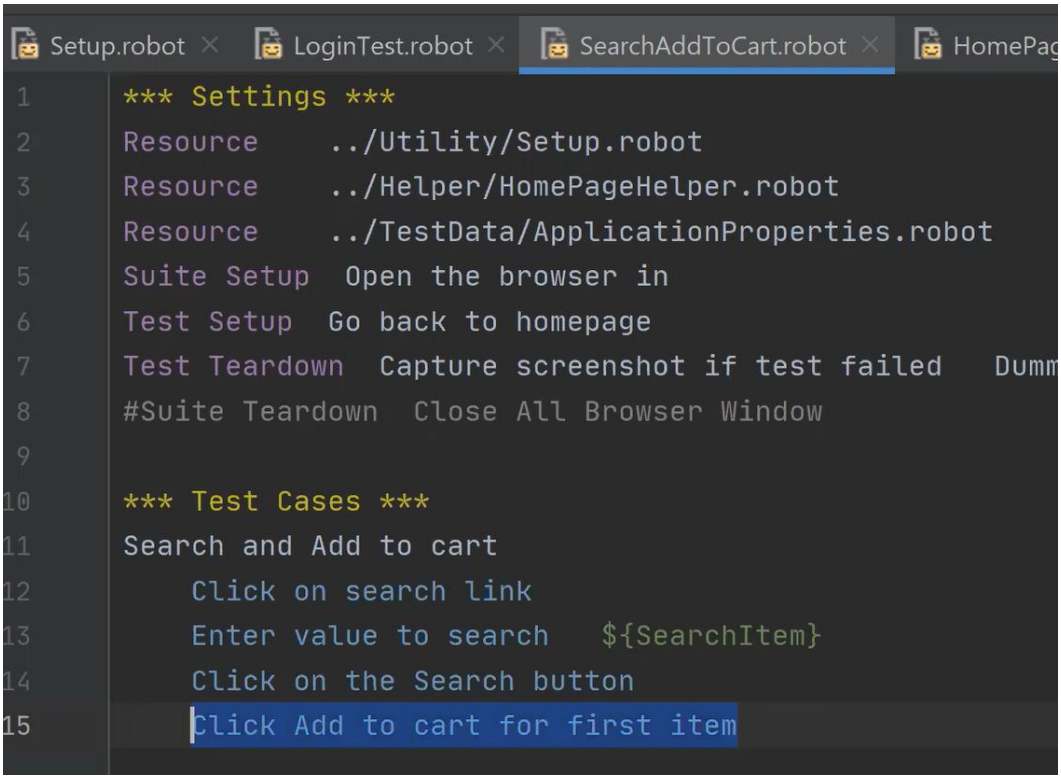
Code: HomePageHelper.robot

```
*** Settings ***
Resource   ../PageObjects/HomePageObjects.robot
Resource   ../Utility/SeleniumKeywords.robot
Library    BuiltIn

*** Keywords ***
Click on search link
   Scroll for element  ${SearchLink}
   Click On Page Element  ${SearchLink}

Enter value to search
   [Arguments]    ${inputText}
   Set Value For Input Field  ${SearchField}  ${inputText}

Click on the Search button
   Click On Page Element  ${SearchButton}
```

As you can see above I am calling function Click On Page Element and passing ${SearchButton} locator from SeleniumKeywords.robot. In that function its calling Click Element which is SeleniumLibrary keyword and clicks on element using locator passed.

Next we will on Add to cart button of first item . Lets add the step for it

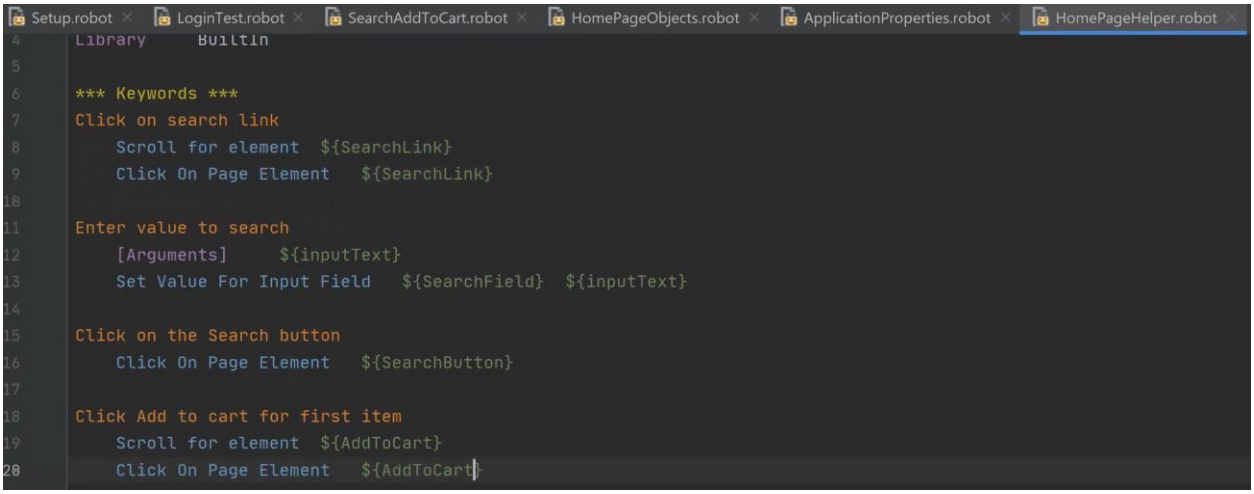
Lets go to SearchAddToCart.robot and you can see below written step Click Add to cart for first item



Code: SearchAddToCart.robot

```
*** Settings ***
Resource    ../Utility/Setup.robot
Resource    ../Helper/HomePageHelper.robot
Resource    ../TestData/ApplicationProperties.robot
Suite Setup  Open the browser in
Test Setup  Go back to homepage
Test Teardown  Capture screenshot if test failed   Dummy
Suite Teardown  Close All Browser Window

*** Test Cases ***
Search and Add to cart
   Click on search link
   Enter value to search   ${SearchItem}
   Click on the Search button
   Click Add to cart for first item
```

Now lets implement this in HomePageHelper.robot



Code: HomePageHelper.robot

```
Click Add to cart for first item
  Scroll for element  ${AddToCart}
  Click On Page Element  ${AddToCart}
```

This is calling same function as we called in Click on Search link just difference this for Add To Cart button as you can see we are passing ${AddToCart} button locator.

After we click on Add to cart button we see the success message on top of the page what we will do is we will verify if that message appears/displayed.

Lets add the step to SearchAddToCart.robot

```
*** Settings ***
Resource    ../Utility/Setup.robot
Resource    ../Helper/HomePageHelper.robot
Resource    ../TestData/ApplicationProperties.robot
Suite Setup  Open the browser in
Test Setup  Go back to homepage
Test Teardown  Capture screenshot if test failed   Dummy
#Suite Teardown  Close All Browser Window

*** Test Cases ***
Search and Add to cart
    Click on search link
    Enter value to search   ${SearchItem}
    Click on the Search button
    Click Add to cart for first item
    Item has added to cart sucessfully
```

Code: SearchAddToCart.robot

```
*** Settings ***
Resource    ../Utility/Setup.robot
Resource    ../Helper/HomePageHelper.robot
Resource    ../TestData/ApplicationProperties.robot
Suite Setup  Open the browser in
Test Setup  Go back to homepage
Test Teardown  Capture screenshot if test failed   Dummy
Suite Teardown  Close All Browser Window

*** Test Cases ***
Search and Add to cart
   Click on search link
   Enter value to search   ${SearchItem}
   Click on the Search button
   Click Add to cart for first item
   Item has added to cart sucessfully
```

Now lets implement this in HomePageHelper.robot

```
Item has added to cart sucessfully
    Wait For Element Present    ${AddedToCart}  10  Item not added to cart
    Click On Page Element   ${CloseNotification}
```

In above code you can see we are calling two functions Wait For Element Present and Click On Page Element from SeleniumKyewords.robot . In Wait For Element Present function we passing parameters locator of Added to cart success message which is ${AddedToCart}, 10 s wait , and error message Item not added to cart which is calling Wait Until Page Contains Element it takes locator of element and wait for 10 s if element isn't present within 10s it will display error message Item not added to cart which is SeleniumLibrary keyword and other function Click on Page Element we have discussed earlier, its clicking on success message close icon

Code: HomePageHelper.robot

```
Item has added to cart sucessfully
   Wait For Element Present   ${AddedToCart}  10  Item not added to cart
   Click On Page Element   ${CloseNotification}
```

Now lets add the step to mouse over the Shopping Cart to see if item added is visible/present.

Go to HomePageHelper.robot and write function to mouse over the Shopping Cart .

```
Verify Cart displayed item
    Mouse Over on the element    ${ShoppingCartLink}
    Verify Element Present  ${AddedItemCount}
```

Code: HomePageHelper.robot

```
Verify Cart displayed item
  Mouse Over on the element  ${ShoppingCartLink}
  Verify Element Present  ${AddedItemCount}
```

In above code you can see we wrote function Verify Cart displayed Item we are calling two functions Mouse Over on the element which in turn is calling function Wait For Element Present which is implemented just above it and it calls Wait Until Page Contains Element which is SeleniumLibrary keyword already explained above. Next we see in Mouse Over the element calls Mouse Over ${locator} which is SeleniumLibrary Keyword

```
Mouse Over on the element
    [Arguments]    ${locator}
    Wait For Element Present    ${locator}  5    Element not found in 5 seconds
    Mouse Over  ${locator}
```

Mouse Over: http://robotframework.org/SeleniumLibrary/SeleniumLibrary.html#Mouse%20Over

Mouse Over	*locator*	Simulates hovering the mouse over the element `locator`. See the *Locating elements* section for details about the locator syntax.

Next in Verify Cart displayed Item is call to function Verify Element Present from SeleniumKeywords.robot I have already discussed earlier , here mouse is over the Shopping Cart it will display item count it verifies its present/displayed.

Now lets add the step in SearchAddToCart.robot

Setup.robot | LoginTest.robot | SearchAddToCart.robot | HomePageO

```
*** Settings ***
Resource    ../Utility/Setup.robot
Resource    ../Helper/HomePageHelper.robot
Resource    ../TestData/ApplicationProperties.robot
Suite Setup  Open the browser in
Test Setup  Go back to homepage
Test Teardown  Capture screenshot if test failed   Dummy
#Suite Teardown  Close All Browser Window

*** Test Cases ***
Search and Add to cart
    Click on search link
    Enter value to search   ${SearchItem}
    Click on the Search button
    Click Add to cart for first item
    Item has added to cart sucessfully
    Verify Cart displayed item
```

Now what I want is to add step to scroll up and take screenshot of the item added to cart success message . So What I did is right after step Click Add to cart for first item I scroll up to page and take screenshot.

Code: SearchAddToCart.robot

```
*** Settings ***
Resource    ../Utility/Setup.robot
Resource    ../Helper/HomePageHelper.robot
Resource    ../TestData/ApplicationProperties.robot
Suite Setup  Open the browser in
Test Setup  Go back to homepage
Test Teardown  Capture screenshot if test failed   Dummy
Suite Teardown  Close All Browser Window

*** Test Cases ***
Search and Add to cart
   Click on search link
   Enter value to search   ${SearchItem}
   Click on the Search button
   Click Add to cart for first item
   Scroll the screen to top
   Take screenshot     AddedItem
   Item has added to cart sucessfully
   Verify Cart displayed item
```

Now lets implement step Scroll the screen to top in HomePageHelper.robot whereas Take screenshot AddedItem is already implemented in Setup.robot where AddedItem is name of file screenshot will be saved as.

Lets go to HomepageHelper.robot

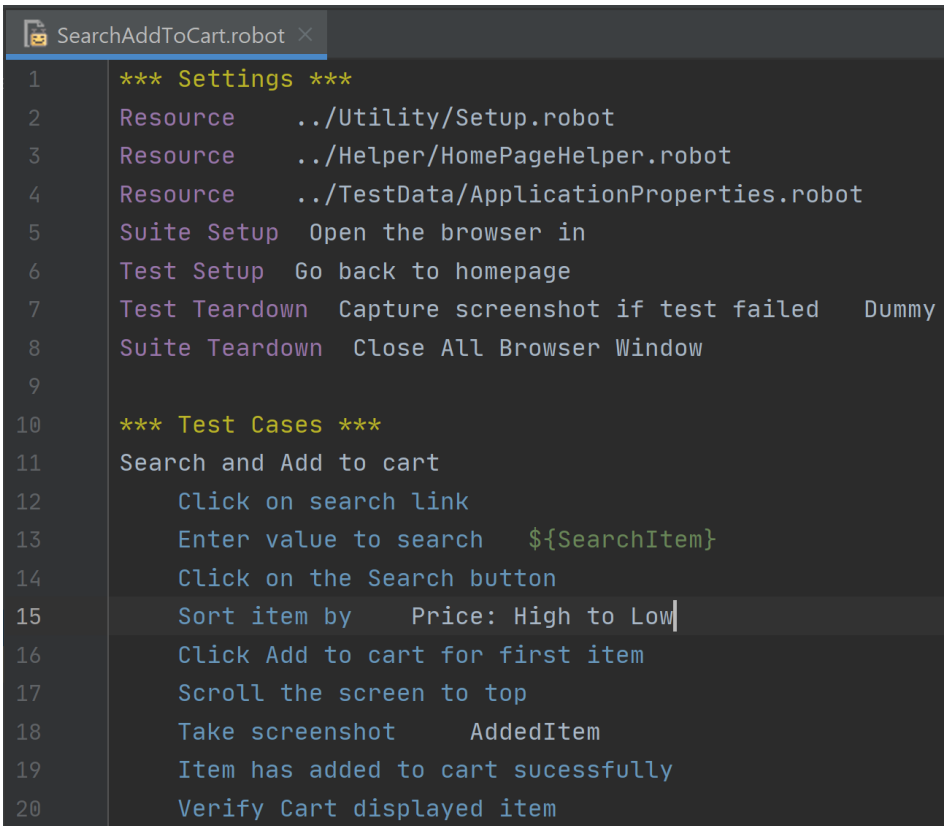
```
Scroll the screen to top
    Scroll at the top of the page
    sleep    2s
```

As you can see I am calling function Scroll at the top of the page from Seleniumeywords.robot which in turn is calling Execute Javascipty which SeleniumLibrary keyword which helps in executing javascript code and code we have is window.scrollTo(0,0) which means top of the page.

Code: HomePageHelper.robot

```
Scroll the screen to top
   Scroll at the top of the page
   sleep   2s
```

Let say clicking search button I want to sort result by Price : High to Low . Lets add step in SearchAddToCart.robot



As you can see above added the step Sort item by Price: High to Low where Price: High to Low is option from sort drop-down on website.

Code: SearchAddToCart.robot

```
*** Settings ***
Resource ../Utility/Setup.robot
Resource ../Helper/HomePageHelper.robot
Resource ../TestData/ApplicationProperties.robot
Suite Setup Open the browser in
Test Setup Go back to homepage
Test Teardown Capture screenshot if test failed Dummy
Suite Teardown Close All Browser Window

*** Test Cases ***
Search and Add to cart
  Click on search link
  Enter value to search ${SearchItem}
  Click on the Search button
  Sort item by Price: High to Low
  Click Add to cart for first item
  Scroll the screen to top
  Take screenshot AddedItem
  Item has added to cart sucessfully
  Verify Cart displayed item
```

Lets implement this in HomePageHelper.robot

```
Sort item by
    [Arguments]     ${visibleText}
    Select List Option By visible Text  ${SortBy_Dropdown}  ${visibleText}
```

Code: SearchAddToCart.robot

```
Sort item by
  [Arguments]    ${visibleText}
  Select List Option By visible Text  ${SortBy_Dropdown}  ${visibleText}
```

As you can see above we are calling function Select List Option By visible Text from SeleniumKeywords.robot and passing locator of drop-down ${SortBy_Dropdown} and text Price: High to Low which we are passing from test step as parameter in ${visibleText}

```
Select List Option By visible Text
    [Arguments]    ${locator}    ${value}
    Wait For Element Present    ${locator}  5    Element not found in 5 seconds
    Select From List By Label    ${locator}    ${value}
```

Now in above code Wait For Element Present function is called and we are passing ${locator} of drop-down , 5 second wait and error message Element not found in 5 seconds which is implemented just above which calls Wait Until Page Contains Element which I have already explained above.

Next in Select List Option By visible Text we are calling Select From List By Label which is SeleniumLibrary keyword

Select From List By Label:
http://robotframework.org/SeleniumLibrary/SeleniumLibrary.html#Select%20From%20List%20By%20Label

Select From List By Label	*locator*, **labels*	Selects options from selection list `locator` by `labels`. If more than one option is given for a single-selection list, the last value will be selected. With multi-selection lists all specified options are selected, but possible old selections are not cleared. See the *Locating elements* section for details about the locator syntax.

It locates the drop-down by locator and once found it looks for the sort text which is Price: High to Low

Below is final code so far covered for all scenarios added

Code: HomePageHelper.robot

```
*** Settings ***
Resource    ../PageObjects/HomePageObjects.robot
Resource    ../Utility/SeleniumKeywords.robot
Library     BuiltIn

*** Keywords ***
Click on search link
    Scroll for element  ${SearchLink}
    Click On Page Element   ${SearchLink}

Enter value to search
    [Arguments]     ${inputText}
    Set Value For Input Field   ${SearchField}  ${inputText}

Click on the Search button
    Click On Page Element   ${SearchButton}

Scroll the screen to top
    Scroll at the top of the page
    sleep   2s

Click Add to cart for first item
    Scroll for element  ${AddToCart}
    Click On Page Element   ${AddToCart}

Item has added to cart sucessfully
    Wait For Element Present    ${AddedToCart}  10  Item not added to cart
    Click On Page Element   ${CloseNotification}

Verify Cart displayed item
    Mouse Over on the element   ${ShoppingCartLink}
    Verify Element Present  ${AddedItemCount}

Sort item by
    [Arguments]     ${visibleText}
    Select List Option By visible Text  ${SortBy_Dropdown}  ${visibleText}
```

Code: LoginHelper.robot

```
*** Settings ***
Resource    ../PageObjects/LoginObjetcs.robot
Resource    ../Utility/SeleniumKeywords.robot
Library     BuiltIn

*** Keywords ***
Click on Login link
    Click On Page Element    ${LoginLink}

Enter username
    [Arguments]     ${value}
    Sleep   2s
    Set Value For Input Field  ${EmailField}   ${value}
```

```
Enter password
    [Arguments]    ${value}
    Set Value For Input Field  ${PasswordField}    ${value}

Click on Login button
    Click On Page Element    ${LoginButton}

Verify error message displayed
    Wait For Element Present    ${InvalidLoginError}    5    Error message not displayed
    Verify Text Present    ${InvaliLoginText}

Verify User is at Home page
    Verify Element Displayed    ${LogOut}
    URL Should Contain  demo.nopcommerce.com
```

Code:HomePageObjects.robot

```
*** Variables ***
${SearchLink}   css=a[href='/search']
${SearchField}  id=q
${SearchButton}     css=div.buttons>input[value='Search']
${AddToCart}    xpath=(//input[@value='Add to cart'])[1]
${AddedToCart}  xpath=//p[contains(text(),'The product has been added to your ')]/a[text()='shopping cart']
${CloseNotification}    css=div.bar-notification>span[title='Close']
${ShoppingCartLink}     css=li#topcartlink>a
${AddedItemCount}   xpath=//div[@id='flyout-cart']//div[@class='count' and contains(text(), 'There are')]/a[text()='1 item(s)']
${SortBy_Dropdown}  id=products-orderby
```

Code: LoginObjects.robot

```
*** Variables ***
${LoginLink}    css=a[href='/login?returnUrl=%2F']
${EmailField}   id=Email
${PasswordField}    id=Password
${LoginButton}  css=input[type='submit'][value='Log in']
${InvalidLoginError}    xpath=//div[contains(@class,'validation-summary-errors') and text()='Login was unsuccessful. Please correct the errors and try again.']
${InvaliLoginText}      Login was unsuccessful. Please correct the errors and try again.
${LogOut}       css=a[href='/logout']
```

Code: LoginTest.robot

```
*** Settings ***
Resource    ../Utility/Setup.robot
Resource    ../Helper/LoginHelper.robot
Resource    ../TestData/ApplicationProperties.robot
Suite Setup  Open the browser in
Test Setup  Go back to homepage
Test Teardown  Capture screenshot if test failed   Dummy
Suite Teardown  Close All Browser Window
```

```
*** Test Cases ***
Invalid Login
   [Tags]  Invalid
   Click on Login link
   Enter username  ${UserName}
   Enter password  ${InvalidPassword}
   Click on Login button
   Verify error message displayed
   Take screenshot     Default

Valid Login Test
   [Tags]  Valid
   Click on Login link
   Enter username  ${UserName}
   Enter password  ${Password}
   Click on Login button
   Verify User is at Home page
```

Code:SearchAddToCart.robot

```
*** Settings ***
Resource    ../Utility/Setup.robot
Resource    ../Helper/HomePageHelper.robot
Resource    ../TestData/ApplicationProperties.robot
Suite Setup  Open the browser in
Test Setup  Go back to homepage
Test Teardown  Capture screenshot if test failed   Dummy
Suite Teardown  Close All Browser Window

*** Test Cases ***
Search and Add to cart
   Click on search link
   Enter value to search   ${SearchItem}
   Click on the Search button
   Sort item by    Price: High to Low
   Click Add to cart for first item
   Scroll the screen to top
   Take screenshot     AddedItem
   Item has added to cart sucessfully
   Verify Cart displayed item
```

Code: ApplicationProperties.robot

```
*** Variables ***
${URL}    https://demo.nopcommerce.com/
${UserName}     waheedahmed55@gmail.com
${InvalidPassword}     test1
${Password}     test123
${SearchItem}   Laptop
```

Code: SeleniumKeywords.robot

```
*** Settings ***
Library    SeleniumLibrary

*** Keywords ***
Clear Input Field
   [Arguments]   ${locator}
   Wait For Element Present   ${locator}  5   Element not found in 5 seconds
   Clear Element Text   ${locator}
   ${value}=    Get Element Attribute   ${locator}    value
   ${backspaces count}=   Get Length    ${value}
   Run Keyword If   """${value}""" != ''
   ...    Repeat Keyword  ${backspaces count +1}  Press Key  ${locator}   \\08

Set Value For Input Field
   [Arguments]   ${locator}   ${inputval}
   sleep   2s
   Clear Input Field   ${locator}
   Input Text   ${locator}   ${inputval}

Click On Page Element
   [Arguments]   ${locator}
   Wait For Element Present   ${locator}  5   Element not found in 5 seconds
   Click Element   ${locator}

Wait For Page Load
   [Arguments]   ${pgloadlocator}
   Wait Until Page Contains Element   ${pgloadlocator}
   Sleep   5s

Wait For Element Present
   [Arguments]   ${locator}   ${timeout}   ${err_msg}
   #Wait For Element Present   ${locator}  5   Element not found in 5 seconds
   Wait Until Page Contains Element   ${locator}   ${timeout}   ${err_msg}

Get Attribute Value
   [Arguments]   ${locator}   ${attribute}
   #Wait For Element Present   ${locator}  5   Element not found in 5 seconds
   ${string}=   Get Element Attribute   ${locator}   ${attribute}
   log to console   \n get attribute ${string}.\n
   [Return]   ${string}

Back On Screen
   Go Back

Wait For Text Present
   [Arguments]   ${text}   ${seconds}   ${message}
   Wait Until Page Contains   ${text}   ${seconds}   ${message}

Verify Text Present
   [Arguments]   ${text}
   Page Should Contain   ${text}

Verify Element Present
```

```
    [Arguments]    ${locator}
    #Wait For Element Present    ${locator}  5    Element not found in 5 seconds
    Page Should Contain Element    ${locator}

Generate Random String With Defined Size
    [Arguments]    ${size}    ${type}
    ${str}=    Generate Random String    ${size}    ${type}
    [Return]    ${str}

Generate Random Number With Defined Size
    [Arguments]    ${size}    ${type}
    ${num}=    Generate Random String    ${size}    ${type}
    [Return]    ${num}

Match Value
    [Arguments]    ${locator}    ${attr_name}    ${match_pattern}
    #Wait For Element Present    ${locator}  5    Element not found in 5 seconds
    Element Attribute Should Match    ${locator}    ${attr_name}    ${match_pattern}

Verify Text Not Present
    [Arguments]    ${text}
    Page Should Not Contain    ${text}

Select List Option By visible Text
    [Arguments]    ${locator}    ${value}
    Wait For Element Present    ${locator}  5    Element not found in 5 seconds
    Select From List By Label    ${locator}    ${value}

Verify Page title
    [Arguments]    ${title}
    Title Should Be    ${title}

Navigate to the page
    [Arguments]    ${URL}
    Go To    ${URL}

Verify Element Displayed
    [Arguments]    ${locator}
    Wait For Element Present    ${locator}  5    Element not found in 5 seconds
    Element Should Be Visible    ${locator}

Verify Element not displayed
    [Arguments]    ${locator}
    Element Should Not Be Visible    ${locator}

Verify Current URL should contains
    [Arguments]    ${URL}
    Location Should Contain    ${URL}

Static wait for
    [Arguments]    ${Second}
    Sleep    ${Second}s

Get all element count
    [Arguments]    ${locator}
```

```
    ${num}=    Get Element Count    ${locator}
    [Return]    ${num}

Verify both values are equal
    [Arguments]    ${val1}  ${val2}
    Should Be Equal    ${val1}    ${val2}

Switch to window
    [Arguments]    ${type}
    Select Window  ${type}

Close current window
    Close Window

Get Current URL
    return from keyword    Get Location

Press keyboard key
    [Arguments]    ${locator}   ${Key}
    Wait For Element Present    ${locator}  5    Element not found in 5 seconds
    Press Key  ${locator}   \\08

Mouse Over on the element
    [Arguments]    ${locator}
    Wait For Element Present    ${locator}  5    Element not found in 5 seconds
    Mouse Over  ${locator}

URL Should Contain
    [Arguments]    ${expected}
    Location Should Contain    ${expected}

Scroll for element
    [Arguments]    ${locator}
    Scroll Element Into View    ${locator}

Scroll at the bottom of the page
    Execute Javascript    window.scrollTo(0,document.body.scrollHeight)

Scroll at the top of the page
    Execute Javascript    window.scrollTo(0,0)
```

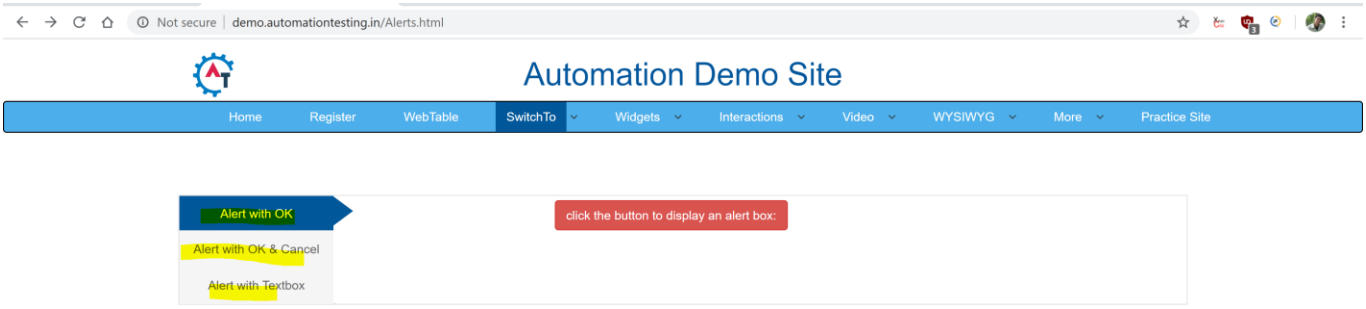
Code: Setup.robot

```
*** Settings ***
Library     SeleniumLibrary
Resource    ../TestData/ApplicationProperties.robot
Library  OperatingSystem
Library  BuiltIn
Library  String

*** Keywords ***
Open the browser in
   Open Browser    ${URL}  Chrome
   Maximize Browser Window
   Set Selenium Timeout    10s

Close All Browser Window
      Close All Browsers

Go back to homepage
   Go To   ${URL}

Capture screenshot if test failed
   [Arguments]  ${FileName}
   ${now}    Evaluate
'{dt.second}{dt.minute}{dt.hour}{dt.day}{dt.month}{dt.year}'.format(dt=datetime.datetime.now())
modules=datetime
   Run Keyword If Test Failed      Capture Page Screenshot
${OUTPUTDIR}/ScreenShot/${FileName}_${now}.png

Take screenshot
   [Arguments]  ${FileName}
   ${now}    Evaluate
'{dt.second}{dt.minute}{dt.hour}{dt.day}{dt.month}{dt.year}'.format(dt=datetime.datetime.now())
modules=datetime
   Capture Page Screenshot    ${OUTPUTDIR}/ScreenShot/${FileName}_${now}.png
```

Save All files . Run the test

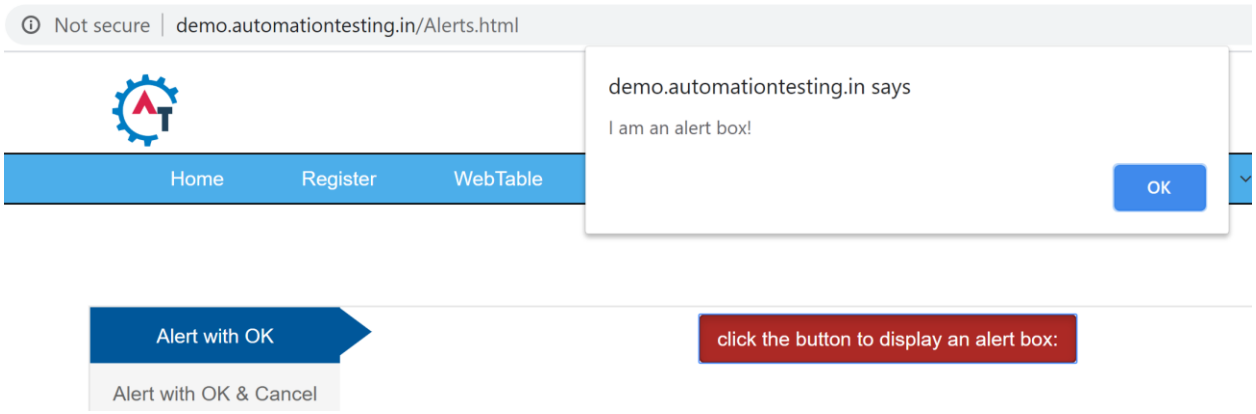
Handle Alerts and Frame:

Now we will look into how to handle Alerts first, for this this section I will be using this website http://demo.automationtesting.in/Alerts.html . As you can see below are three different types of Alerts



1. Alert with ok

As you click on red button “click the button to display on alert box” it will show pop up we will handle that using SeleniumLibrary keyword Handle Alert with Action to ACCEPT. Keep in mind these alerts are not part of html code so you wont be able to locate them, that’s why they have developed this keyword to handle



Create new Test file under TestCase HandleAndFrame.robot

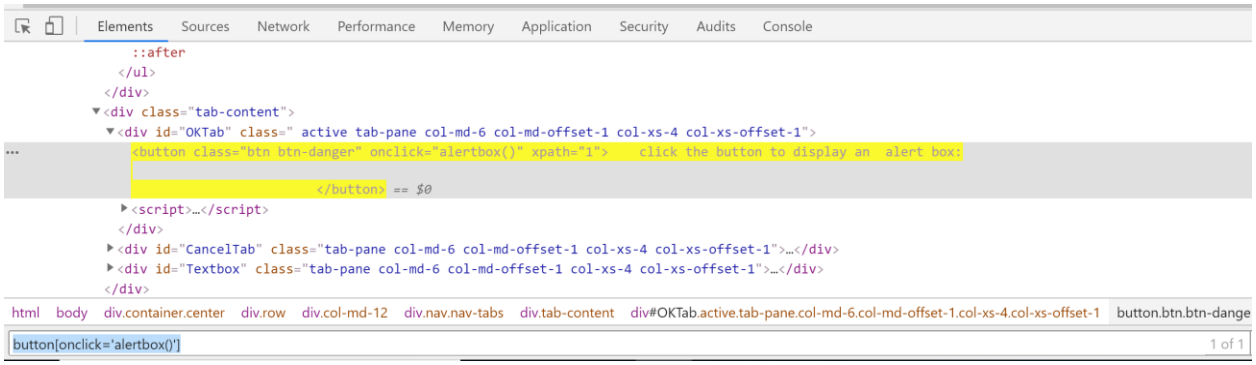
Create new helper file under Helper directory HomePageHelper.robot

Create new page object file under PageObjects directory HandleObjects.robot

First you will need to locate that red button on which we have to click, find its locator.

I was able to locate the button with css button[onclick='alertbox()']

Syntax: tag[attribute=’value’]

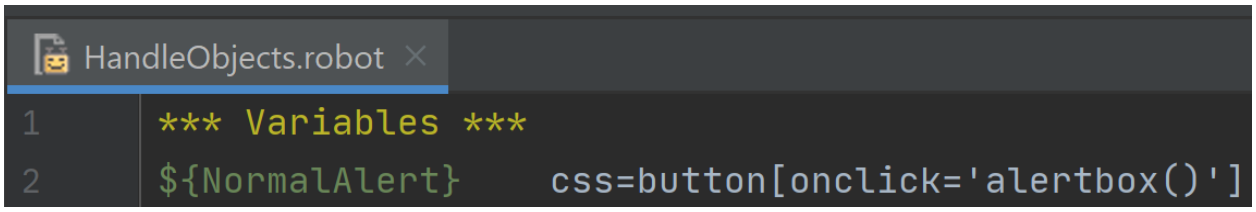


Tag = button

Attribute = onclick

Value = ‘alertbox()’

Lets add this locator in HandleObjects.robot



Code: HandleObjects.robot

```
*** Variables ***
${NormalAlert}    css=button[onclick='alertbox()']
${AlertWithCancelLink}  css=a[href='#CancelTab']
```

Now we will write helper function to click on this button and handle alert pop-up

Lets go test file HandleAndFrame.robot and write test steps and then after that we will implement in HandleHelper.robot

HandleAndFrame.robot

```
*** Settings ***
Resource    ../Utility/Setup.robot
Resource    ../Helper/HandleHelper.robot
Resource    ../TestData/ApplicationProperties.robot
Suite Setup  Open the browser in
Test Setup  Go back to homepage
Test Teardown  Capture screenshot if test failed   Dummy
Suite Teardown  Close All Browser Window

*** Test Cases ***
Verify Alert
    Navigate to the site    ${AlertPageUrl}
    Click on the alert button
    Accept the alert
```

So you can see in above screenshot have created test case Verify Alert in which first I will navigate to that site , than I will click on the red alert button and will handle alert using Handle Accept SeleniumLibrary keyword.

Code: HandleAndFrame.robot

```
*** Settings ***
Resource    ../Utility/Setup.robot
Resource    ../Helper/HandleHelper.robot
Resource    ../TestData/ApplicationProperties.robot
Suite Setup  Open the browser in
Test Setup  Go back to homepage
Test Teardown  Capture screenshot if test failed   Dummy
Suite Teardown  Close All Browser Window

*** Test Cases ***
Verify Alert
    Navigate to the site    ${AlertPageUrl}
    Click on the alert button
    Accept the alert
```

Here we created variable ${AlertPageUrl} in ApplicationProperties.robot

ApplicationProperties.robot

```
*** Variables ***
${URL}    https://demo.nopcommerce.com/
${UserName}     waheedahmed55@gmail.com
${InvalidPassword}     test1
${Password}     test123
${SearchItem}   Laptop
${AlertPageUrl}     http://demo.automationtesting.in/Alerts.html
```

Now lets go to HandleHelper.robot and implement the test steps.

HandleHelper.robot

```
*** Settings ***
Resource    ../PageObjects/HandleObjects.robot
Resource    ../Utility/SeleniumKeywords.robot
Library     BuiltIn

*** Keywords ***
Navigate to the site
    [Arguments]     ${URL}
    Navigate to the page    ${URL}

Click on the alert button
    sleep   3s
    Click On Button   ${NormalAlert}
    sleep   3s
Accept the alert
    Handle the alert box    ACCEPT
```

As you can see in above first we will implemented Navigate to the site in which we are calling function Navigate to the page and passing the ${URL} from the test step , this function is implemented in SeleniumKeywords.robot.

Lets look into it. As you can see below its calling Selenium Library keyword Go To with arugment ${URL} which is being passed from function, it will navigate the browser to the provided url which in our case is http://demo.automationtesting.in/Alerts.html coming from ${AlertPageUrl}

```
Navigate to the page
    [Arguments]    ${URL}
    Go To    ${URL}
```

Go To: https://robotframework.org/SeleniumLibrary/SeleniumLibrary.html#Go%20To

Go To	*url*	Navigates the current browser window to the provided `url`.

Next we implemented Click on the alert button in this function I am Click on Button function from SeleniumKeywords.robot

```
Click On Button
    [Arguments]    ${locator}
    Wait For Element Present    ${locator}  5    Element not found in 5 seconds
    Click Button    ${locator}
```

Here we are calling Click Button SeleniumLibrary keyword used for clicking on buttons with provided locator which we passed from function ${NormalAlert}

Click Button: https://robotframework.org/SeleniumLibrary/SeleniumLibrary.html#Click%20Button

Click Button	*locator*, *modifier: bool=False*	Clicks the button identified by `locator`. See the *Locating elements* section for details about the locator syntax. When using the default locator strategy, buttons are searched using `id`, `name`, and `value`. See the *Click Element* keyword for details about the `modifier` argument. The `modifier` argument is new in SeleniumLibrary 3.3

Last step was to handle alert , we implemented Accept the Alert function . In this function we are calling Handle the Alert box function with passing ACCEPT from SeleniumKeywords.robot. Lets look into the SeleniumKeywords.robot

```
Handle the alert box
    [Arguments]    ${action}
    Handle Alert    ${action}
```

Now here we are calling SeleniumLibrary keyword Handle Alert with arugment ${action} which we are passing from function which in our case is ACCEPT.

There three default actions that can be performed to handle : ACCEPT, DISMISS and LEAVE

Handle Alert: https://robotframework.org/SeleniumLibrary/SeleniumLibrary.html#Handle%20Alert

Handle Alert | *action=ACCEPT, timeout: NoneType=None*

Handles the current alert and returns its message.

By default, the alert is accepted, but this can be controlled with the `action` argument that supports the following case-insensitive values:

- `ACCEPT` : Accept the alert i.e. press `Ok` . Default.
- `DISMISS` : Dismiss the alert i.e. press `Cancel` .
- `LEAVE` : Leave the alert open.

The `timeout` argument specifies how long to wait for the alert to appear. If it is not given, the global default *timeout* is used instead.

Examples:

Handle Alert			# Accept alert.
Handle Alert	action=DISMISS		# Dismiss alert.
Handle Alert	timeout=10 s		# Use custom timeout and accept alert.
Handle Alert	DISMISS	1 min	# Use custom timeout and dismiss alert.
${message} =	Handle Alert		# Accept alert and get its message.
${message} =	Handle Alert	LEAVE	# Leave alert open and get its message.

Code: HandleHelper.robot

```
*** Settings ***
Resource    ../PageObjects/HandleObjects.robot
Resource    ../Utility/SeleniumKeywords.robot
Library     BuiltIn

*** Keywords ***
Navigate to the site
    [Arguments]    ${URL}
    Navigate to the page    ${URL}

Click on the alert button
    sleep   3s
    Click On Button   ${NormalAlert}
    sleep   3s
Accept the alert
    Handle the alert box    ACCEPT
```

& the code we added in SeleniumKeywords.robot

```
Navigate to the page
    [Arguments]    ${URL}
    Go To   ${URL}
Click On Button
    [Arguments]    ${locator}
    Wait For Element Present    ${locator}  5    Element not found in 5 seconds
    Click Button    ${locator}
Handle the alert box
    [Arguments]    ${action}
    Handle Alert    ${action}
```

Now you can save all files and run the test. I am going to run only this test file

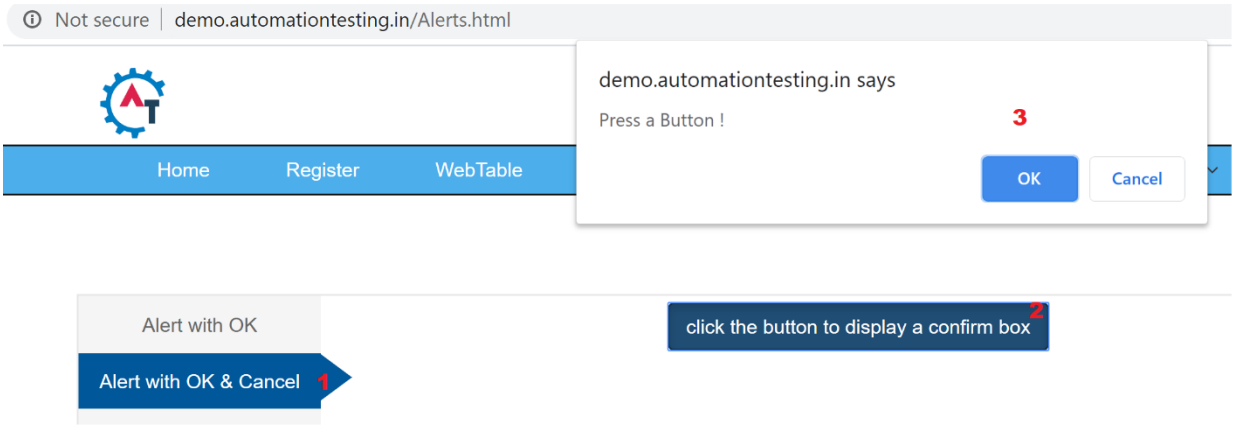
robot -d "Reports" TestCases/HandleAndFrame.robot

```
Terminal:  Local  Local (2)  +

(venv) C:\Users\user\PycharmProjects\robotframeworkautomation>robot -d "Reports" TestCases/HandleAndFrame.robot
```

```
DevTools listening on ws://127.0.0.1:57224/devtools/browser/039f1ee2-c808-49f5-97ec-86c6a1f6ac14
[16996:15488:0513/122540.533:ERROR:browser_switcher_service.cc(238)] XXX Init()
Verify Alert                                                          | PASS |
------------------------------------------------------------------------------
HandleAndFrame                                                        | PASS |
1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
==============================================================================
Output:  C:\Users\user\PycharmProjects\robotframeworkautomation\Reports\output.xml
Log:     C:\Users\user\PycharmProjects\robotframeworkautomation\Reports\log.html
Report:  C:\Users\user\PycharmProjects\robotframeworkautomation\Reports\report.html
```

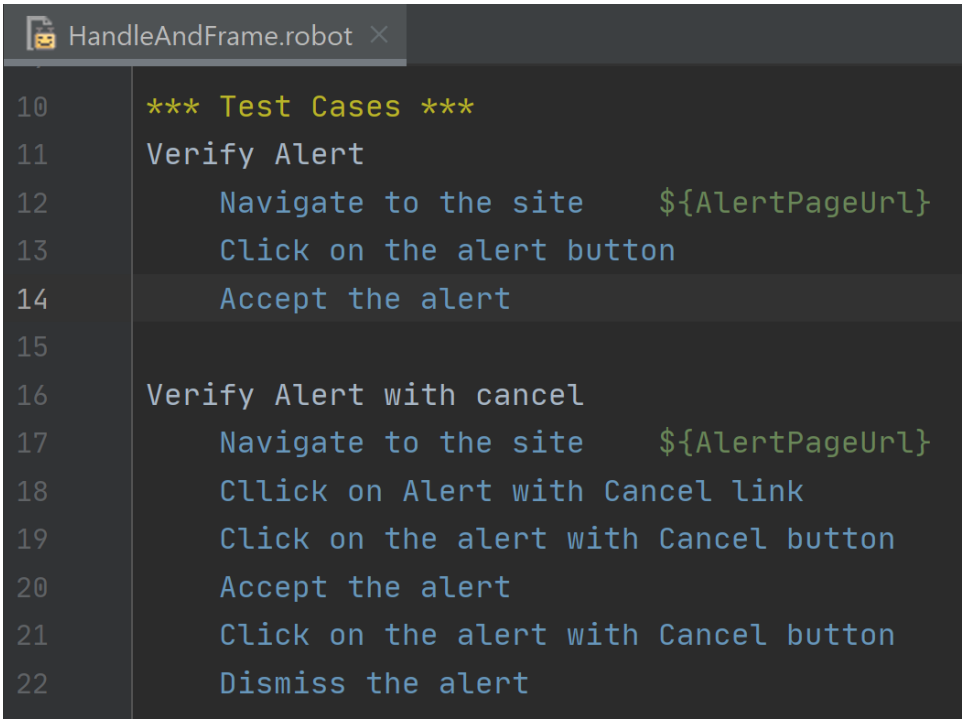
Now lets handle the 2 type of Alet with OK and Cancel button

2. Alert with OK & Cancel



Let see scenario steps. First we will click on Alert with OK & Cancel and then click on blue button “click the button to display a confirm box and finally we will see popup and handle here we will perform both ACCEPT and DISMISS actions on the alert.

Here are steps for this in HandleAndFrame.robot



First we will navigate to the site and click on “Alert with OK & cancel” and then click on blue button “click the button to display a confirm box” and finally we will ACCEPT the alert and than we will click on the blue button again but this time we will DISMISS

Code: HandleAndFrame.robot

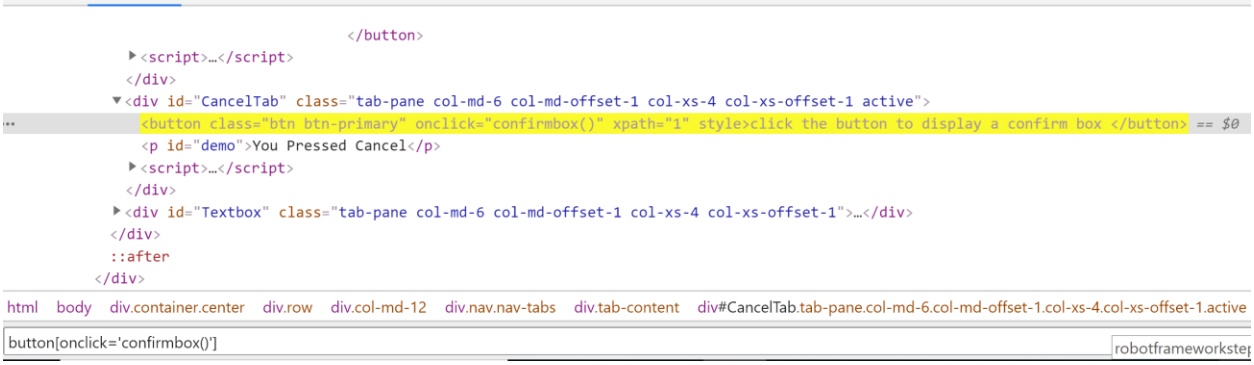
```
*** Settings ***
Resource    ../Utility/Setup.robot
Resource    ../Helper/HandleHelper.robot
Resource    ../TestData/ApplicationProperties.robot
Suite Setup  Open the browser in
Test Setup  Go back to homepage
Test Teardown  Capture screenshot if test failed   Dummy
Suite Teardown  Close All Browser Window

*** Test Cases ***
Verify Alert
   Navigate to the site    ${AlertPageUrl}
   Click on the alert button
   Accept the alert

Verify Alert with cancel
   Navigate to the site    ${AlertPageUrl}
   Cllick on Alert with Cancel link
   Click on the alert with Cancel button
   Accept the alert
   Click on the alert with Cancel button
   Dismiss the alert
```

Before we proceed to implement lets locate objects for “Alert with OK & Cancel “ and button “click the button to display a confirm box”

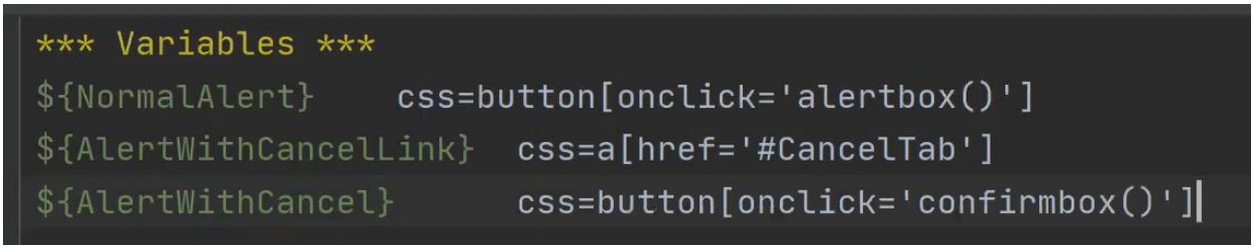
HandleObjects.robot

Alert with OK and Cancel link:

click the button to display a confirm box:



Now add these two to the HandleObjects.robot



Code: HandleObjects.robot

```
*** Variables ***
${NormalAlert}    css=button[onclick='alertbox()']
${AlertWithCancelLink}  css=a[href='#CancelTab']
${AlertWithCancel}      css=button[onclick='confirmbox()']
```

Now lets implement these steps in HandleHelper.robot

The first step is same so wont be covering lets see implementation Cllick on Alert with Cancel link

```
Cllick on Alert with Cancel link
    Click On Page Element    ${AlertWithCancelLink}
```

In this function we are calling Click On Page Element and passing ${AlertWithCancelLink} function from SeleniumKeywords.robot We have already discussed this function so we will move on to the next step. All it does is click on the link “Alert with OK & Cancel”

Next we will look into implementation of Click on the alert with Cancel button

```
Click on the alert with Cancel button
    sleep    3s
    Click On Button    ${AlertWithCancel}
    sleep    3s
```

Here you can see we are calling function Click On Button and passing ${AlertWithCancel} which is locator of the blue button “click the button to display a confirm box” from SeleniumKeywords.robot. Lets look in SeleniumKeywords.robot

```
Click On Button
    [Arguments]    ${locator}
    Wait For Element Present    ${locator}  5    Element not found in 5 seconds
    Click Button    ${locator}
```

We are already discussed Wait For Element Present previously. Lets see Click Button ${locator} which SeleniumLibrary keyword

Here we are calling Click Button SeleniumLibrary keyword used for clicking on buttons with provided locator which we passed from function ${AlertWithCancel}

Click Button: https://robotframework.org/SeleniumLibrary/SeleniumLibrary.html#Click%20Button

Click Button	*locator, modifier: bool=False*	Clicks the button identified by `locator`. See the *Locating elements* section for details about the locator syntax. When using the default locator strategy, buttons are searched using `id`, `name`, and `value`. See the *Click Element* keyword for details about the `modifier` argument. The `modifier` argument is new in SeleniumLibrary 3.3

Code: HandleHelper.robot

```
*** Settings ***
Resource    ../PageObjects/HandleObjects.robot
Resource    ../Utility/SeleniumKeywords.robot
Library     BuiltIn

*** Keywords ***
Navigate to the site
    [Arguments]    ${URL}
    Navigate to the page    ${URL}

Click on the alert button
    sleep    3s
    Click On Button    ${NormalAlert}
    sleep    3s
Accept the alert
    Handle the alert box    ACCEPT
Cllick on Alert with Cancel link
    Click On Page Element    ${AlertWithCancelLink}
Click on the alert with Cancel button
    sleep    3s
    Click On Button    ${AlertWithCancel}
    sleep    3s
```

Next step which we have implement is Accept the alert which is already implemented (we implemented in previous test case) . So I wont be covering that here.

Lets implement Click on the alert with Cancel button step in HandleHelper.robot but this also already implemented (we implemented in previous test case). So I wont be covere that here

Let see the implementation of last step Dismiss the alert in HandleHelper.robot . Remember we discussed that second we will perform DISMISS action thus we will implement method similar to Accept the alert but this time we will set action as DISMISS

```
Dismiss the alert
    Handle the alert box    DISMISS
```

Now here you can see we are calling same function Handle the alert box from SeleniumKeywords.robot but this time we are passing it DISMISS action.

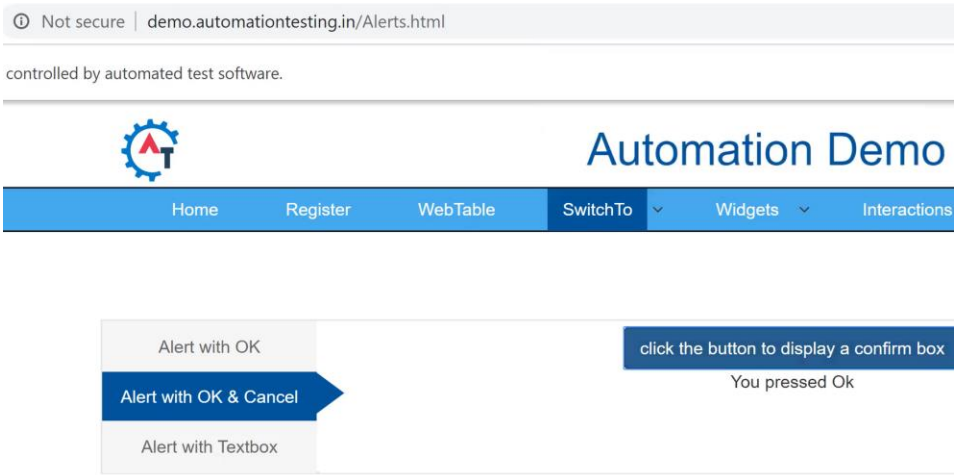
We already discussed the implementation of Handle the alert box previously so lets move on.

Code:HandleHelper.robot

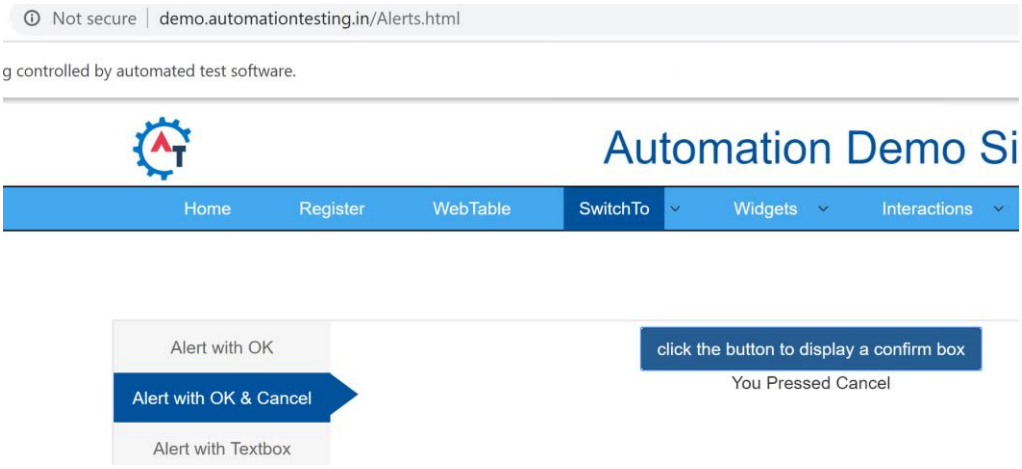
```
*** Settings ***
Resource    ../PageObjects/HandleObjects.robot
Resource    ../Utility/SeleniumKeywords.robot
Library     BuiltIn

*** Keywords ***
Navigate to the site
    [Arguments]    ${URL}
    Navigate to the page    ${URL}

Click on the alert button
    sleep    3s
    Click On Button    ${NormalAlert}
    sleep    3s
Accept the alert
    Handle the alert box    ACCEPT
Cllick on Alert with Cancel link
    Click On Page Element    ${AlertWithCancelLink}
Click on the alert with Cancel button
    sleep    3s
    Click On Button    ${AlertWithCancel}
    sleep    3s
Dismiss the alert
    Handle the alert box    DISMISS
```

Now save all files and run the test.

You will see first it will perform ACCEPT which is clickin OK on Alert pop-up and then perform DISMISS which press cancel on pop-alert.



& then

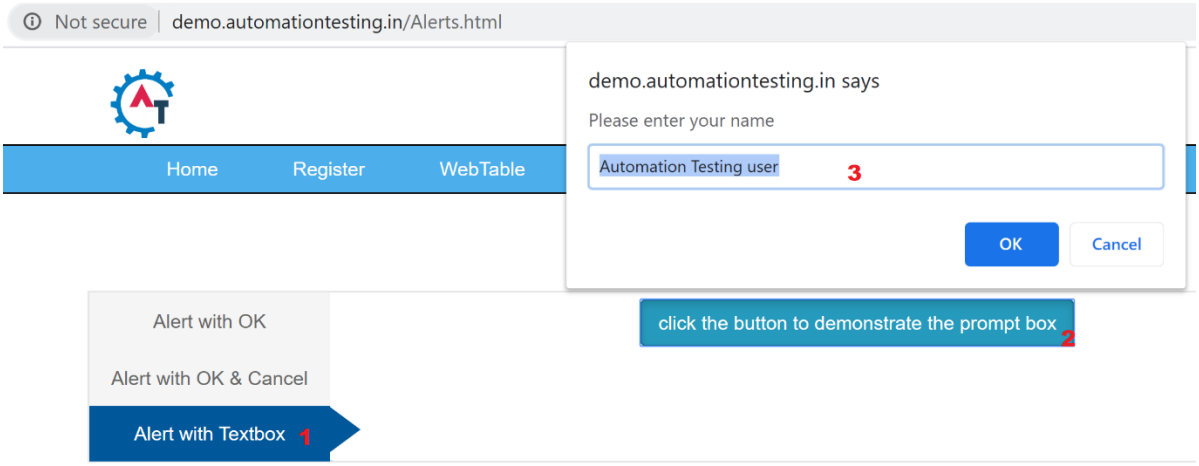



```
Terminal:  Local  Local (2)  +
[15880:12084:0514/104415.938:ERROR:browser_switcher_service.cc(238)] XXX Init()
Verify Alert                                                          | PASS |
------------------------------------------------------------------------------
Verify Alert with cancel                                              | PASS |
------------------------------------------------------------------------------
HandleAndFrame                                                        | PASS |
2 critical tests, 2 passed, 0 failed
2 tests total, 2 passed, 0 failed
==============================================================================
Output:  C:\Users\user\PycharmProjects\robotframeworkautomation\Reports\output.xml
Log:     C:\Users\user\PycharmProjects\robotframeworkautomation\Reports\log.html
Report:  C:\Users\user\PycharmProjects\robotframeworkautomation\Reports\report.html
```

Now lets look at third type of Alert pop-up Alert with textbox

3. Alert with Textbox

Lets see the steps first we will navigate to web site and then we will click on “Alert with Textbox” and click on button “click the button to demonstrate the prompt box” and finally enter the text in alert. Here we need to understand SeleniumLibrary we will use will by default accept the pop-up as we enter text.



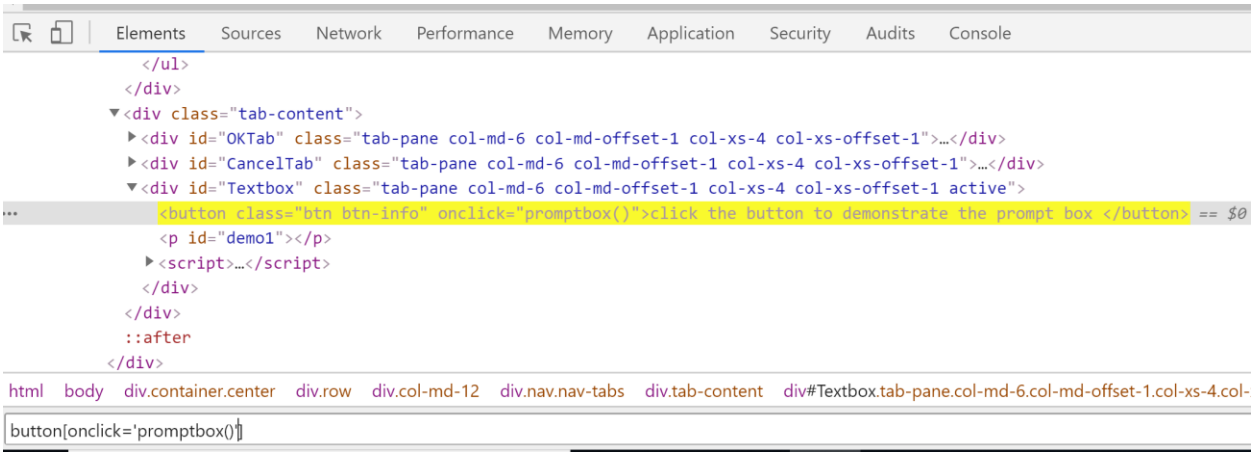
Code: HandleAndFrame.robot

```
*** Settings ***
Resource    ../Utility/Setup.robot
Resource    ../Helper/HandleHelper.robot
Resource    ../TestData/ApplicationProperties.robot
Suite Setup  Open the browser in
Test Setup  Go back to homepage
Test Teardown  Capture screenshot if test failed   Dummy
Suite Teardown  Close All Browser Window

*** Test Cases ***
Verify Alert
   Navigate to the site    ${AlertPageUrl}
   Click on the alert button
   Accept the alert

Verify Alert with cancel
   Navigate to the site    ${AlertPageUrl}
   Cllick on Alert with Cancel link
   Click on the alert with Cancel button
   Accept the alert
   Click on the alert with Cancel button
   Dismiss the alert

Verify Alert with Textbox
   Navigate to the site    ${AlertPageUrl}
   Click on Alert with Textbox link
   Click on the alert with textbox button
   Enter text in alert     Dummy Alert Text
```

Lets add the locators of for the “Alert with Textbox” which is link and button “click the button to demonstrate the prompt box”

Alert with Textbox link:

click the button to demonstrate the prompt box:



Lets add them into HandleObjects.robot

Code: HandleObjects.robot

```
*** Variables ***
${NormalAlert}    css=button[onclick='alertbox()']
${AlertWithCancelLink}  css=a[href='#CancelTab']
${AlertWithCancel}     css=button[onclick='confirmbox()']
${AlertWithTextBoxLink}     css=a[href='#Textbox']
${AlertWithTextbox}     css=button[onclick='promptbox()']
```

Lets see the implementation of these steps in HandleHelper.robot

I will skip the first step ”Navigate to the site” its already discussed and implemented . Lets go to second step Click on Alert with Textbox link

```
Click on Alert with Textbox link
    Click On Page Element    ${AlertWithTextBoxLink}
```

The Click on Alert with Textbox link is calling function Click On Page Element passing locator ${AlertWithTextBoxLink} to SeleniunKeywords.robot. I wont explain as its already covered previously .

Next step we will implement is Click on the alert with textbox button, its calling Click On Button passing ${AlertWithTextbox} locator function from SeleniumKeywords.robot

```
Click on the alert with textbox button
    Click On Button    ${AlertWithTextbox}
```

Lets look into SeleniumKeywords.robot

```
Click On Button
    [Arguments]    ${locator}
    Wait For Element Present    ${locator}  5    Element not found in 5 seconds
    Click Button    ${locator}
```

We are already discussed Wait For Element Present previously. Lets see Click Button ${locator} which SeleniumLibrary keyword

Here we are calling Click Button SeleniumLibrary keyword used for clicking on buttons with provided locator which we passed from function ${AlertWithTextbox}

Click Button: https://robotframework.org/SeleniumLibrary/SeleniumLibrary.html#Click%20Button

Click Button	*locator, modifier: bool=False*	Clicks the button identified by `locator`. See the *Locating elements* section for details about the locator syntax. When using the default locator strategy, buttons are searched using `id`, `name`, and `value`. See the *Click Element* keyword for details about the `modifier` argument. The `modifier` argument is new in SeleniumLibrary 3.3

Next let see we see implementation of Enter text in alert

```
Enter text in alert
    [Arguments]     ${text}
    Enter text in alert box     ${text}
    sleep   3s
```

Here you can see we are calling function Enter text in the alert box and passing ${text} from test step to SeleniumKeywords.robot.

```
Enter text in alert box
    [Arguments]     ${text}
    Input text into Alert   ${text}
```

We are using Input text into Alert SeleniumLibrary keywords , here interestingly you don't have to pass the action infact its befault perfoms ACCEPT as we set the text.

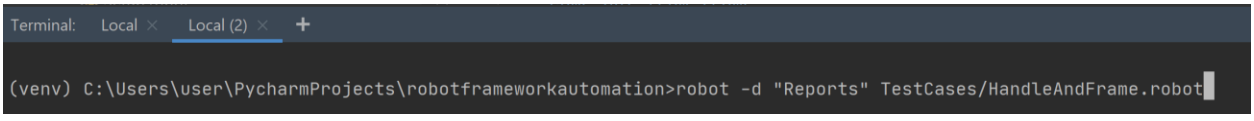
Input text into Alert:
https://robotframework.org/SeleniumLibrary/SeleniumLibrary.html#Input%20Text%20Into%20Alert

Input Text Into Alert	*text, action=ACCEPT, timeout=NoneType=None*	Types the given `text` into an input field in an alert. The alert is accepted by default, but that behavior can be controlled by using the `action` argument same way as with *Handle Alert*. `timeout` specifies how long to wait for the alert to appear. If it is not given, the global default *timeout* is used instead. New in SeleniumLibrary 3.0.

Code:HandleHelper.robot

```
*** Settings ***
Resource    ../PageObjects/HandleObjects.robot
Resource    ../Utility/SeleniumKeywords.robot
Library     BuiltIn

*** Keywords ***
Navigate to the site
    [Arguments]    ${URL}
    Navigate to the page    ${URL}

Click on the alert button
    sleep   3s
    Click On Button   ${NormalAlert}
    sleep   3s
Accept the alert
    Handle the alert box    ACCEPT
Cllick on Alert with Cancel link
    Click On Page Element   ${AlertWithCancelLink}
Click on the alert with Cancel button
    sleep   3s
    Click On Button   ${AlertWithCancel}
    sleep   3s
Dismiss the alert
    Handle the alert box    DISMISS
Click on Alert with Textbox link
    Click On Page Element   ${AlertWithTextBoxLink}
Click on the alert with textbox button
    Click On Button   ${AlertWithTextbox}
Enter text in alert
    [Arguments]    ${text}
    Enter text in alert box    ${text}
    sleep   3s
```

Now save files and run test.



Output:

```
==============================================================================
HandleAndFrame
==============================================================================

DevTools listening on ws://127.0.0.1:62988/devtools/browser/88b997b9-5319-4e4d-a794-de1b324836db
[15552:17360:0514/140313.663:ERROR:browser_switcher_service.cc(238)] XXX Init()
Verify Alert                                                          | PASS |
------------------------------------------------------------------------------
Verify Alert with cancel                                              | PASS |
------------------------------------------------------------------------------
Verify Alert with Textbox                                             | PASS |
------------------------------------------------------------------------------
[15548:16396:0514/140352.092:ERROR:broker_win.cc(55)] Error reading broker pipe: The pipe has been ended. (0x6D)
[15548:16396:0514/140352.096:ERROR:broker_win.cc(141)] Error sending sync broker message: The pipe is being closed. (0xE8)
[15548:16396:0514/140352.098:ERROR:broker_win.cc(141)] Error sending sync broker message: The pipe is being closed. (0xE8)
HandleAndFrame                                                        | PASS |
3 critical tests, 3 passed, 0 failed
3 tests total, 3 passed, 0 failed
==============================================================================
Output:  C:\Users\user\PycharmProjects\robotframeworkautomation\Reports\output.xml
Log:     C:\Users\user\PycharmProjects\robotframeworkautomation\Reports\log.html
Report:  C:\Users\user\PycharmProjects\robotframeworkautomation\Reports\report.html
```

Handle Frame:

Next we will see how to handle frame for this section I will be using this site http://demo.automationtesting.in/Frames.html

HTML frames are used to divide your browser window into multiple sections where each section can load a separate HTML document. A collection of frames in the browser window is known as a frameset. The window is divided into frames in a similar way the tables are organized: into rows and columns.

Thus, in order to access anything inside the frame for example input text field as you see below

We will need to switch into frame and access the field and set text into . That's exactly what we will do in our test.

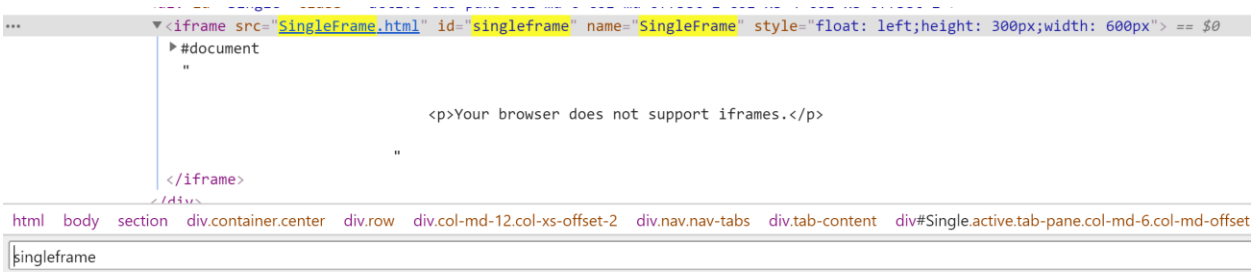
Before we write steps lets add the website url in ApplicationProperties.robot

Code: ApplicationProperties.robot

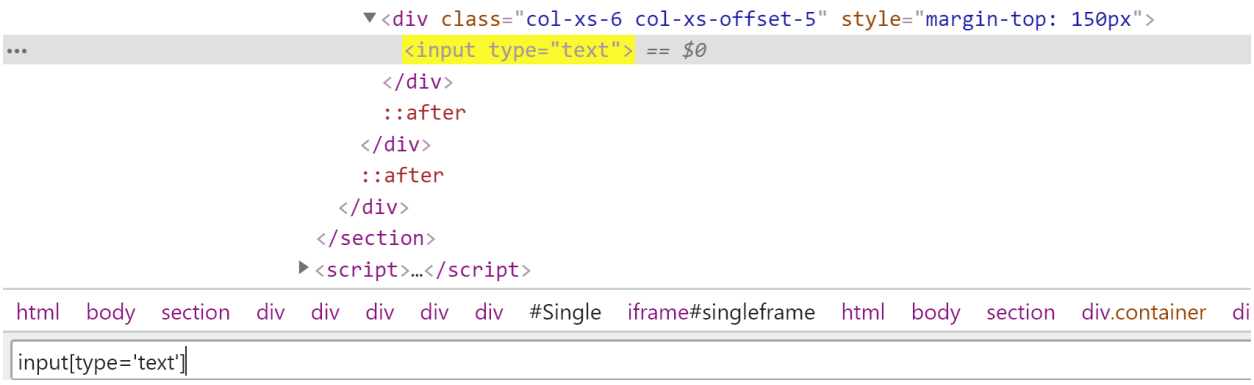
```
*** Variables ***
${URL}    https://demo.nopcommerce.com/
${UserName}    waheedahmed55@gmail.com
${InvalidPassword}    test1
${Password}    test123
${SearchItem}    Laptop
${AlertPageUrl}    http://demo.automationtesting.in/Alerts.html
${FrameUrl}    http://demo.automationtesting.in/Frames.html
```

Lets add the locator for frame and input text field.

Frame:



Inout text field:



Code: HandleObjects.robot

```
*** Variables ***
${NormalAlert}    css=button[onclick='alertbox()']
${AlertWithCancelLink}    css=a[href='#CancelTab']
${AlertWithTextBoxLink}    css=a[href='#Textbox']
${AlertWithCancel}    css=button[onclick='confirmbox()']
${AlertWithTextbox}    css=button[onclick='promptbox()']
${SingleFrame}    id=singleframe
${InputText}    css=input[type='text']
```

Lets write the steps in the HandleAndFrame.robot

Here are the steps to verify

```
Verify Frame
    Navigate to the site    ${FrameUrl}
    Switch to the frame
    Enter text in the field     Dummy Alert Text
    Come out from frame
```

Let see implementation of these steps in HandleHelper.robot . The first step is generic just in this test case its navigating to different url which is http://demo.automationtesting.in/Frames.html

Next step is Switch to the frame its calling Select a Frame passing locator of the frame to it

```
Switch to the frame
    Select a Frame  ${SingleFrame}
```

in SeleniumKeywords.robot . Here we can see below we are using Select Frame SeleniumLibrary keyword which locates the frame by locator and switch into it

```
Select a Frame
    [Arguments]  ${locator}
    Select Frame    ${locator}
```

Select Frame: https://robotframework.org/SeleniumLibrary/SeleniumLibrary.html#Select%20Frame

Select Frame	*locator*	Sets frame identified by `locator` as the current frame. See the *Locating elements* section for details about the locator syntax. Works both with frames and iframes. Use *Unselect Frame* to cancel the frame selection and return to the main frame. Example:

Select Frame	top-frame	# Select frame with id or name 'top-frame'
Click Link	example	# Click link 'example' in the selected frame
Unselect Frame		# Back to main frame.
Select Frame	//iframe[@name='xxx']	# Select frame using xpath

Once switched into you will be able to access the input text field. However anything outside the frame wont be accessible.

Next step is Enter text in the field let see its implementation

```
Enter text in the field
    [Arguments]     ${text}
    Set Value For Input Field   ${InputText}    ${text}
```

In this we are calling function Set Vaue For Input Field which we have already discussed previously .

Next step is important as we need to come out of frame as soon as we are done. Lets see how Come out from frame step is implemented:

```
Come out from frame
    UnSelect a Frame
```

Above you can see we are calling function UnSelect a Frame Lets look into SeleniumKeyword.robot , is called Unselect Frame Selenium Library keyword , what it does whatever frame we are in it will automatically take us out of it.

```
UnSelect a Frame
    Unselect Frame
```

Unselect Frame:
https://robotframework.org/SeleniumLibrary/SeleniumLibrary.html#Unselect%20Frame

Unselect Frame		Sets the main frame as the current frame. In practice cancels the previous *Select Frame* call.

Code: HandleHelper.robot

```
*** Settings ***
Resource    ../PageObjects/HandleObjects.robot
Resource    ../Utility/SeleniumKeywords.robot
Library     BuiltIn

*** Keywords ***
Navigate to the site
    [Arguments]     ${URL}
    Navigate to the page    ${URL}

Click on the alert button
    sleep   3s
    Click On Button   ${NormalAlert}
    sleep   3s
Accept the alert
    Handle the alert box    ACCEPT

Click on the alert with Cancel button
    sleep   3s
    Click On Button   ${AlertWithCancel}
    sleep   3s

Cllick on Alert with Cancel link
    Click On Page Element   ${AlertWithCancelLink}

Dismiss the alert
    Handle the alert box    DISMISS

Click on Alert with Textbox link
    Click On Page Element   ${AlertWithTextBoxLink}

Click on the alert with textbox button
    Click On Button   ${AlertWithTextbox}

Enter text in alert
    [Arguments]     ${text}
    Enter text in alert box     ${text}
    sleep   3s

Switch to the frame
    Select a Frame  ${SingleFrame}

Enter text in the field
    [Arguments]     ${text}
    Set Value For Input Field   ${InputText}    ${text}

Come out from frame
    UnSelect a Frame
```

Code : SeleniumKeywords.robot

```
*** Settings ***
Library    SeleniumLibrary

*** Keywords ***
Clear Input Field
   [Arguments]   ${locator}
   Wait For Element Present   ${locator}  5   Element not found in 5 seconds
   Clear Element Text   ${locator}
   ${value}=    Get Element Attribute   ${locator}    value
   ${backspaces count}=   Get Length    ${value}
   Run Keyword If   """${value}""" != ''
   ...    Repeat Keyword  ${backspaces count +1}  Press Key  ${locator}   \\08

Set Value For Input Field
   [Arguments]   ${locator}   ${inputval}
   sleep   2s
   Clear Input Field   ${locator}
   Input Text   ${locator}   ${inputval}

Click On Page Element
   [Arguments]   ${locator}
   Wait For Element Present   ${locator}  5   Element not found in 5 seconds
   Click Element   ${locator}

Click On Button
   [Arguments]   ${locator}
   Wait For Element Present   ${locator}  5   Element not found in 5 seconds
   Click Button   ${locator}

Wait For Page Load
   [Arguments]   ${pgloadlocator}
   Wait Until Page Contains Element   ${pgloadlocator}
   Sleep   5s

Wait For Element Present
   [Arguments]   ${locator}   ${timeout}   ${err_msg}
   #Wait For Element Present   ${locator}  5   Element not found in 5 seconds
   Wait Until Page Contains Element   ${locator}   ${timeout}   ${err_msg}

Get Attribute Value
   [Arguments]   ${locator}   ${attribute}
   #Wait For Element Present   ${locator}  5   Element not found in 5 seconds
   ${string}=   Get Element Attribute   ${locator}   ${attribute}
   log to console   \n get attribute ${string}.\n
   [Return]   ${string}

Back On Screen
   Go Back

Wait For Text Present
   [Arguments]   ${text}   ${seconds}   ${message}
   Wait Until Page Contains   ${text}   ${seconds}   ${message}
```

```
Verify Text Present
    [Arguments]    ${text}
    Page Should Contain    ${text}

Verify Element Present
    [Arguments]    ${locator}
    #Wait For Element Present    ${locator}  5    Element not found in 5 seconds
    Page Should Contain Element    ${locator}

Generate Random String With Defined Size
    [Arguments]    ${size}    ${type}
    ${str}=    Generate Random String    ${size}    ${type}
    [Return]    ${str}

Generate Random Number With Defined Size
    [Arguments]    ${size}    ${type}
    ${num}=    Generate Random String    ${size}    ${type}
    [Return]    ${num}

Match Value
    [Arguments]    ${locator}    ${attr_name}    ${match_pattern}
    #Wait For Element Present    ${locator}  5    Element not found in 5 seconds
    Element Attribute Should Match    ${locator}    ${attr_name}    ${match_pattern}

Verify Text Not Present
    [Arguments]    ${text}
    Page Should Not Contain    ${text}

Select List Option By visible Text
    [Arguments]    ${locator}    ${value}
    Wait For Element Present    ${locator}  5    Element not found in 5 seconds
    Select From List By Label    ${locator}    ${value}

Verify Page title
    [Arguments]    ${title}
    Title Should Be    ${title}

Navigate to the page
    [Arguments]    ${URL}
    Go To    ${URL}

Verify Element Displayed
    [Arguments]    ${locator}
    Wait For Element Present    ${locator}  5    Element not found in 5 seconds
    Element Should Be Visible    ${locator}

Verify Element not displayed
    [Arguments]    ${locator}
    Element Should Not Be Visible    ${locator}

Verify Current URL should contains
    [Arguments]    ${URL}
    Location Should Contain    ${URL}

Static wait for
```

```
    [Arguments]    ${Second}
    Sleep    ${Second}s

Get all element count
    [Arguments]    ${locator}
    ${num}=    Get Element Count    ${locator}
    [Return]    ${num}

Verify both values are equal
    [Arguments]    ${val1}  ${val2}
    Should Be Equal    ${val1}    ${val2}

Switch to window
    [Arguments]    ${type}
    Select Window  ${type}

Close current window
    Close Window

Get Current URL
    return from keyword    Get Location

Press keyboard key
    [Arguments]    ${locator}   ${Key}
    Wait For Element Present    ${locator}  5    Element not found in 5 seconds
    Press Key  ${locator}   \\08

Mouse Over on the element
    [Arguments]    ${locator}
    Wait For Element Present    ${locator}  5    Element not found in 5 seconds
    Mouse Over  ${locator}

URL Should Contain
    [Arguments]    ${expected}
    Location Should Contain    ${expected}

Scroll for element
    [Arguments]    ${locator}
    Scroll Element Into View    ${locator}

Scroll at the bottom of the page
    Execute Javascript    window.scrollTo(0,document.body.scrollHeight)

Scroll at the top of the page
    Execute Javascript    window.scrollTo(0,0)

Handle the alert box
    [Arguments]    ${action}
    Handle Alert    ${action}

Enter text in alert box
    [Arguments]    ${text}
    Input text into Alert   ${text}

Select a Frame
```

```
    [Arguments]  ${locator}
    Select Frame    ${locator}

UnSelect a Frame
    Unselect Frame
```

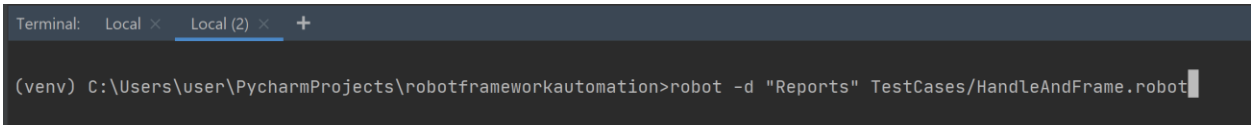
Code: HandleAndFrame.robot

```
*** Settings ***
Resource    ../Utility/Setup.robot
Resource    ../Helper/HandleHelper.robot
Resource    ../TestData/ApplicationProperties.robot
Suite Setup  Open the browser in
Test Setup  Go back to homepage
Test Teardown  Capture screenshot if test failed   Dummy
Suite Teardown  Close All Browser Window

*** Test Cases ***
Verify Alert
    Navigate to the site    ${AlertPageUrl}
    Click on the alert button
    Accept the alert

Verify Alert with cancel
    Navigate to the site    ${AlertPageUrl}
    Cllick on Alert with Cancel link
    Click on the alert with Cancel button
    Accept the alert
    Click on the alert with Cancel button
    Dismiss the alert

Verify Alert with Textbox
    Navigate to the site    ${AlertPageUrl}
    Click on Alert with Textbox link
    Click on the alert with textbox button
    Enter text in alert     Dummy Alert Text

Verify Frame
    Navigate to the site    ${FrameUrl}
    Switch to the frame
    Enter text in the field     Dummy Alert Text
    Come out from frame
```

Code: HandleObjects.robot

```
*** Variables ***
${NormalAlert}    css=button[onclick='alertbox()']
${AlertWithCancelLink}  css=a[href='#CancelTab']
${AlertWithTextBoxLink}     css=a[href='#Textbox']
${AlertWithCancel}      css=button[onclick='confirmbox()']
${AlertWithTextbox}      css=button[onclick='promptbox()']
${SingleFrame}      id=singleframe
${InputText}        css=input[type='text']
```

Save all files and run the tests.



Output:

Best practice for Automation

1. Page Object Model:

A popular test automation design pattern, the page object model will help you make robust testing frameworks that are resistant to small tweaks in the UI. The page object model has two core advantages:

i. There is clean separation between test code and page specific code such as locators (or their use if you're using a UI map) and layout.

ii. There is single repository for the services or operations offered by the page rather than having these services scattered throughout the tests.

2. Make Tests Independent Of Each Other:

Testing one single action or process, independent of any other tests is known as atomic testing. This testing strategy will keep you from making chained, brittle tests. Chaining together tests, while saving time upfront, can be a huge hindrance to an agile or CI workflow. Keep tests as small as possible.

3. Don't Automate Unstable Functionality:

As a new feature or functionality is being developed, many things can go wrong and even the feature may no longer be applicable because the business have changed their mind. If you started automating tests as the feature was being developed, the tests need to be updated many times as the feature evolves and can be quite daunting trying to keep up with all the changes. And if the feature is no longer applicable, all that effort on test automation is wasted.

4. Don't Automate Every Test:

100% Test Coverage is not possible since there can be millions of combinations. We always execute a subset of possible tests. The same principle applies to automate testing.

5. **Use the Right Locators:**

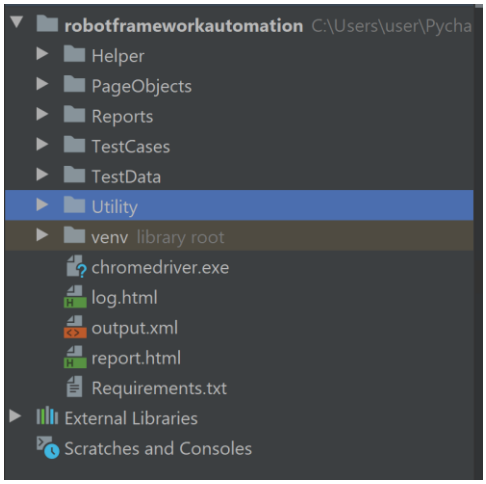
   At the heart of the Robot framework is interaction with the browser, letting you navigate, click, type and check different objects within the DOM using a few set actions. It does this using several different types of locators:

   a. CSS
   b. ID
   c. TagName
   d. Name
   e. Xpath

   Selecting the right locators can be the difference between a test script that is flexible and successful, and a brittle test that breaks on the slightest UI change

6. **NEVER use Thread.sleep() unless there are specific test requirements:**
7. **Do not run ALL tests across ALL target browser**
8. **Name your tests wisely**
9. **Take screenshots for failure investigation**
10. **Make tests simpler instead of adding comments**
11. **Follow the “green tests run” policy**
12. **All tests should be independent**
13. **Setup detailed automation tests reporting**

Lets look into framework structure and see how it aligns with best practices outlined above:



Helper :
In this directory we will have centralized the functions we will using in the tests to perform steps, this way it keeps code clean and easy to maintain. If need to change won't have to change in all tests in corresponding test file infact just change here. Next you will observe no hard-coded values used in any of helper functions all test data is centralized as well in ApplicationProperties.robot. We have CommonHelper.robot which holds common keywords which will be used repeatedly to achieve reusability

PageObjects:
In this directory we will store objects relevant to pages to achieve reusability in Helper classes, easy to maintain lets say locator of any element is changed by developer you wont have to change every in helper classes in fact just change from here last but least it keeps the code clean . Ensure to use id, css, relative xpath, in some cases id's might not be available try css and xpath (relative). We have CommonObjects.robot to hold objects tied to common keywords and will be used repeatedly .

TestCases:
In this directory we will write all test cases here . In each test case code is written using clean code practices like no hard-coded values and clear to read. Testcases are designed to handle to capture screenshot if any test fails.

TestData:
In this directory we have created ApplicationProperties.robot to store all test data centralized , easy to maintain if changes we can change here and don't have to change every in tests.

Utility:
In this directory we have Setup.robot which is responsible to start test for example opening browsers, navigating to website or closing browser to taking screenshot. Instead of writing all that code in every class we have made it code cleaner and abstracted and directly used in test cases. The other file here we have SeleniumKeywords.robot which is core of performing common re-usable SeleniumLibrary functions. We have centralized here this will easy to use and access, re-usbaility and keep code clean in Helper classes.

Data-Drive Testing – Excel

In order to read the test data from excel we would to install robotframework-datadriver . You can find more information here : https://pypi.org/project/robotframework-datadriver/

pip install --upgrade robotframework-datadriver

```
C:\Users\user>pip install --upgrade robotframework-datadriver
Collecting robotframework-datadriver
  Using cached robotframework_datadriver-0.3.6-py3-none-any.whl (35 kB)
Collecting Pygments
  Using cached Pygments-2.6.1-py3-none-any.whl (914 kB)
Requirement already satisfied, skipping upgrade: robotframework>=3.1 in c:\users\user\appdata\local\programs\python\pyth
on38-32\lib\site-packages (from robotframework-datadriver) (3.2.1)
Collecting docutils
  Using cached docutils-0.16-py2.py3-none-any.whl (548 kB)
Installing collected packages: Pygments, docutils, robotframework-datadriver
Successfully installed Pygments-2.6.1 docutils-0.16 robotframework-datadriver-0.3.6
```

Then run this command

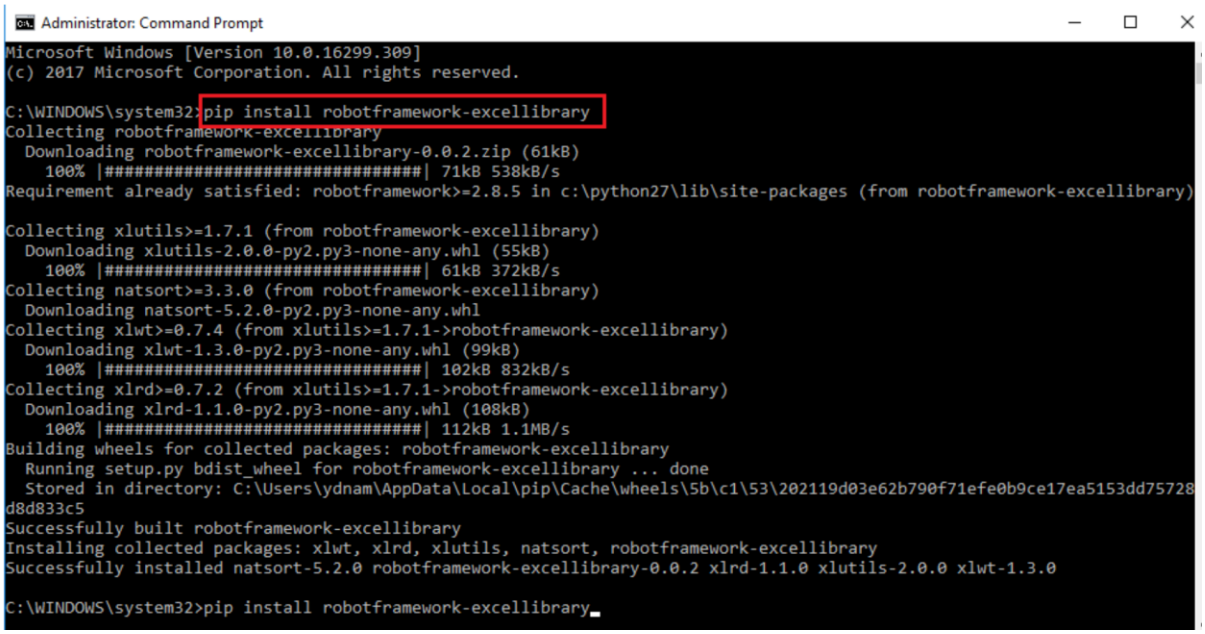
pip install --upgrade robotframework-datadriver[XLS]

```
C:\Users\user>pip install --upgrade robotframework-datadriver[XLS]
Requirement already up-to-date: robotframework-datadriver[XLS] in c:\users\user\appdata\local\programs\python\python38-3
2\lib\site-packages (0.3.6)
Requirement already satisfied, skipping upgrade: robotframework>=3.1 in c:\users\user\appdata\local\programs\python\pyth
on38-32\lib\site-packages (from robotframework-datadriver[XLS]) (3.2.1)
Requirement already satisfied, skipping upgrade: Pygments in c:\users\user\appdata\local\programs\python\python38-32\lib
\site-packages (from robotframework-datadriver[XLS]) (2.6.1)
Requirement already satisfied, skipping upgrade: docutils in c:\users\user\appdata\local\programs\python\python38-32\lib
\site-packages (from robotframework-datadriver[XLS]) (0.16)
Requirement already satisfied, skipping upgrade: xlrd>=1.0.0; extra == "xls" in c:\users\user\appdata\local\programs\pyt
hon\python38-32\lib\site-packages (from robotframework-datadriver[XLS]) (1.2.0)
Requirement already satisfied, skipping upgrade: pandas; extra == "xls" in c:\users\user\appdata\local\programs\python\p
ython38-32\lib\site-packages (from robotframework-datadriver[XLS]) (1.0.1)
Requirement already satisfied, skipping upgrade: numpy; extra == "xls" in c:\users\user\appdata\local\programs\python\py
thon38-32\lib\site-packages (from robotframework-datadriver[XLS]) (1.18.1)
Requirement already satisfied, skipping upgrade: python-dateutil>=2.6.1 in c:\users\user\appdata\local\programs\python\p
ython38-32\lib\site-packages (from pandas; extra == "xls"->robotframework-datadriver[XLS]) (2.8.1)
Requirement already satisfied, skipping upgrade: pytz>=2017.2 in c:\users\user\appdata\local\programs\python\python38-32
\lib\site-packages (from pandas; extra == "xls"->robotframework-datadriver[XLS]) (2019.3)
Requirement already satisfied, skipping upgrade: six>=1.5 in c:\users\user\appdata\roaming\python\python38\site-packages
 (from python-dateutil>=2.6.1->pandas; extra == "xls"->robotframework-datadriver[XLS]) (1.14.0)
```

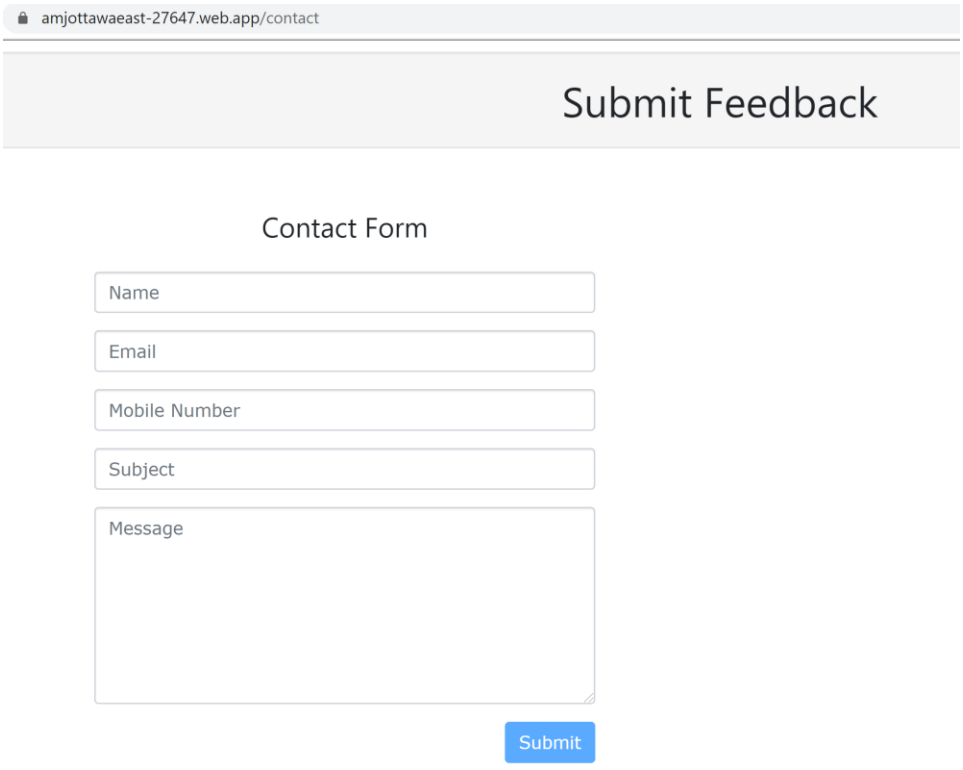
DataDriver is a Data-Driven Testing library for Robot Framework. DataDriver is used/imported as Library but does not provide keywords which can be used in a test. DataDriver uses the Listener Interface Version 3 to manipulate the test cases and creates new test cases based on a Data-File that contains the data for Data-Driven Testing. These data file may be .csv , .xls or .xlsx files.

and also would need to install excel library

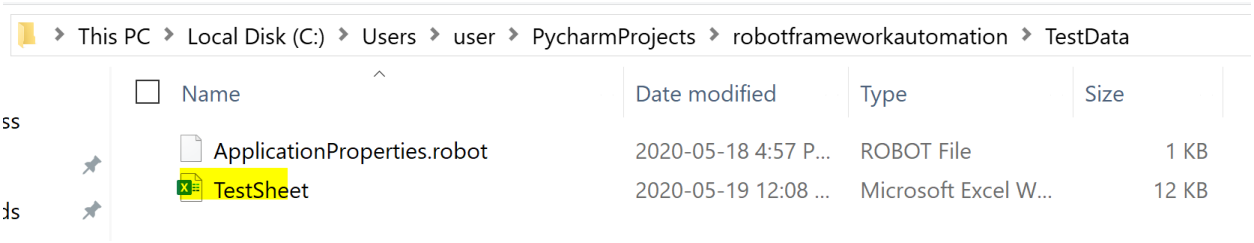
```
pip install robotframework-excellibrary
```



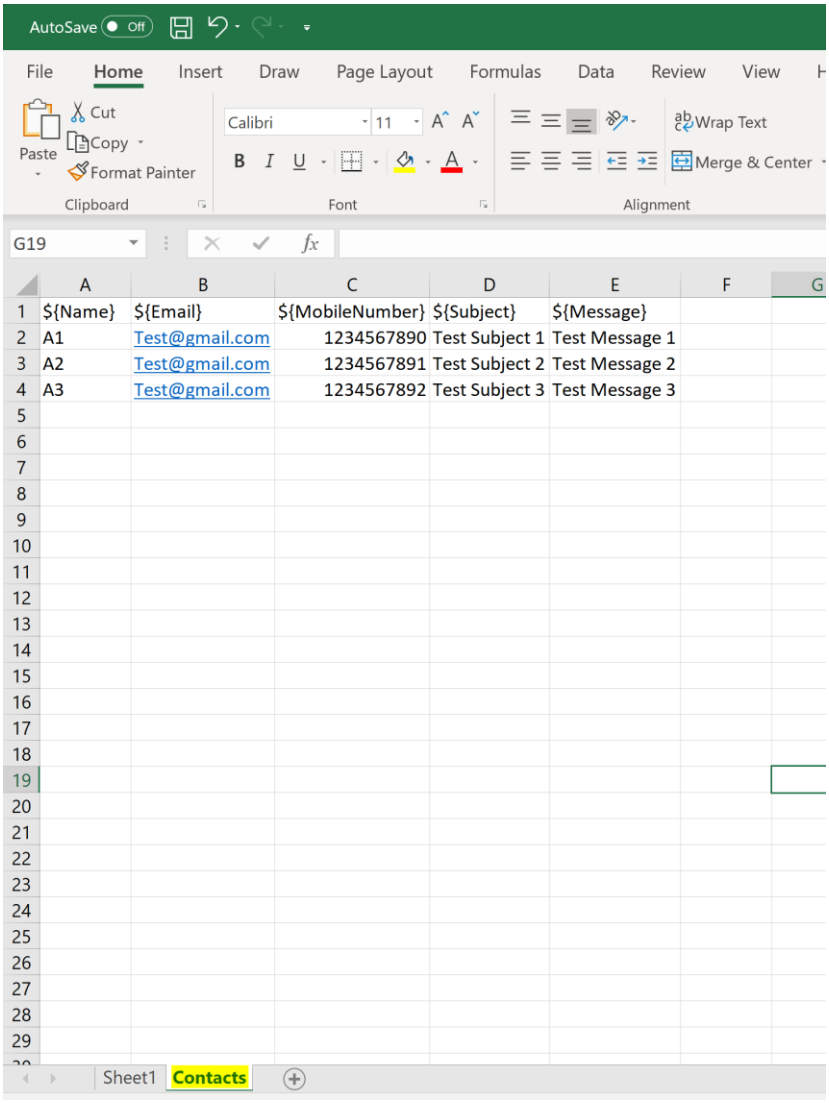
Lets get started . What we will do is we will the online form by reading test data from excel.



Lets setup our TestSheet.xlsx and we will place this under TestData folder

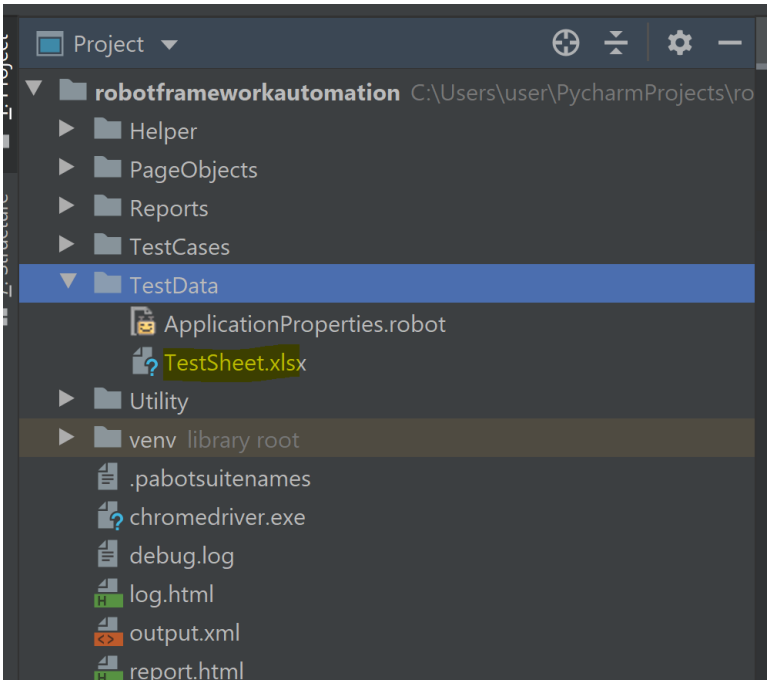


Now lets fill the test data in it . I created Contacts sheet . You can see headers of column will be always in ${name_of_column} , this how it recognize this row and its column name .

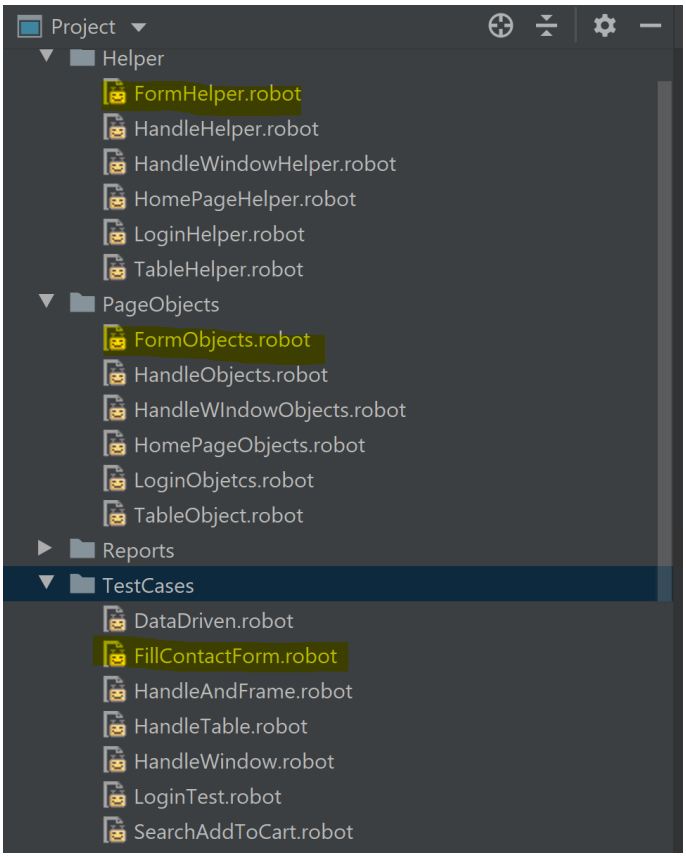


Save the file.

You should see file under TestData folder

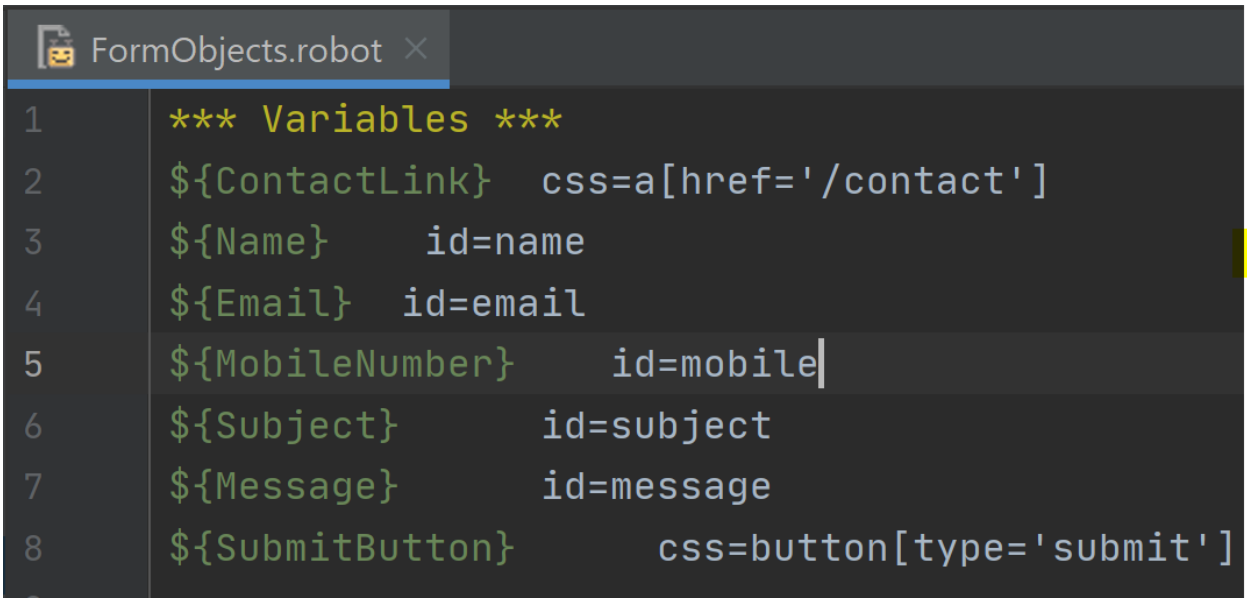


Lets create the following



FillContactForm.robot under TestCases folder in this we will write test steps and data driver to read data from excel, FormObjects.robot this where we will store the locators of the form elements, and FormHelper.robot this where we will implement the test steps.

Lets add all locators of UI elements we will be interacting with in FormObjects.robot



Code: FormObjects.robot

```
*** Variables ***
${ContactLink}  css=a[href='/contact']
${Name}  id=name
${Email}  id=email
${MobileNumber}  id=mobile
${Subject}  id=subject
${Message}  id=message
${SubmitButton}  css=button[type='submit']
```

Now lets look into FillContactForm.robot that's where we will write test steps.

This time we will add import for ExcelLibrary , DataDriver and Test Template .

FillContactForm.robot

```
*** Settings ***
Resource    ../Utility/Setup.robot
Resource    ../Helper/FormHelper.robot
Library     ExcelLibrary
Library     DataDriver  file=${EXECDIR}/TestData/TestSheet.xlsx     sheet_name=Contacts
Resource    ../TestData/ApplicationProperties.robot
Suite Setup  Open the browser in
Test Setup  Go back to homepage
Test Teardown  Capture screenshot if test failed   Dummy
Suite Teardown  Close All Browser Window
Test Template   Fill Form by given data
```

You can see we imported ExcelLibrary . Next we imported library DataDriver in which we specify the file location using ${EXECDIR}, an absolute path to the directory where test execution was started from, then we sheet name since our test data is in Contacts sheet so we have reffered it here. Note in any give test case we can refer only 1 sheet not multiple , thus if you have multiple tests which needs data from different sheets create separate test files and refer them respectively.

Last but not least we have used the Test Template. Test templates convert normal keyword-driven test cases into data-driven tests. Whereas the body of a keyword-driven test case is constructed from keywords and their possible arguments, test cases with template contain only the arguments for the template keyword. Instead of repeating the same keyword multiple times per test and/or with all tests in a file, it is possible to use it only per test or just once per file.

Template keywords can accept both normal positional and named arguments, as well as arguments embedded to the keyword name. Unlike with other settings, it is not possible to define a template using a variable.

Now let see how keyword is converted into data driven

```
Test Template   Fill Form by given data

*** Test Cases ***
Fill contact form for ${Name}

*** Keywords ***
Fill Form by given data
    [Arguments]     ${Name}     ${Email}    ${MobileNumber}     ${Subject}  ${Message}
    Navigate to the site    ${ContactFormURL}
    Click on Contact link
    Enter name  ${Name}
    Enter email    ${Email}
    Enter MobileNumber    ${MobileNumber}
    Enter Subject   ${Subject}
    Enter Message  ${Message}
    Click on submit button
```

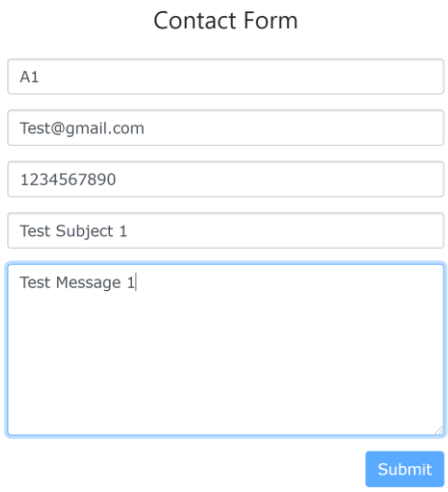
You can see under Test Cases section we are just putting the name of test case along with column name ${Name} from column in excel .

Under keywords section we have referred the Test Template Fill Form by given data and right in next line we provided the [Arguments] which are column headers from excel . Please ensure they are same as in excel.

So it reads row by row from excel sheet so you can see it will enter data from first row from excel

${Name}	${Email}	${MobileNumber}	${Subject}	${Message}
			Test Subject	Test Message
A1	Test@gmail.com	1234567890	1	1

Into the contact form



As you see these steps

```
Enter name  ${Name}
Enter email    ${Email}
Enter MobileNumber    ${MobileNumber}
Enter Subject   ${Subject}
Enter Message  ${Message}
Click on submit button
```

Now in each of step above we pass the value from each column and pass as Argument except the last step it clicks on submit button . After that it goes back 1st step and but this time it will iterate on data in 2nd row this will continue until all rows are completed .

Code: FillContactForm.robot

```
*** Settings ***
Resource    ../Utility/Setup.robot
Resource    ../Helper/FormHelper.robot
Library     ExcelLibrary
Library     DataDriver  file=${EXECDIR}/TestData/TestSheet.xlsx    sheet_name=Contacts
Resource    ../TestData/ApplicationProperties.robot
Suite Setup  Open the browser in
Test Setup  Go back to homepage
Test Teardown  Capture screenshot if test failed   Dummy
Suite Teardown  Close All Browser Window
Test Template   Fill Form by given data

*** Test Cases ***
Fill contact form for ${Name}

*** Keywords ***
Fill Form by given data
   [Arguments]    ${Name}    ${Email}   ${MobileNumber}    ${Subject}  ${Message}
   Navigate to the site    ${ContactFormURL}
   Click on Contact link
   Enter name  ${Name}
   Enter email    ${Email}
   Enter MobileNumber    ${MobileNumber}
   Enter Subject   ${Subject}
   Enter Message  ${Message}
   Click on submit button
```

Now let see the implementation in FormHelper.robot .

```
*** Settings ***
Resource    ../PageObjects/FormObjects.robot
Resource    ../Utility/SeleniumKeywords.robot
Library     BuiltIn

*** Keywords ***
Navigate to the site
    [Arguments]    ${URL}
    Navigate to the page    ${URL}

Click on Contact link
    Click On Page Element    ${ContactLink}

Enter name
    [Arguments]    ${text}
    Set Value For Input Field    ${Name}    ${text}

Enter email
    [Arguments]    ${text}
    Set Value For Input Field    ${Email}    ${text}

Enter MobileNumber
    [Arguments]    ${text}
    Set Value For Input Field    ${MobileNumber}    ${text}

Enter Subject
    [Arguments]    ${text}
    Set Value For Input Field    ${Subject}    ${text}

Enter Message
    [Arguments]    ${text}
    Set Value For Input Field    ${Message}    ${text}

Click on submit button
    Click On Button    ${SubmitButton}
```

Now you can see these all steps are implemented using functions in SeleniumKeywords.robot but all of them are discussed previously so I wont go into details.

You will need to add the url for this site in ApplicationProperties.robot

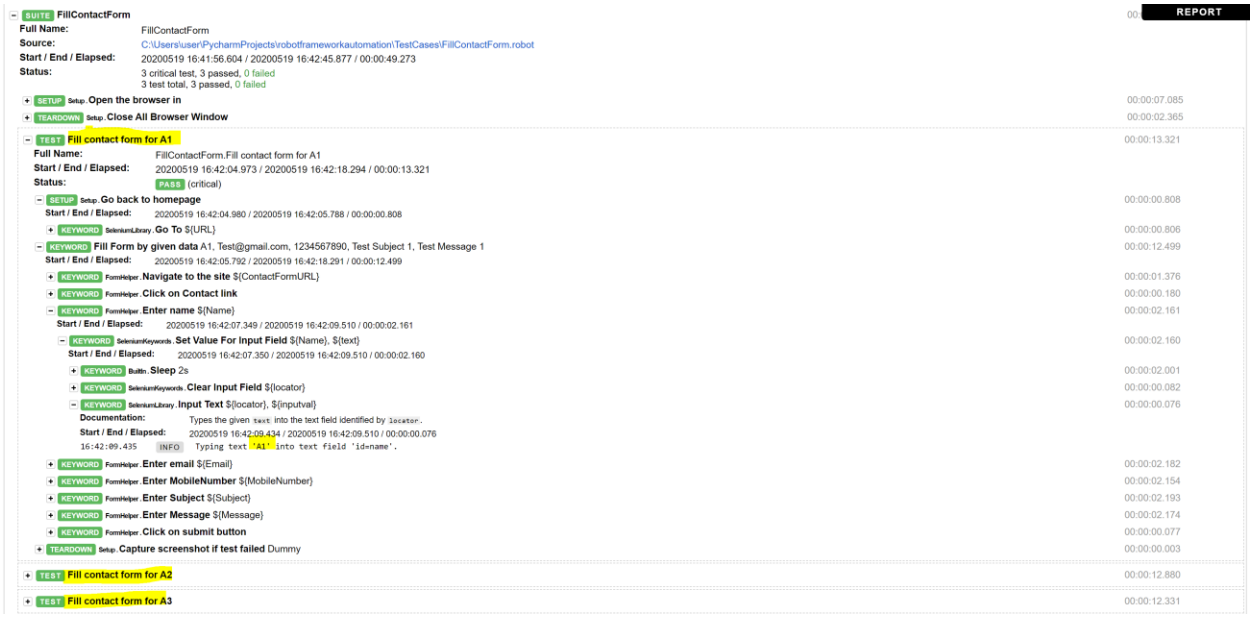
```
*** Variables ***
${URL}    https://demo.nopcommerce.com/
${UserName}    waheedahmed55@gmail.com
${InvalidPassword}    test1
${Password}    test123
${SearchItem}    Laptop
${AlertPageUrl}    http://demo.automationtesting.in/Alerts.html
${FrameUrl}    http://demo.automationtesting.in/Frames.html
${TableUrl}    https://info.sice.indiana.edu/~hrosenba/Demo/Demo4.html
${ContactFormURL}    https://amjottawaeast-27647.web.app
${WindowURL}    http://demo.automationtesting.in/Windows.html
```

Now we will run

robot -d "Reports" TestCases/FillContactForm.robot

```
Terminal:  Local ×  Local (2) ×  +
(venv) C:\Users\user\PycharmProjects\robotframeworkautomation>robot -d "Reports" TestCases/FillContactForm.robot
==============================================================================
FillContactForm
==============================================================================

DevTools listening on ws://127.0.0.1:60028/devtools/browser/9b077ec4-3fa7-45b3-9aa1-4927247a0d59
[21076:18060:0519/164158.769:ERROR:browser_switcher_service.cc(238)] XXX Init()
Fill contact form for A1                                              | PASS |
------------------------------------------------------------------------------
Fill contact form for A2                                              | PASS |
------------------------------------------------------------------------------
Fill contact form for A3                                              | PASS |
------------------------------------------------------------------------------
FillContactForm                                                       | PASS |
3 critical tests, 3 passed, 0 failed
3 tests total, 3 passed, 0 failed
==============================================================================
Output:  C:\Users\user\PycharmProjects\robotframeworkautomation\Reports\output.xml
Log:     C:\Users\user\PycharmProjects\robotframeworkautomation\Reports\log.html
Report:  C:\Users\user\PycharmProjects\robotframeworkautomation\Reports\report.html
```

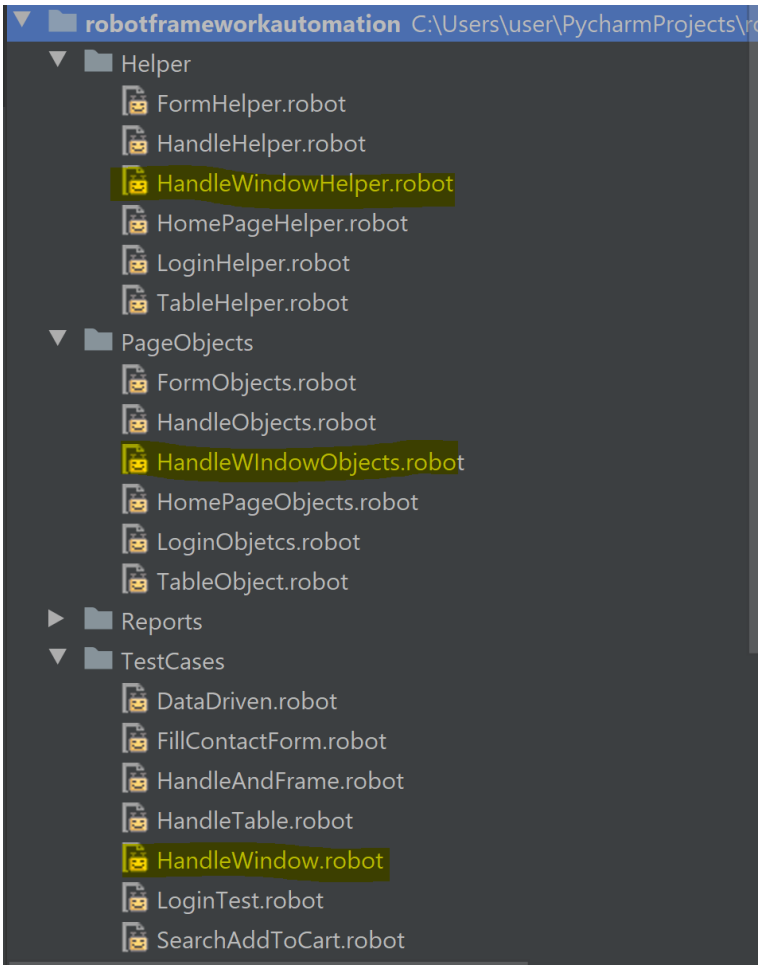
You can see in test report as well



Handle Tab Windows and New Browser Windows

In this section we will discuss how we will handle new tabs opened and we can switch to newly opened tab and switch back and same for new browser windows.

Lets create the following files:



For this section I will be using this site http://demo.automationtesting.in/Windows.html . Add this to ApplicationProperties.robot

Code: ApplicationProperties.robot

```
*** Variables ***
${URL}    https://demo.nopcommerce.com/
${UserName}    waheedahmed55@gmail.com
${InvalidPassword}    test1
${Password}    test123
${SearchItem}    Laptop
${AlertPageUrl}    http://demo.automationtesting.in/Alerts.html
${FrameUrl}    http://demo.automationtesting.in/Frames.html
${TableUrl}    https://info.sice.indiana.edu/~hrosenba/Demo/Demo4.html
${ContactFormURL}    https://amjottawaeast-27647.web.app
${WindowURL}    http://demo.automationtesting.in/Windows.html
```

Now let’s add the locators in HandleWindowObjects.robot

Code: HandleWindowObjects.robot

```
*** Variables ***
${TabWindowButton}    css=a[href='http://www.sakinalium.in']
${SaprateWindowLink}  css=a[href='#Seperate']
${MultipleWindowLink}  css=a[href='#Multiple']
${NewWindowButton}    css=button[onclick='newwindow()']
${MultipleWindowButton}    css=button[onclick='multiwindow()']
```

Now lets see our test steps we will be performing in HandleWindow.robot

Code: HandleWindow.robot

```
*** Settings ***
Resource    ../Utility/Setup.robot
Resource    ../Helper/HandleWindowHelper.robot
Resource    ../TestData/ApplicationProperties.robot
Suite Setup  Open the browser in
Test Setup  Go back to homepage
Test Teardown  Capture screenshot if test failed   Dummy
Suite Teardown  Close All Browser Window

*** Test Cases ***
Manage Tabbed window
   Navigate to the site    ${WindowURL}
   Print title of window
   Click on tab button
   Switch to new window    NEW
   Print title of window
   Switch to new window    MAIN
   Print title of window

Manage Browser
   Navigate to the site    ${WindowURL}
   Click on seprate tab
   Print title of window
   Click on new window button
   Switch to new window    NEW
   Print title of window
   Switch to new window    MAIN
   Print title of window

Manage Multiple window
   Navigate to the site    ${WindowURL}
   Click on multiple tab
   Print title of window
   Click on multiple window button
   Switch to new window    NEW
   Take screenshot    NewWindow
   Print title of window
   Switch to new window    MAIN
   Print title of window
```

These are very simple and straightforward test cases with easy test steps. Lets look at our first test case Manage Tabbed window

```
*** Test Cases ***
Manage Tabbed window
    Navigate to the site    ${WindowURL}
    Print title of window
    Click on tab button
    Switch to new window    NEW
    Print title of window
    Switch to new window    MAIN
    Print title of window
```

First as usual we are navigating to site http://demo.automationtesting.in/Windows.html . Next we are Printing the title of window . lets look at its implementation in HandleWindowHelper.robot

```
Print title of window
    ${val}=  Title of the window
    log to console  ${val}
```

Here you can see we are calling Title of the window function from SeleniumKeywords.robot and that function returns a value which we are saving in ${val} .

Lets dive into SeleniumKeywords.robot and see implementation of this function

```
Title of the window
    ${val}=    Get Title
    [return]   ${val}
```

Here you can you see we are using SeleniumLibrary keyword Get Title which returns title of page and we are saving in variable ${val} and in ver next line we are returning that value . Which in our HandleWindowHelper.robot file we are saving in ${val}

Get Title: https://robotframework.org/SeleniumLibrary/SeleniumLibrary.html#Get%20Title

Get Title		Returns the title of the current page.

This is how we can create functions to return and save values in variables.

Now lets go back to HandleWindowHelper.robot

```
Print title of window
    ${val}=  Title of the window
    log to console  ${val}
```

After saving value which is title of the window we are printing it on console using log to console built in function.

Next step is just clicking on the button, its simple. Lets move to main step which is Switch to new window NEW , lets look into its implementation HandleWindowHelper.robot

```
Switch to new window
    [Arguments]    ${type}
    Switch to window    ${type}
```

Here you can see this function takes arugment which is passed from test step , in our case it was NEW

We will pass this to Swtich to window function which we have implemented in SeleniumKeywords.robot

Lets look into implementation of it :

```
Switch to window
    [Arguments]    ${type}
    Switch Window  ${type}
```

Here you can see we are using Switch Window SeleniumLibrary keywor and passing parameter to it which in our test case is NEW . This keyword Swtich Window switches to browser window matching locator which in our case is NEW. There are three pre-defined locators this keyword accepts , NEW, MAIN and CURRENT. This because if we have to switch back to where we started initial test than we can pass MAIN as locator to this keyword and it will switch back to it.

Switch Window:
https://robotframework.org/SeleniumLibrary/SeleniumLibrary.html#Switch%20Window

Switch Window	*locator=MAIN, timeout: NoneType=None, browser=CURRENT*	Switches to browser window matching `locator`. If the window is found, all subsequent commands use the selected window, until this keyword is used again. If the window is not found, this keyword fails. The previous windows handle is returned and can be used to switch back to it later. Notice that alerts should be handled with *Handle Alert* or other alert related keywords. The `locator` can be specified using different strategies somewhat similarly as when *locating elements* on pages. ▪ By default, the `locator` is matched against window handle, name, title, and URL. Matching is done in that order and the first matching window is selected. ▪ The `locator` can specify an explicit strategy by using the format `strategy:value` (recommended) or `strategy=value`. Supported strategies are `name`, `title`, and `url`. These matches windows using their name, title, or URL, respectively. Additionally, `default` can be used to explicitly use the default strategy explained above. ▪ If the `locator` is `NEW` (case-insensitive), the latest opened window is selected. It is an error if this is the same as the current window. ▪ If the `locator` is `MAIN` (default, case-insensitive), the main window is selected. ▪ If the `locator` is `CURRENT` (case-insensitive), nothing is done. This effectively just returns the current window handle. ▪ If the `locator` is not a string, it is expected to be a list of window handles *to exclude*. Such a list of excluded windows can be got from *Get Window Handles* before doing an action that opens a new window.

Next step you see I am printing title this time it will print title of new window just to ensure we have switched to it. Similarly in following step I am calling same function Switch to new window MAIN but

using MAIN as locator for Switch Window keyword as explained above. Once it switches back I print the title of page once again to be sure I am on correct main page.

The other two test cases are similar nothing new implemented nor used any new SeleniumLibrary keyword.

Code: HandleWindowHelper.robot

```
*** Settings ***
Resource    ../PageObjects/HandleWIndowObjects.robot
Resource    ../Utility/SeleniumKeywords.robot
Library     BuiltIn

*** Keywords ***
Navigate to the site
    [Arguments]     ${URL}
    Navigate to the page    ${URL}

Click on tab button
    Click On Page Element   ${TabWindowButton}

Switch to new window
    [Arguments]     ${type}
    Switch to window    ${type}

Print title of window
    ${val}=  Title of the window
    log to console  ${val}

Click on seprate tab
    Click On Page Element   ${SaprateWindowLink}

Click on new window button
    Click On Button     ${NewWindowButton}

Click on multiple tab
    Click On Page Element   ${MultipleWindowLink}

Click on multiple window button
      Click On Button     ${MultipleWindowButton}
```

Save all files and run the tests.

robot -d "Reports" TestCases/HandleWindow.robot

```
(venv) C:\Users\user\PycharmProjects\robotframeworkautomation>robot -d "Reports" TestCases/HandleWindow.robot
```

Output on console: You can see it printed the title of the page

```
DevTools listening on ws://127.0.0.1:51112/devtools/browser/dae4c1c3-3612-49d5-bc57-c06103502563
[16412:25140:0520/090008.612:ERROR:browser_switcher_service.cc(238)] XXX Init()
Manage Tabbed window                                                    ..Frames & windows
...Sakinalium | Home
..Frames & windows
Manage Tabbed window                                                    | PASS |
------------------------------------------------------------------------------
Manage Browser                                                          ...Frames & windows
...Sakinalium | Home
..Frames & windows
Manage Browser                                                          | PASS |
------------------------------------------------------------------------------
Manage Multiple window                                                  ...Frames & windows
....Index
Manage Multiple window                                                  .Frames & windows
Manage Multiple window                                                  | PASS |
------------------------------------------------------------------------------
```

In Manage Multiple window I added step to Take Screenshot once its on new window lets examine it from test report.

HandleWindow Log

Generated
20200520 09:00:41 UTC-04:00
2 minutes 39 seconds ago

Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	3	3	0	00:00:25	
All Tests	3	3	0	00:00:25	

Statistics by Tag	Total	Pass	Fail	Elapsed	Pass / Fail
No Tags					

Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
HandleWindow	3	3	0	00:00:34	

Test Execution Log

SUITE **HandleWindow**

Full Name:	HandleWindow
Source:	C:\Users\user\PycharmProjects\robotframeworkautomation\TestCases\HandleWindow.robot
Start / End / Elapsed:	20200520 09:00:07.272 / 20200520 09:00:41.552 / 00:00:34.280
Status:	3 critical test, 3 passed, 0 failed 3 test total, 3 passed, 0 failed

SETUP Setup. **Open the browser in**

TEARDOWN Setup. **Close All Browser Window**

TEST **Manage Tabbed window**

TEST **Manage Browser**

TEST **Manage Multiple window**

If we expand Manage Multiple window test case as it switched it took screenchot

TEST **Manage Multiple window**

Full Name: HandleWindow.Manage Multiple window

Start / End / Elapsed: 20200520 09:00:30.899 / 20200520 09:00:38.687 / 00:00:07.788

Status: PASS (critical)

SETUP Setup. **Go back to homepage**

KEYWORD HandleWindowHelper. **Navigate to the site** ${WindowURL}

KEYWORD HandleWindowHelper. **Click on multiple tab**

KEYWORD HandleWindowHelper. **Print title of window**

KEYWORD HandleWindowHelper. **Click on multiple window button**

KEYWORD HandleWindowHelper. **Switch to new window** NEW

KEYWORD Setup. **Take screenshot** NewWindow

Start / End / Elapsed: 20200520 09:00:36.454 / 20200520 09:00:38.423 / 00:00:01.969

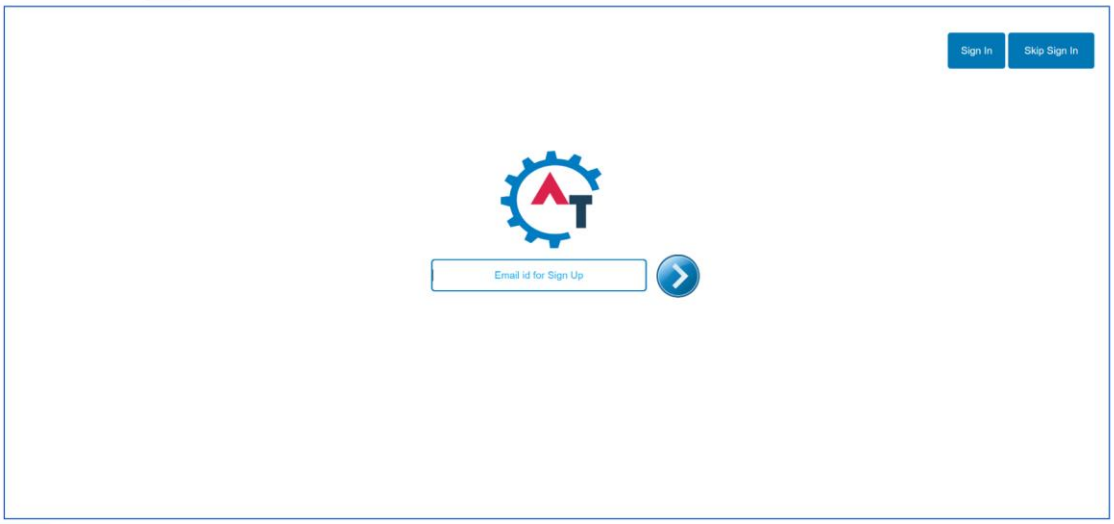
KEYWORD ${now} = BuiltIn. **Evaluate** '{dt.second}{dt.minute}{dt.hour}{dt.day}{dt.month}{dt.year}'.format(dt=datetime.datetime.now()), modules=datetime

KEYWORD SeleniumLibrary. **Capture Page Screenshot** ${OUTPUTDIR}/ScreenShot/${FileName}_${now}.png

Documentation: Takes a screenshot of the current page and embeds it into a log file.

Start / End / Elapsed: 20200520 09:00:36.456 / 20200520 09:00:38.423 / 00:00:01.967

09:00:38.423 INFO



KEYWORD HandleWindowHelper. **Print title of window**

KEYWORD HandleWindowHelper. **Switch to new window** MAIN

KEYWORD HandleWindowHelper. **Print title of window**

TEARDOWN Setup. **Capture screenshot if test failed** Dummy

Code: SeleniumKeywords.robot

```
*** Settings ***
Library    SeleniumLibrary

*** Keywords ***
Clear Input Field
   [Arguments]   ${locator}
   Wait For Element Present   ${locator}  5   Element not found in 5 seconds
   Clear Element Text   ${locator}
   ${value}=    Get Element Attribute   ${locator}    value
   ${backspaces count}=   Get Length    ${value}
   Run Keyword If   """${value}""" != ''
   ...    Repeat Keyword  ${backspaces count +1}  Press Key  ${locator}   \\08

Set Value For Input Field
   [Arguments]   ${locator}   ${inputval}
   sleep   2s
   Clear Input Field   ${locator}
   Input Text   ${locator}   ${inputval}

Click On Page Element
   [Arguments]   ${locator}
   Wait For Element Present   ${locator}  5   Element not found in 5 seconds
   Click Element   ${locator}

Click On Button
   [Arguments]   ${locator}
   Wait For Element Present   ${locator}  5   Element not found in 5 seconds
   Click Button   ${locator}

Wait For Page Load
   [Arguments]   ${pgloadlocator}
   Wait Until Page Contains Element   ${pgloadlocator}
   Sleep   5s

Wait For Element Present
   [Arguments]   ${locator}   ${timeout}   ${err_msg}
   #Wait For Element Present   ${locator}  5   Element not found in 5 seconds
   Wait Until Page Contains Element   ${locator}   ${timeout}   ${err_msg}

Get Attribute Value
   [Arguments]   ${locator}   ${attribute}
   #Wait For Element Present   ${locator}  5   Element not found in 5 seconds
   ${string}=   Get Element Attribute   ${locator}   ${attribute}
   log to console   \n get attribute ${string}.\n
   [Return]   ${string}

Back On Screen
   Go Back

Wait For Text Present
   [Arguments]   ${text}   ${seconds}   ${message}
   Wait Until Page Contains   ${text}   ${seconds}   ${message}
```

```
Verify Text Present
    [Arguments]    ${text}
    Page Should Contain    ${text}

Verify Element Present
    [Arguments]    ${locator}
    #Wait For Element Present    ${locator}  5    Element not found in 5 seconds
    Page Should Contain Element    ${locator}

Generate Random String With Defined Size
    [Arguments]    ${size}    ${type}
    ${str}=    Generate Random String    ${size}    ${type}
    [Return]    ${str}

Generate Random Number With Defined Size
    [Arguments]    ${size}    ${type}
    ${num}=    Generate Random String    ${size}    ${type}
    [Return]    ${num}

Match Value
    [Arguments]    ${locator}    ${attr_name}    ${match_pattern}
    #Wait For Element Present    ${locator}  5    Element not found in 5 seconds
    Element Attribute Should Match    ${locator}    ${attr_name}    ${match_pattern}

Verify Text Not Present
    [Arguments]    ${text}
    Page Should Not Contain    ${text}

Select List Option By visible Text
    [Arguments]    ${locator}    ${value}
    Wait For Element Present    ${locator}  5    Element not found in 5 seconds
    Select From List By Label    ${locator}    ${value}

Verify Page title
    [Arguments]    ${title}
    Title Should Be    ${title}

Navigate to the page
    [Arguments]    ${URL}
    Go To    ${URL}

Verify Element Displayed
    [Arguments]    ${locator}
    Wait For Element Present    ${locator}  5    Element not found in 5 seconds
    Element Should Be Visible    ${locator}

Verify Element not displayed
    [Arguments]    ${locator}
    Element Should Not Be Visible    ${locator}

Verify Current URL should contains
    [Arguments]    ${URL}
    Location Should Contain    ${URL}

Static wait for
```

```
    [Arguments]    ${Second}
    Sleep    ${Second}s

Get all element count
    [Arguments]    ${locator}
    ${num}=    Get Element Count    ${locator}
    [Return]    ${num}

Verify both values are equal
    [Arguments]    ${val1}  ${val2}
    Should Be Equal    ${val1}    ${val2}

Switch to window
    [Arguments]    ${type}
    Switch Window  ${type}

Close current window
    Close Window

Get Current URL
    return from keyword    Get Location

Press keyboard key
    [Arguments]    ${locator}   ${Key}
    Wait For Element Present    ${locator}  5    Element not found in 5 seconds
    Press Key  ${locator}   \\08

Mouse Over on the element
    [Arguments]    ${locator}
    Wait For Element Present    ${locator}  5    Element not found in 5 seconds
    Mouse Over  ${locator}

URL Should Contain
    [Arguments]    ${expected}
    Location Should Contain    ${expected}

Scroll for element
    [Arguments]    ${locator}
    Scroll Element Into View    ${locator}

Scroll at the bottom of the page
    Execute Javascript    window.scrollTo(0,document.body.scrollHeight)

Scroll at the top of the page
    Execute Javascript    window.scrollTo(0,0)

Handle the alert box
    [Arguments]    ${action}
    Handle Alert    ${action}

Enter text in alert box
    [Arguments]    ${text}
    Input text into Alert   ${text}

Select a Frame
```

```
    [Arguments]  ${locator}
    Select Frame    ${locator}

UnSelect a Frame
    Unselect Frame

Get Text From element
    [Arguments]  ${locator}
    ${val}=    Get Text    ${locator}
    [return]    ${val}

Title of the window
    ${val}=    Get Title
    [return]    ${val}
```

Parallel Test Execution:

In this section we will see how to execute the test in parallel. In order to do so we would need pabot library . A parallel executor for Robot Framework tests. With Pabot you can split one execution into multiple and save test execution time.

Lets install pabot : run the command pip install robotframework-pabot

```
(venv) C:\Users\user\PycharmProjects\robotframeworkautomation>pip install robotframework-pabot
```

```
Terminal:  Local ×  Local (2) ×  +
  Downloading robotframework-pabot-1.7.0.tar.gz (30 kB)
Requirement already satisfied: robotframework in c:\users\user\pycharmprojects\robotframeworkautomation\venv\lib\site-packages (from robotframework-pabot) (3.2.1)
Collecting robotremoteserver>=1.1
  Downloading robotremoteserver-1.1-py2.py3-none-any.whl (15 kB)
Could not build wheels for robotframework-pabot, since package 'wheel' is not installed.
Could not build wheels for robotframework, since package 'wheel' is not installed.
Installing collected packages: robotremoteserver, robotframework-pabot
    Running setup.py install for robotframework-pabot ... done
Successfully installed robotframework-pabot-1.7.0 robotremoteserver-1.1

(venv) C:\Users\user\PycharmProjects\robotframeworkautomation>
```

In order to run the test cases in test file in parallel mode we will use pabot command with defined number of process we want to run. For this example, I will run multiple test files in parallel mode by using following command

pabot --processes 3 --outputdir "Reports" TestCases/*.robot

here --processess 3 is count of the window which will be open during execution.

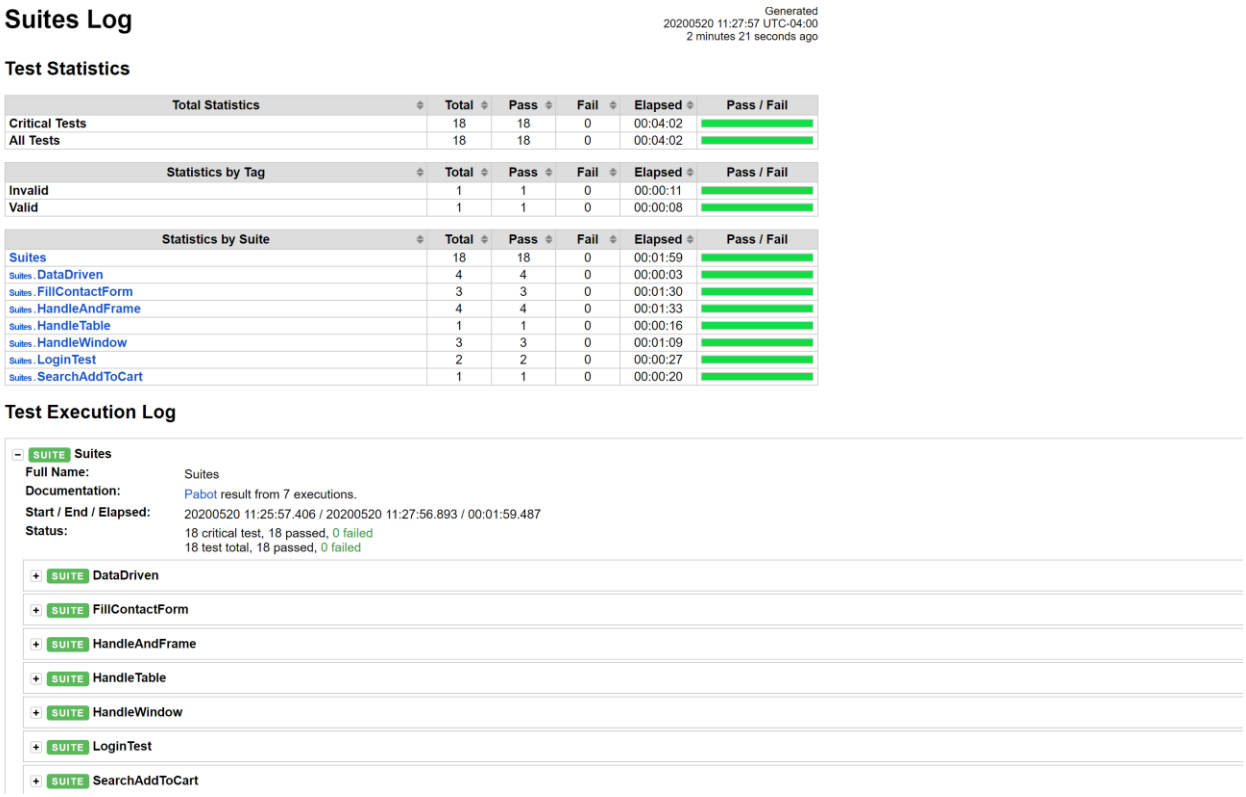
```
(venv) C:\Users\user\PycharmProjects\robotframeworkautomation>pabot --processes 3 --outputdir "Reports" TestCases/*.robot
```

You will see three chrome browsers

Console output:

```
Terminal:  Local ×  Local (2) ×  +
[23112:25184:0520/112730.593:ERROR:browser_switcher_service.cc(238)] XXX Init()
2020-05-20 11:27:31.669802 [PID:22960] [2] [ID:2] PASSED Suites.HandleAndFrame in 75.1 seconds

DevTools listening on ws://127.0.0.1:65227/devtools/browser/7ca42ea2-b42d-44e0-a071-be8bb9e849ed
[21644:24864:0520/112731.781:ERROR:browser_switcher_service.cc(238)] XXX Init()
2020-05-20 11:27:44.785493 [PID:21828] [1] [ID:5] still running Suites.LoginTest after 15.0 seconds
2020-05-20 11:27:45.088500 [PID:2420] [0] [ID:6] still running Suites.SearchAddToCart after 15.0 seconds
2020-05-20 11:27:50.014825 [PID:2420] [0] [ID:6] PASSED Suites.SearchAddToCart in 19.6 seconds
2020-05-20 11:27:56.669436 [PID:21828] [1] [ID:5] PASSED Suites.LoginTest in 26.2 seconds
18 critical tests, 18 passed, 0 failed
18 tests total, 18 passed, 0 failed
==================================================
Output:  C:\Users\user\PycharmProjects\robotframeworkautomation\Reports\output.xml
Log:     C:\Users\user\PycharmProjects\robotframeworkautomation\Reports\log.html
Report:  C:\Users\user\PycharmProjects\robotframeworkautomation\Reports\report.html
Total testing: 4 minutes 26.50 seconds
Elapsed time:  2 minutes 0.21 seconds
```

You can see from test report all test suites passed



Suites Log

Generated
20200520 11:27:57 UTC-04:00
2 minutes 21 seconds ago

Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	18	18	0	00:04:02	
All Tests	18	18	0	00:04:02	

Statistics by Tag	Total	Pass	Fail	Elapsed	Pass / Fail
Invalid	1	1	0	00:00:11	
Valid	1	1	0	00:00:08	

Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
Suites	18	18	0	00:01:59	
Suites . DataDriven	4	4	0	00:00:03	
Suites . FillContactForm	3	3	0	00:01:30	
Suites . HandleAndFrame	4	4	0	00:01:33	
Suites . HandleTable	1	1	0	00:00:16	
Suites . HandleWindow	3	3	0	00:01:09	
Suites . LoginTest	2	2	0	00:00:27	
Suites . SearchAddToCart	1	1	0	00:00:20	

Test Execution Log

SUITE Suites
Full Name: Suites
Documentation: Pabot result from 7 executions.
Start / End / Elapsed: 20200520 11:25:57.406 / 20200520 11:27:56.893 / 00:01:59.487
Status: 18 critical test, 18 passed, 0 failed
18 test total, 18 passed, 0 failed

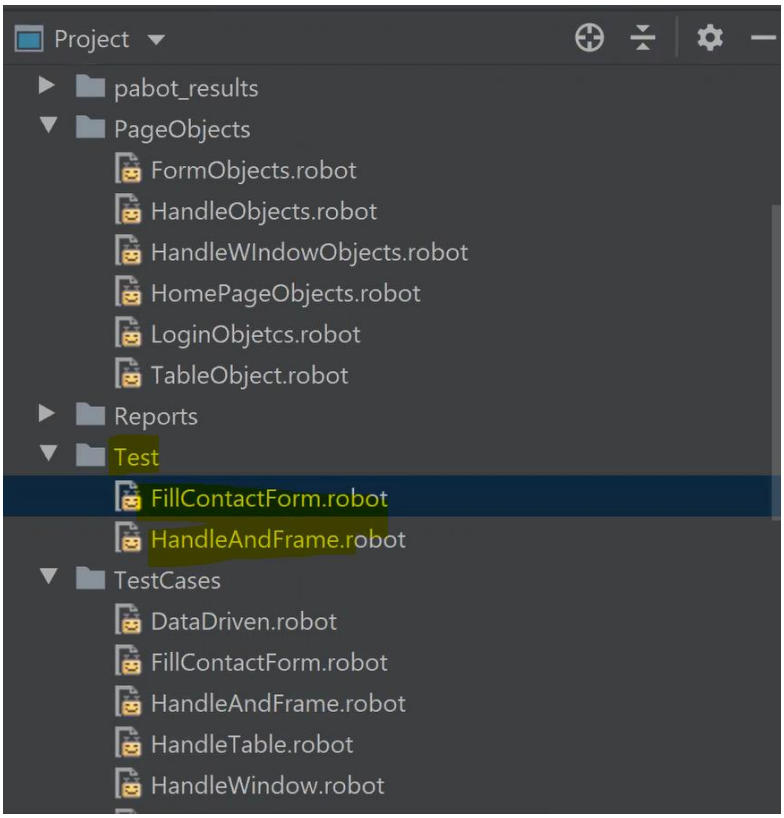
SUITE DataDriven
SUITE FillContactForm
SUITE HandleAndFrame
SUITE HandleTable
SUITE HandleWindow
SUITE LoginTest
SUITE SearchAddToCart

This is be default running test cases at suite level meaning in order it executes 1st suite then 2nd and so on . Now let see how we can parallel execute at level split.

We will be using command line option --testlevelsplit

--testlevelsplit
Split execution on test level instead of default suite level. If .pabotsuitenames contains both tests and suites then this will only affect new suites and split only them. Leaving this flag out when both suites and tests in .pabotsuitenames file will also only affect new suites and add them as suite files.

Let me create a Test directory and move two suites in it , as you can see I moved FillContactForm.robot and HandleAndFrame.robot



Now lets run

pabot --processes 2 --testlevelsplit Test/*.robot

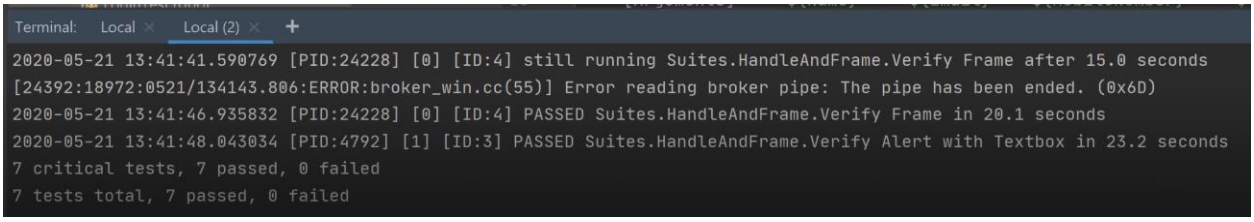
```
(venv) C:\Users\user\PycharmProjects\robotframeworkautomation>pabot --processes 2 --testlevelsplit Test/*.robot
```

Keep in mind it will execute any test in any order so no order is followed.

You will see it will launch two browsers, in [0] its running test cases from FillContactForm and in [1] its running HandleAndFrame . Its running suite in alphabetical order.

```
Terminal:  Local  Local (2)  +
Storing .pabotsuitenames file
2020-05-21 13:40:12.239452 [PID:22304] [0] [ID:0] EXECUTING Suites.FillContactForm.Fill contact form for ${Name}
2020-05-21 13:40:12.239452 [PID:24912] [1] [ID:1] EXECUTING Suites.HandleAndFrame.Verify Alert
```

Output on console will be like this



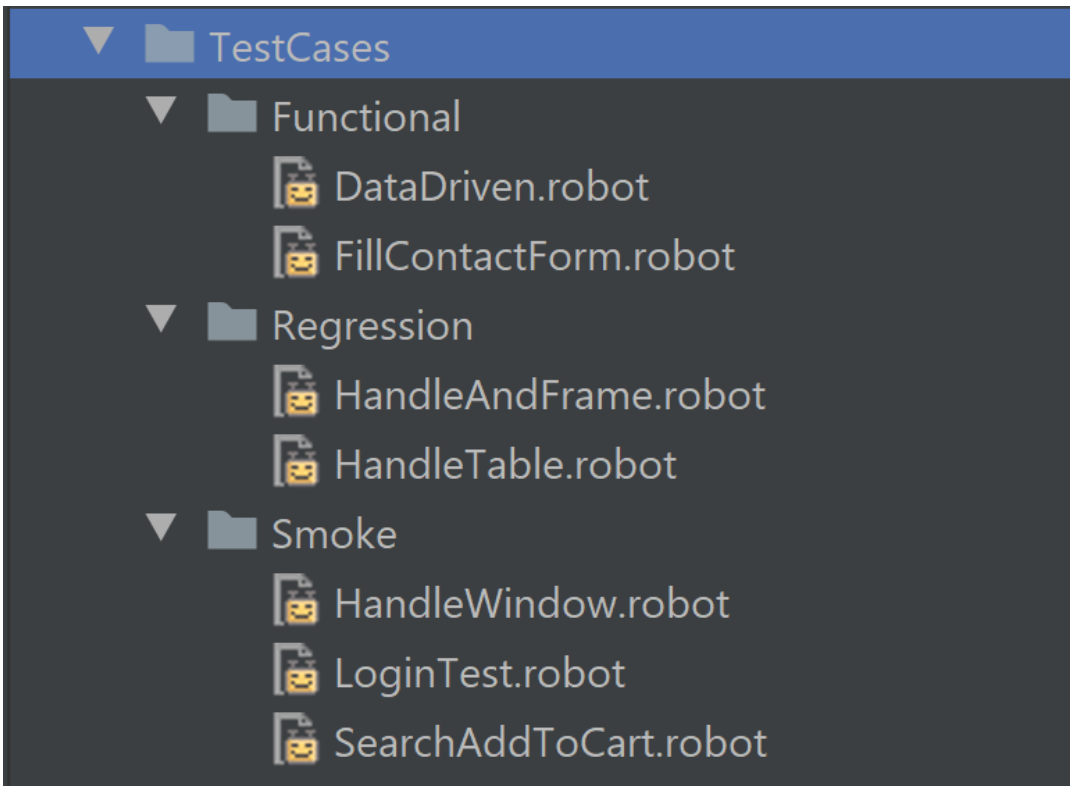
You can further understand the order of parallel execution by visint this site https://pabot.org/

Now restore folder struct back to normal move two suites from Test folder to under TestCases directory and then delete the Test directory.

Folder Restructure : Test Type Architect

Now lets organize the TestCases into folder structure representing different type of testing for example folder for Smoke test cases, folder for Functional test cases, and folder for Regression. For the tutorial purpose we will simulate by moving some test suites two into each folder, just imagine in real work scenario you have these three-common types of testing. In functional we usually keep those tests which we are testing in active sprint, in regression we keep those tests which were executed in previous sprints meaning old functionality and smoke test to ensure basic functionality is intact. So as we progress we will move from sprint to sprint tests in functional to regression and constantly update the smoke tests. This is standard practise and will try to fit in our project as well.

Here is how it will look



As we have added one more directory level we will need to update the paths in import statements, for example in DataDriven.robot we added one more ../ to paths as you can see highlighted below, reason is we have created additional directory and in order to access these we have to move out of that directory

DataDriven.robot

```
*** Settings ***
#Resource    ../../Utility/Setup.robot
#Resource    ../../Helper/TableHelper.robot
Library     ExcelLibrary
Library     DataDriver  file=${EXECDIR}/TestData/TestSheet.xlsx     sheet_name=Sheet1
Resource    ../../TestData/ApplicationProperties.robot
#Suite Setup  Open the browser in
#Test Setup  Go back to homepage
#Test Teardown  Capture screenshot if test failed   Dummy
#Suite Teardown  Close All Browser Window
Test Template   DataDriven Test cases
```

FillContactForm.robot

```
*** Settings ***
Resource    ../../Utility/Setup.robot
Resource    ../../Helper/FormHelper.robot
Library     ExcelLibrary
Library     DataDriver  file=${EXECDIR}/TestData/TestSheet.xlsx     sheet_name=Contac
Resource    ../../TestData/ApplicationProperties.robot
Suite Setup  Open the browser in
Test Setup  Go back to homepage
Test Teardown  Capture screenshot if test failed   Dummy
```

HandleAndFrame.robot

```
*** Settings ***
Resource    ../../Utility/Setup.robot
Resource    ../../Helper/HandleHelper.robot
Resource    ../../TestData/ApplicationProperties.robot
Suite Setup  Open the browser in
```

HandleTable.robot

```
*** Settings ***
Resource    ../../Utility/Setup.robot
Resource    ../../Helper/TableHelper.robot
Resource    ../../TestData/ApplicationProperties.robot
```

HandleWindow.robot

```
*** Settings ***
Resource    ../../Utility/Setup.robot
Resource    ../../Helper/HandleWindowHelper.robot
Resource    ../../TestData/ApplicationProperties.robot
Suite Setup    Open the browser in
```

LoginTest.robot

```
*** Settings ***
Resource    ../../Utility/Setup.robot
Resource    ../../Helper/LoginHelper.robot
Resource    ../../TestData/ApplicationProperties.robot
```

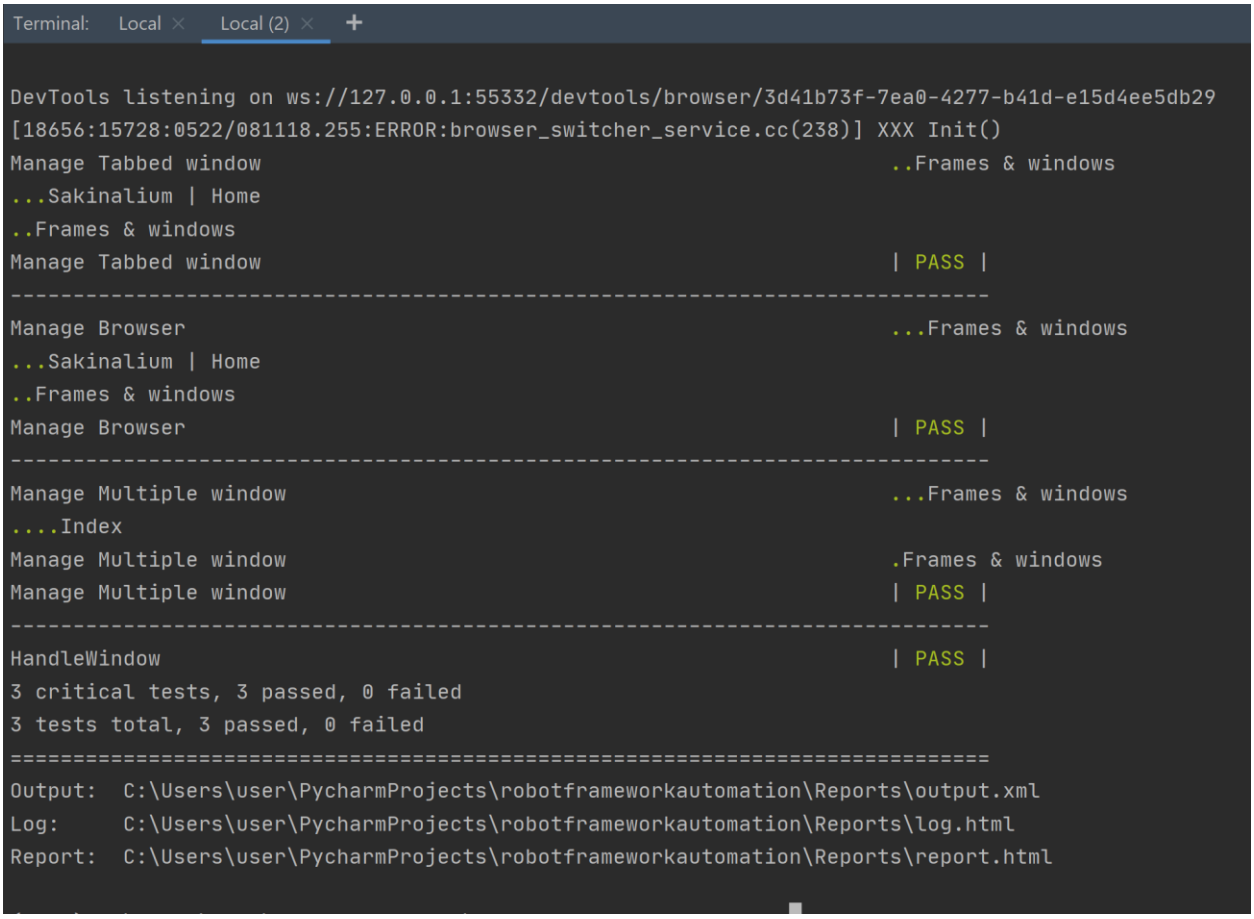
SearchAddToCart.robot

```
*** Settings ***
Resource    ../../Utility/Setup.robot
Resource    ../../Helper/HomePageHelper.robot
Resource    ../../TestData/ApplicationProperties.robot
```

Save all files. Lets try to run tests under smoke directory, run the following command

robot -d “Reports” TestCases/Smoke/HandleWindow.robot

Output on console:

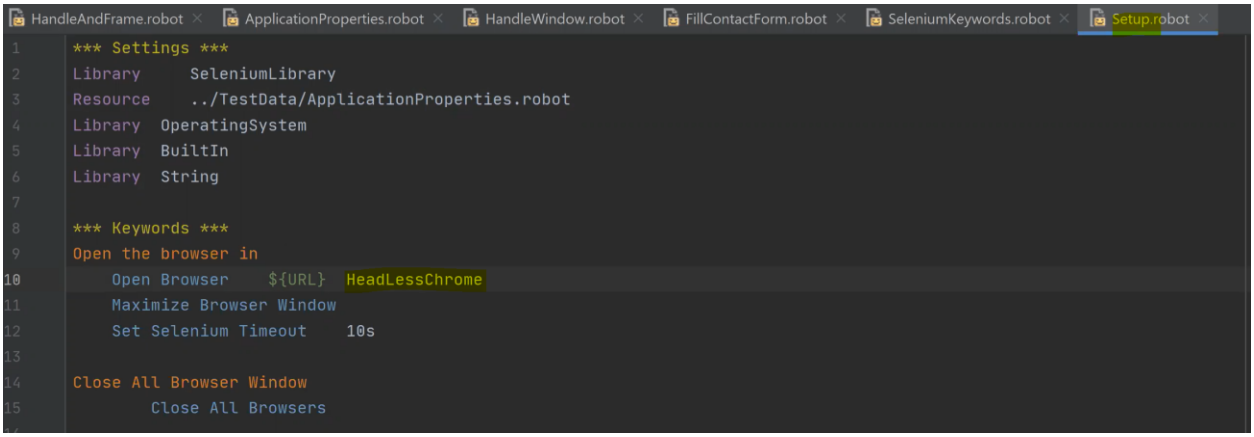


Headless Browser Testing:

In robotframework we can run tests in headless mode by using HeadLessChrome instead of Chrome in our Setup.robot. Keep in mind in headless mode the windows doesn't get maximized and if website doesn't have proper bootstrapped meaning to adjust components according to size of screen it will fail tests because it will overlap ui elements and wont be able to perform actions.

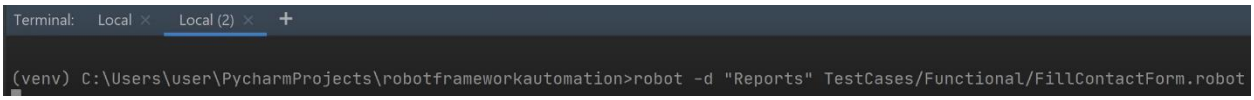
Lets run the FillContactForm.robot in headless mode we will see it fail at certain step and I will show you why it fails and will explain how we can fix this (this is in case if website is not properly designed like I developed https://amjottawaeast-27647.web.app/).

First go to Setup.robot and change from Chrome to HeadLessChrome (or HeadlessChrome) .

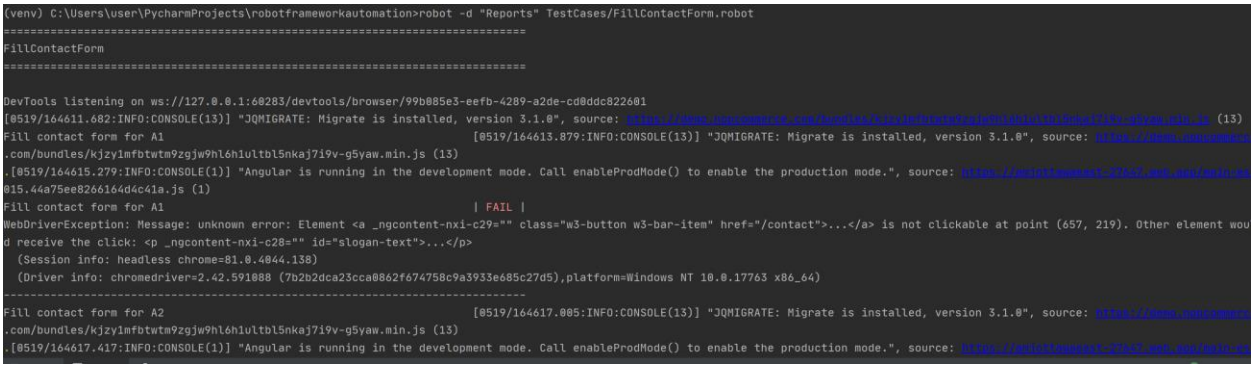


Now lets run the FillContactForm.robot

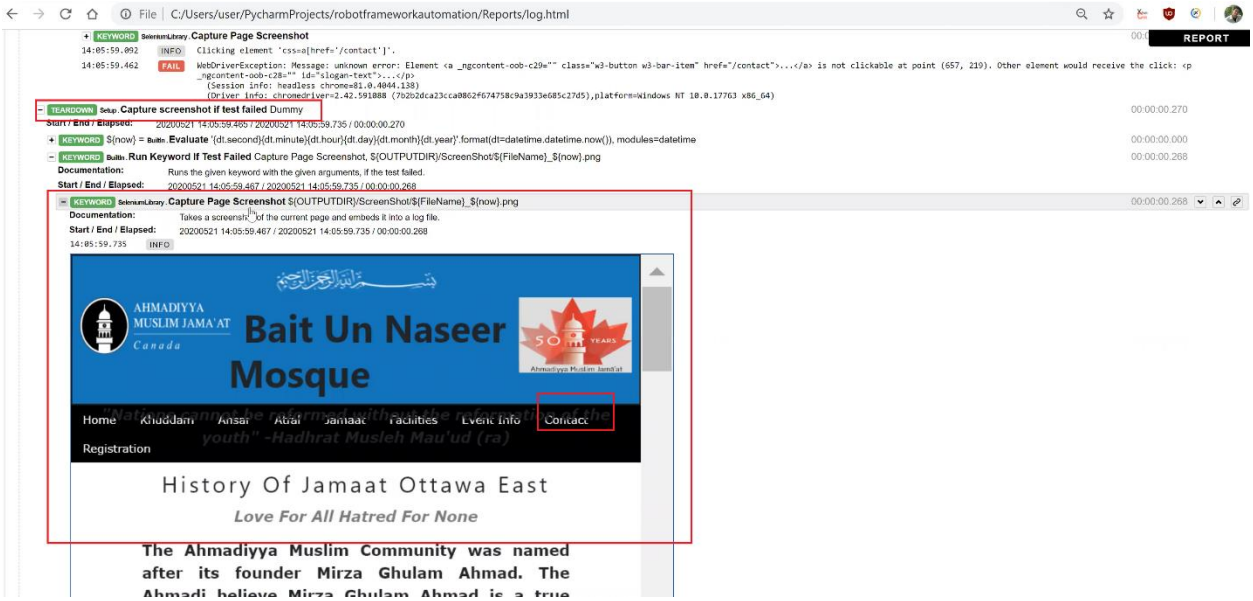
robot -d “Reports” TestCases/Functional/FillContactForm.robot



Output on console:



Now good thing is we have setup framework in such a way as soon as test step fail it takes screenshot so lets checkout and I want to show you why it failed. Lets open test report.



You can see above the Contact gets hidden/messed up as the browser wasn't maximized , thus wont able to click on it in headless mode, same test if you run in normal Chrome mode it will pass .

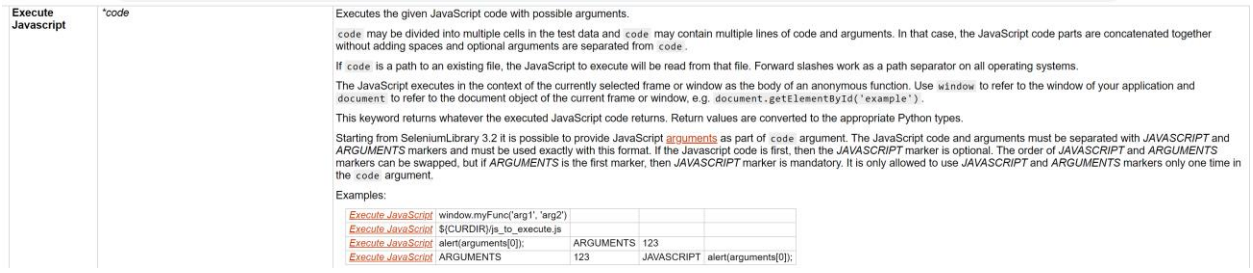
Now let see how we can fix this, what we will do is we will use Javascript executor to click on it. This will fix this in both Chrome and HeadlessChrome mode.

We added the function in SeleniumKeywords.robot

```
Perform JavaScript Action
    [Arguments]    ${script}
    Execute Javascript    ${script}
```

The Execute Javascript is SeleniumLibrary keyword which will help us to execute javascript commands which are used to perform certain actions by locating element like click similar to what we do with them.

Execute Javascript:
https://robotframework.org/SeleniumLibrary/SeleniumLibrary.html#Execute%20Javascript



Now lets go to FormHelper.robot that's where we will write script and pass it to as parameter. As you can see below we are using javascript command document.querySelector("a[href='/contact']").click

This is to click on the contact link by looking into DOM .

```
Click on Contact link
    #Click On Page Element    ${ContactLink}
    #Click On Page Element by JavaScript    ${ContactLink}
    Perform JavaScript Action    document.querySelector("a[href='/contact']").click()
```

Save all files .

Code: HelperForm.robot

```
*** Settings ***
Resource    ../PageObjects/FormObjects.robot
Resource    ../Utility/SeleniumKeywords.robot
Library     BuiltIn

*** Keywords ***
Navigate to the site
   [Arguments]    ${URL}
   Navigate to the page    ${URL}

Click on Contact link
   #Click On Page Element   ${ContactLink}
   #Click On Page Element by JavaScript   ${ContactLink}
   Perform JavaScript Action   document.querySelector("a[href='/contact']").click()

Enter name
   [Arguments]    ${text}
   Set Value For Input Field   ${Name}    ${text}

Enter email
   [Arguments]    ${text}
   Set Value For Input Field   ${Email}   ${text}

Enter MobileNumber
   [Arguments]    ${text}
   Set Value For Input Field   ${MobileNumber}    ${text}

Enter Subject
   [Arguments]    ${text}
   Set Value For Input Field   ${Subject}    ${text}

Enter Message
   [Arguments]    ${text}
   Set Value For Input Field   ${Message}    ${text}

Click on submit button
   Click On Button    ${SubmitButton}
```

Code: SeleniumKeywords.robot

```
*** Settings ***
Library    SeleniumLibrary

*** Keywords ***
Clear Input Field
   [Arguments]   ${locator}
   Wait For Element Present   ${locator}  5   Element not found in 5 seconds
   Clear Element Text   ${locator}
   ${value}=    Get Element Attribute   ${locator}    value
   ${backspaces count}=   Get Length    ${value}
   Run Keyword If   """${value}""" != ''
   ...    Repeat Keyword  ${backspaces count +1}  Press Key  ${locator}   \\08

Set Value For Input Field
   [Arguments]   ${locator}   ${inputval}
   sleep   2s
   Clear Input Field   ${locator}
   Input Text   ${locator}   ${inputval}

Click On Page Element
   [Arguments]   ${locator}
   Wait For Element Present   ${locator}  5   Element not found in 5 seconds
   Click Element   ${locator}

Click On Button
   [Arguments]   ${locator}
   Wait For Element Present   ${locator}  5   Element not found in 5 seconds
   Click Button   ${locator}

Wait For Page Load
   [Arguments]   ${pgloadlocator}
   Wait Until Page Contains Element   ${pgloadlocator}
   Sleep   5s

Wait For Element Present
   [Arguments]   ${locator}   ${timeout}   ${err_msg}
   #Wait For Element Present   ${locator}  5   Element not found in 5 seconds
   Wait Until Page Contains Element   ${locator}   ${timeout}   ${err_msg}

Get Attribute Value
   [Arguments]   ${locator}   ${attribute}
   #Wait For Element Present   ${locator}  5   Element not found in 5 seconds
   ${string}=   Get Element Attribute   ${locator}   ${attribute}
   log to console   \n get attribute ${string}.\n
   [Return]   ${string}

Back On Screen
   Go Back

Wait For Text Present
   [Arguments]   ${text}   ${seconds}   ${message}
   Wait Until Page Contains   ${text}   ${seconds}   ${message}
```

```
Verify Text Present
    [Arguments]    ${text}
    Page Should Contain    ${text}

Verify Element Present
    [Arguments]    ${locator}
    #Wait For Element Present    ${locator}  5    Element not found in 5 seconds
    Page Should Contain Element    ${locator}

Generate Random String With Defined Size
    [Arguments]    ${size}    ${type}
    ${str}=    Generate Random String    ${size}    ${type}
    [Return]    ${str}

Generate Random Number With Defined Size
    [Arguments]    ${size}    ${type}
    ${num}=    Generate Random String    ${size}    ${type}
    [Return]    ${num}

Match Value
    [Arguments]    ${locator}    ${attr_name}    ${match_pattern}
    #Wait For Element Present    ${locator}  5    Element not found in 5 seconds
    Element Attribute Should Match    ${locator}    ${attr_name}    ${match_pattern}

Verify Text Not Present
    [Arguments]    ${text}
    Page Should Not Contain    ${text}

Select List Option By visible Text
    [Arguments]    ${locator}    ${value}
    Wait For Element Present    ${locator}  5    Element not found in 5 seconds
    Select From List By Label    ${locator}    ${value}

Verify Page title
    [Arguments]    ${title}
    Title Should Be    ${title}

Navigate to the page
    [Arguments]    ${URL}
    Go To    ${URL}

Verify Element Displayed
    [Arguments]    ${locator}
    Wait For Element Present    ${locator}  5    Element not found in 5 seconds
    Element Should Be Visible    ${locator}

Verify Element not displayed
    [Arguments]    ${locator}
    Element Should Not Be Visible    ${locator}

Verify Current URL should contains
    [Arguments]    ${URL}
    Location Should Contain    ${URL}

Static wait for
```

```
    [Arguments]    ${Second}
    Sleep    ${Second}s

Get all element count
    [Arguments]    ${locator}
    ${num}=    Get Element Count    ${locator}
    [Return]    ${num}

Verify both values are equal
    [Arguments]    ${val1}  ${val2}
    Should Be Equal    ${val1}    ${val2}

Switch to window
    [Arguments]    ${type}
    Switch Window  ${type}

Close current window
    Close Window

Get Current URL
    return from keyword    Get Location

Press keyboard key
    [Arguments]    ${locator}   ${Key}
    Wait For Element Present    ${locator}  5    Element not found in 5 seconds
    Press Key  ${locator}   \\08

Mouse Over on the element
    [Arguments]    ${locator}
    Wait For Element Present    ${locator}  5    Element not found in 5 seconds
    Mouse Over  ${locator}

URL Should Contain
    [Arguments]    ${expected}
    Location Should Contain    ${expected}

Scroll for element
    [Arguments]    ${locator}
    Scroll Element Into View    ${locator}

Scroll at the bottom of the page
    Execute Javascript    window.scrollTo(0,document.body.scrollHeight)

Scroll at the top of the page
    Execute Javascript    window.scrollTo(0,0)

Handle the alert box
    [Arguments]    ${action}
    Handle Alert    ${action}

Enter text in alert box
    [Arguments]    ${text}
    Input text into Alert   ${text}

Select a Frame
```

```
    [Arguments]  ${locator}
    Select Frame    ${locator}

UnSelect a Frame
    Unselect Frame

Get Text From element
    [Arguments]  ${locator}
    ${val}=    Get Text    ${locator}
    [return]    ${val}

Title of the window
    ${val}=    Get Title
    [return]    ${val}

Perform JavaScript Action
    [Arguments]    ${script}
    Execute Javascript    ${script}
```

Code: Setup.robot

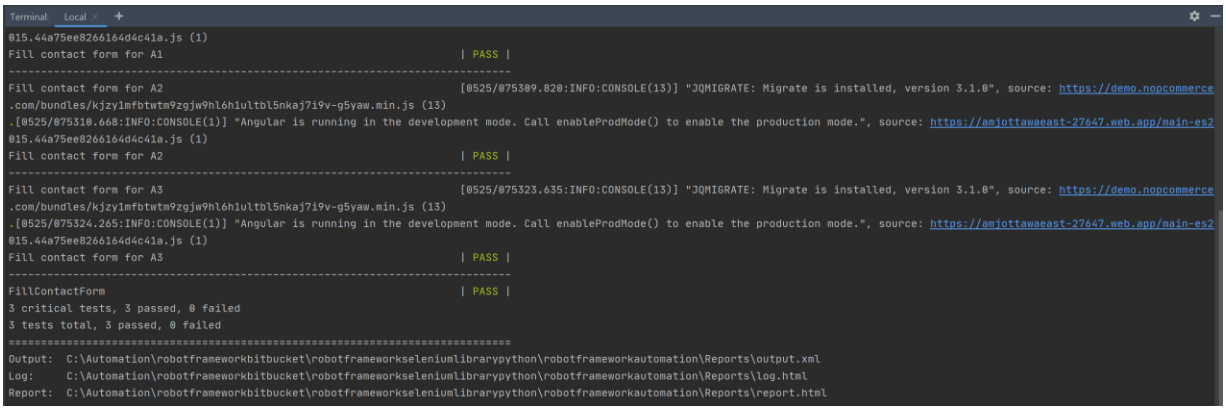
```
*** Settings ***
Library     SeleniumLibrary
Resource    ../TestData/ApplicationProperties.robot
Library  OperatingSystem
Library  BuiltIn
Library  String

*** Keywords ***
Open the browser in
    Open Browser    ${URL}  headLessChrome
    log to console  headless
    Maximize Browser Window
    Set Selenium Timeout    10s

Close All Browser Window
        Close All Browsers

Go back to homepage
    Go To   ${URL}

Capture screenshot if test failed
    [Arguments]  ${FileName}
    ${now}    Evaluate
'{dt.second}{dt.minute}{dt.hour}{dt.day}{dt.month}{dt.year}'.format(dt=datetime.datetime.now())
modules=datetime
    Run Keyword If Test Failed      Capture Page Screenshot
${OUTPUTDIR}/ScreenShot/${FileName}_${now}.png

Take screenshot
    [Arguments]  ${FileName}
    ${now}    Evaluate
'{dt.second}{dt.minute}{dt.hour}{dt.day}{dt.month}{dt.year}'.format(dt=datetime.datetime.now())
modules=datetime
    Capture Page Screenshot    ${OUTPUTDIR}/ScreenShot/${FileName}_${now}.png
```

Now run it again.

robot -d "Reports" TestCases/Functional/FillContactForm.robot

```
(venv) C:\Users\user\PycharmProjects\robotframeworkautomation>robot -d "Reports" TestCases/Functional/FillContactForm.robot
```

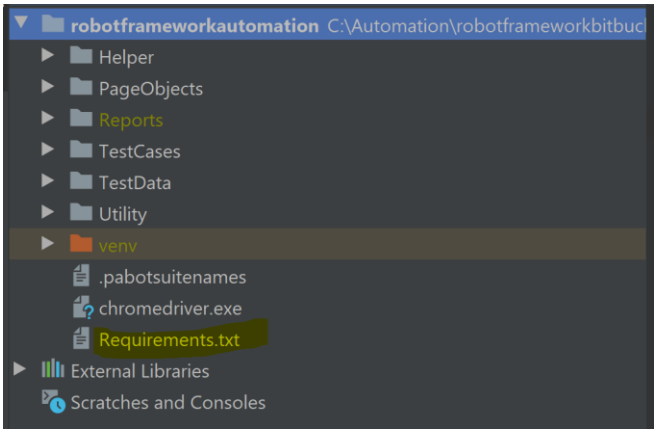
Console Output:

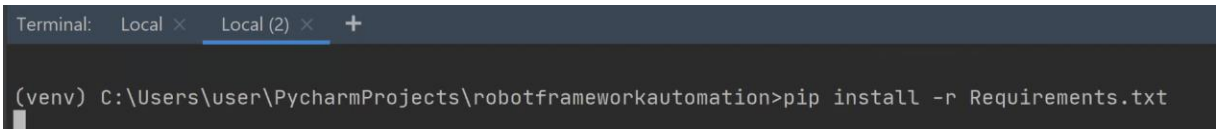


Requirements.txt : I created this file to list all the dependencies meaning libraries we installed, it serves two purposes one if you want to give this project to someone all that person needs to do is run pip install Requirments.txt it will install them automatically and add to project. Other purpose is for Jenkins wherever Jenkins will be before running tests we will need to install there as well we will use same command.

Code:Requirements.txt

```
robotframework
robotframework-seleniumlibrary
robotframework-pabot
robotframework-datadriver
```





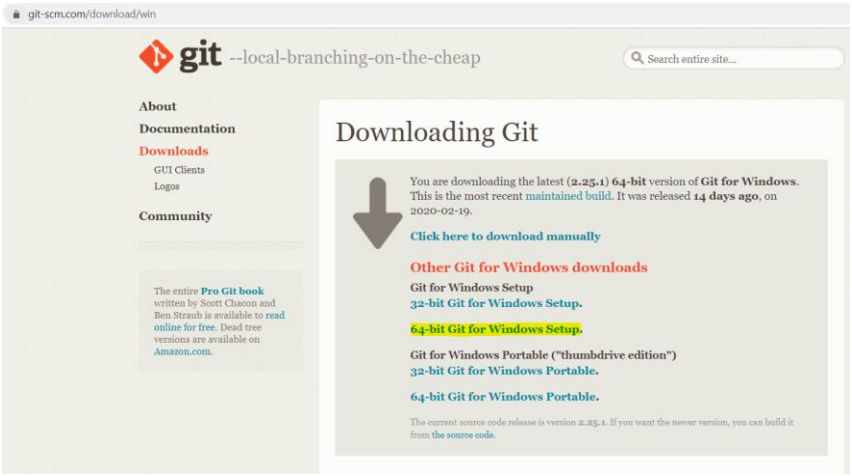
Bitbucket: Create Project & Repository

Pre-requisite:

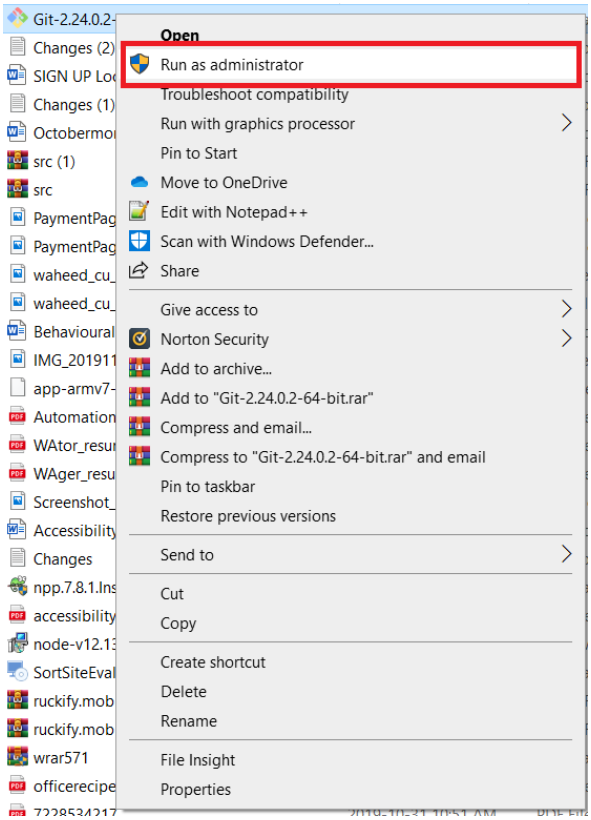
Git:

1. Install Git

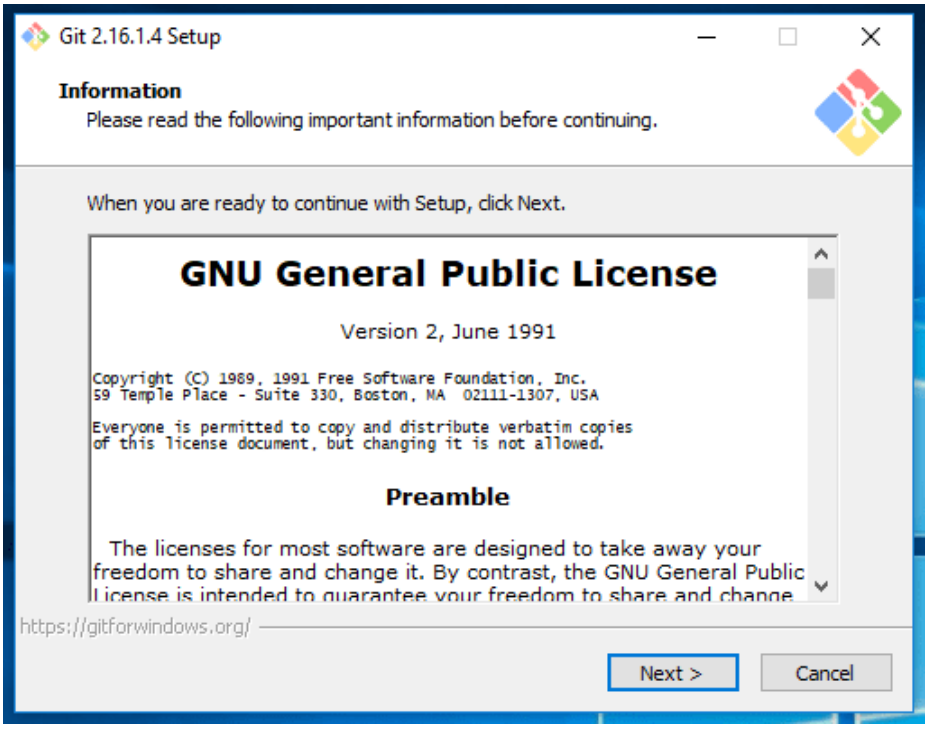
Go to official web site git-scm.com



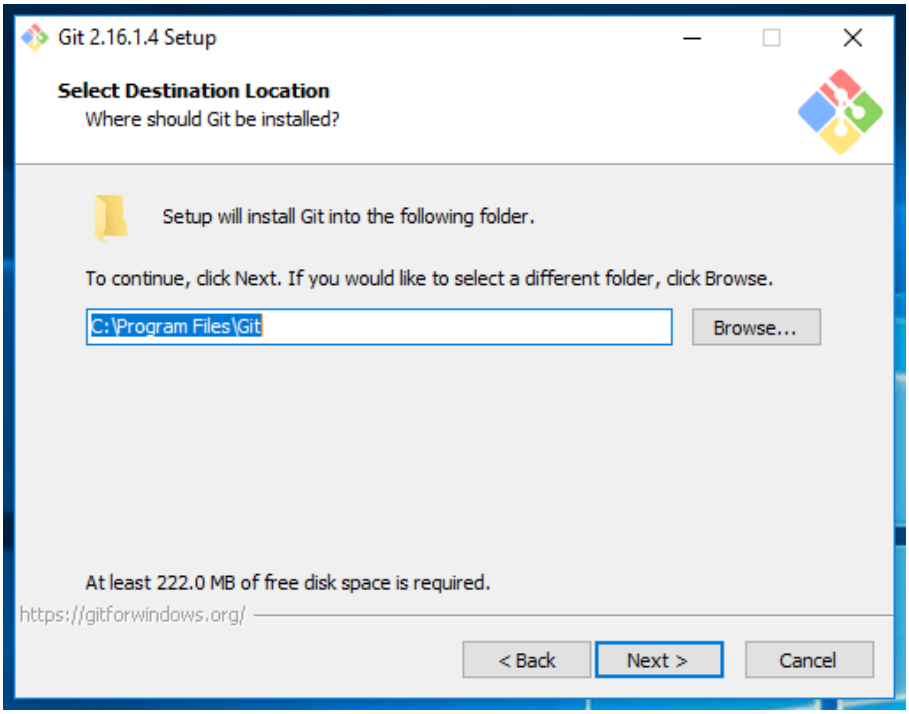
After download is complete right click and select Run as administrator



On installation wizard click on next

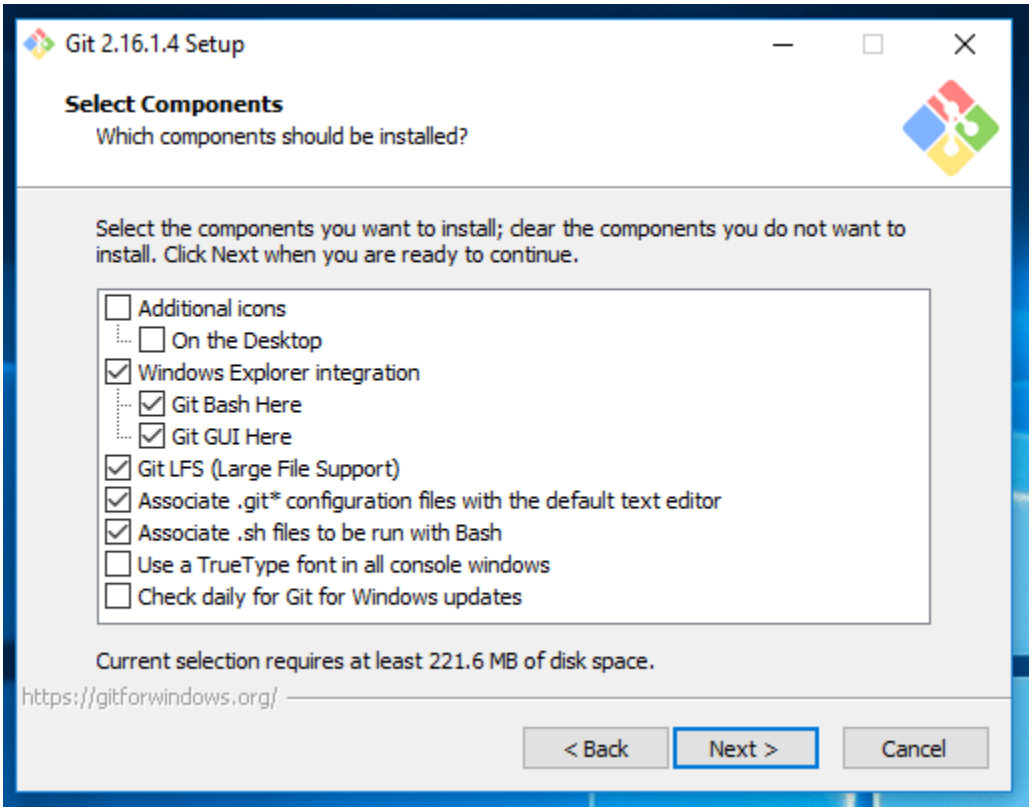


It will ask for Git installation directory by default it will be in C:\Program Files\Git, you can change if you wish.



Click on Next>

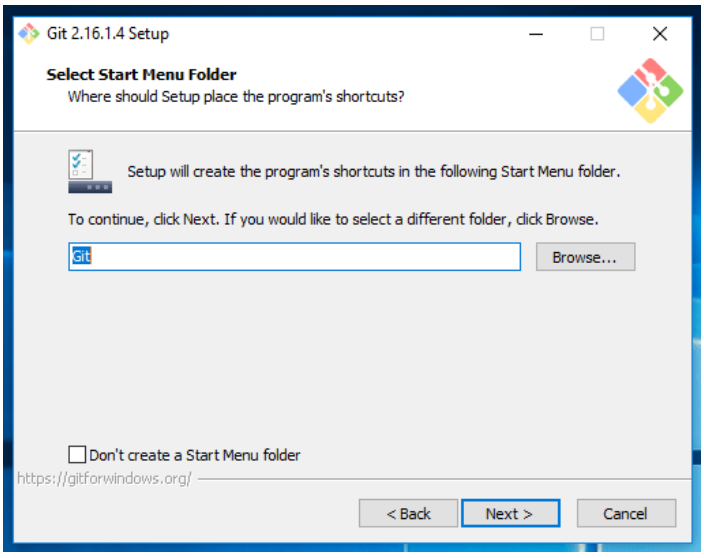
This step allows you to select the git components which you want to install as part of the git installation.



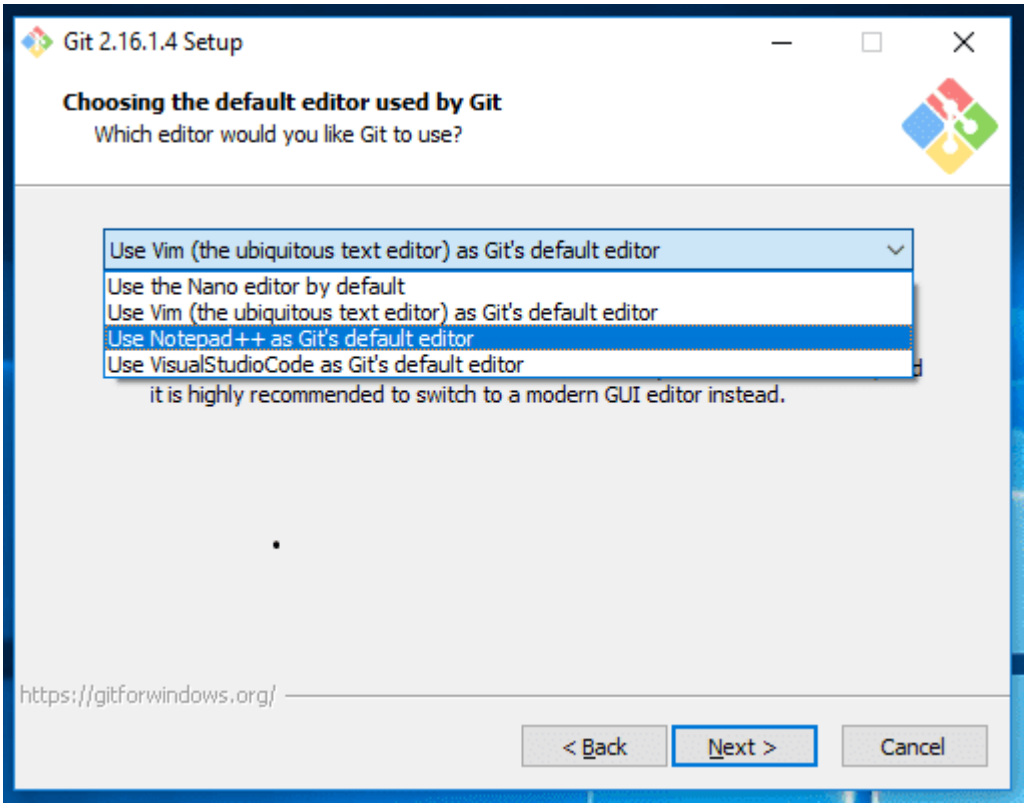
Above selection is recommended, those recommendations bring you Git GUI and Git Command prompt.

Click Next >

This dialogue asking you to create shortcuts of git, leave this as default and click Next >

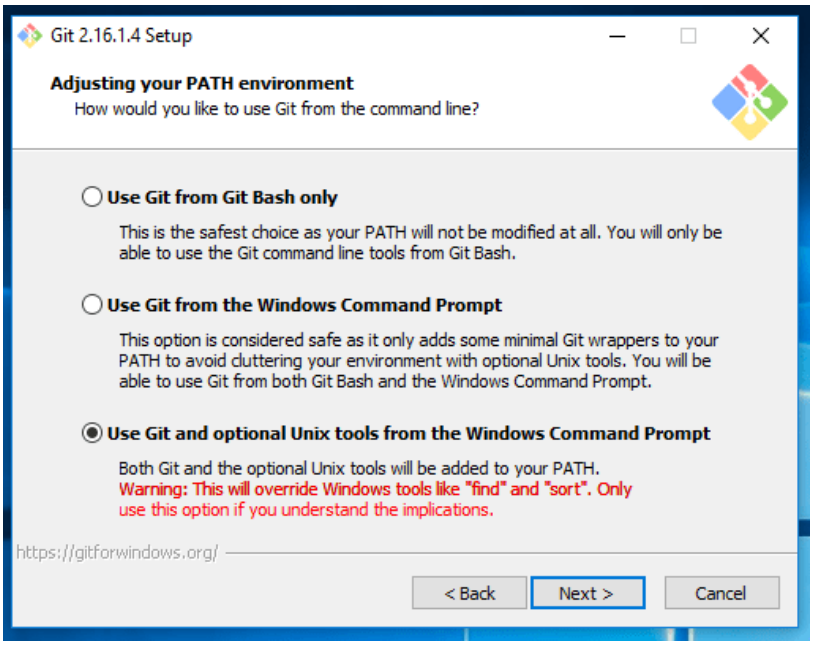


Choose your favourite text editor to work with Git.



Click on Next >

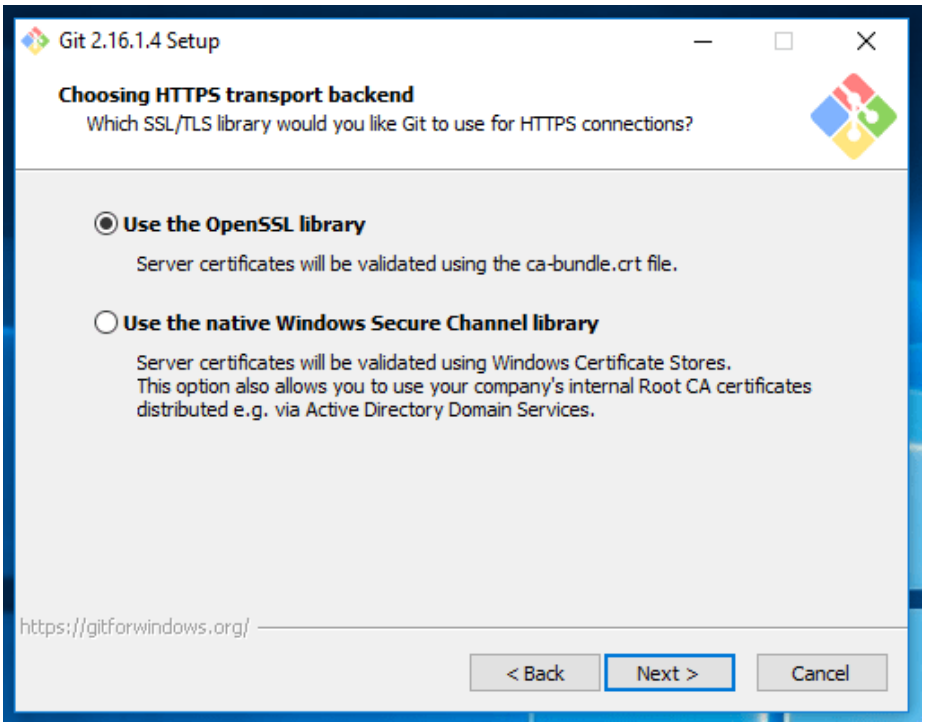
This dialogue asking you to how do you want to use git to set the PATH, go with the recommended option.



By choosing this step git automatically set PATH for you. I will show you how to see the generated PATH at the end of this article.

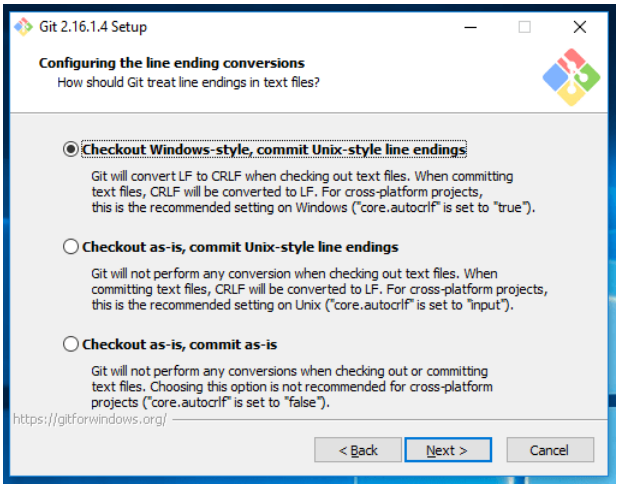
Click on Next >

Choose for HTTPS transport, leave it as default.



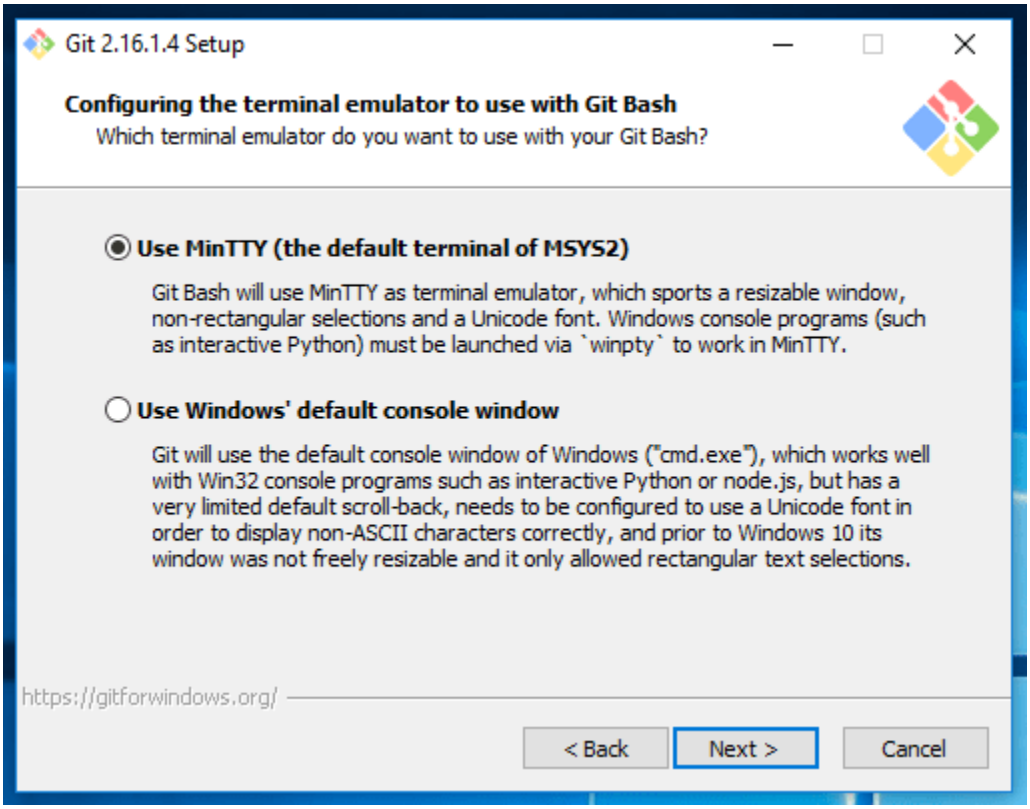
Click on Next >

Configuring the line ending conversions, leave it as default recommendation.



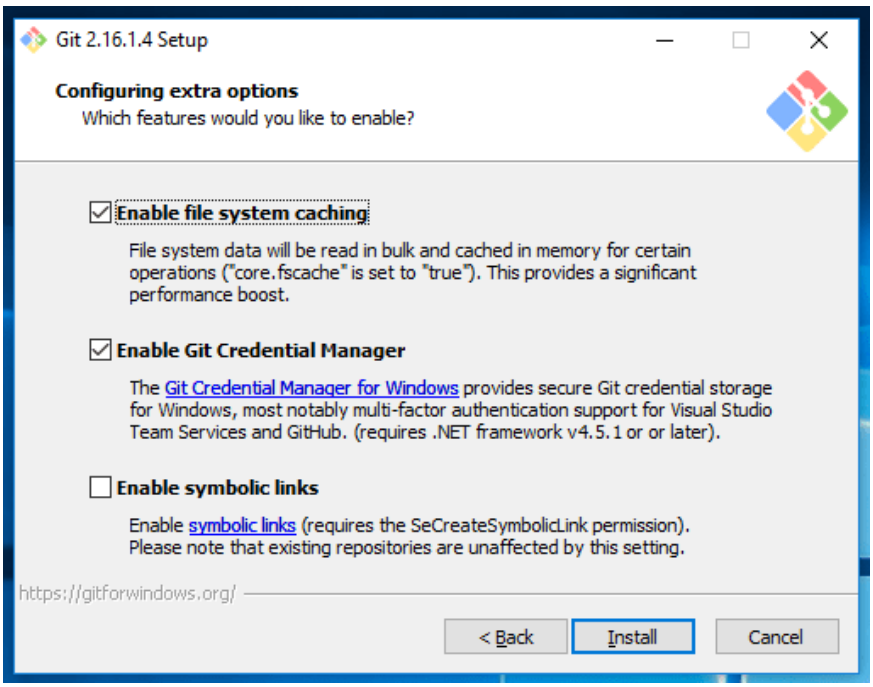
Click on Next>

Choose for the terminal emulator to use Git Bash.



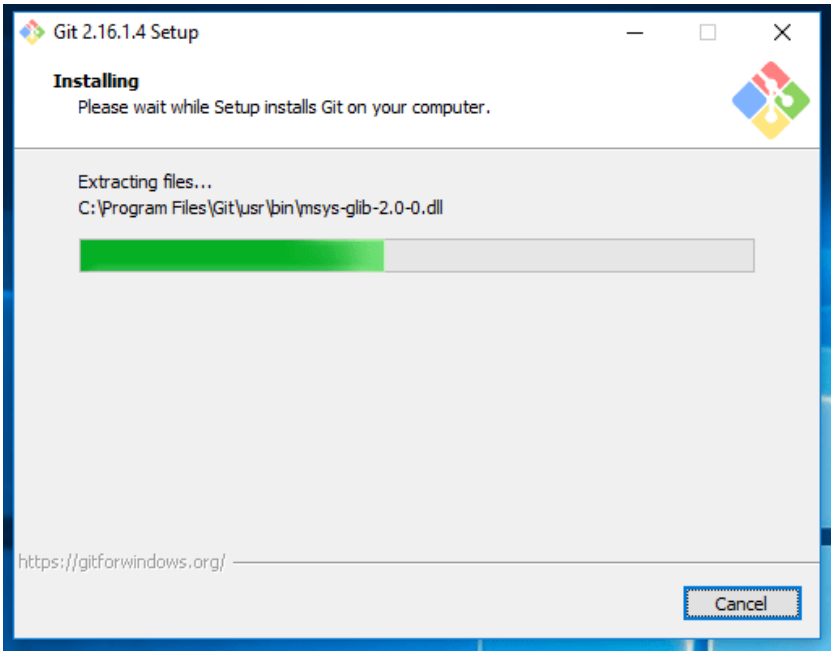
Click on Next >

This window allows you to enable extra features. Select the below recommendations.



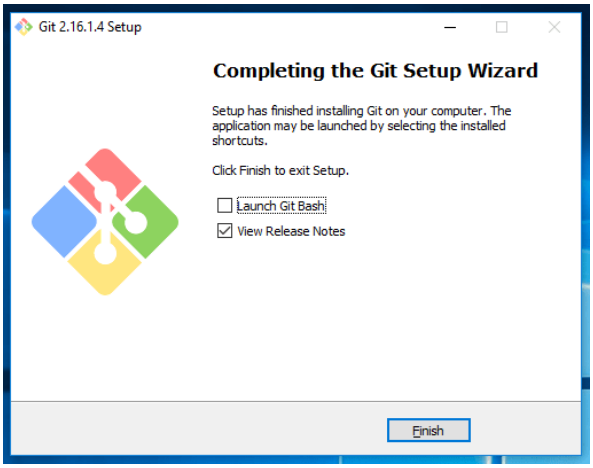
Click on Install

Now started the installation process.



It will take a couple of minutes to complete.

If everything is done well, you can see the below information.



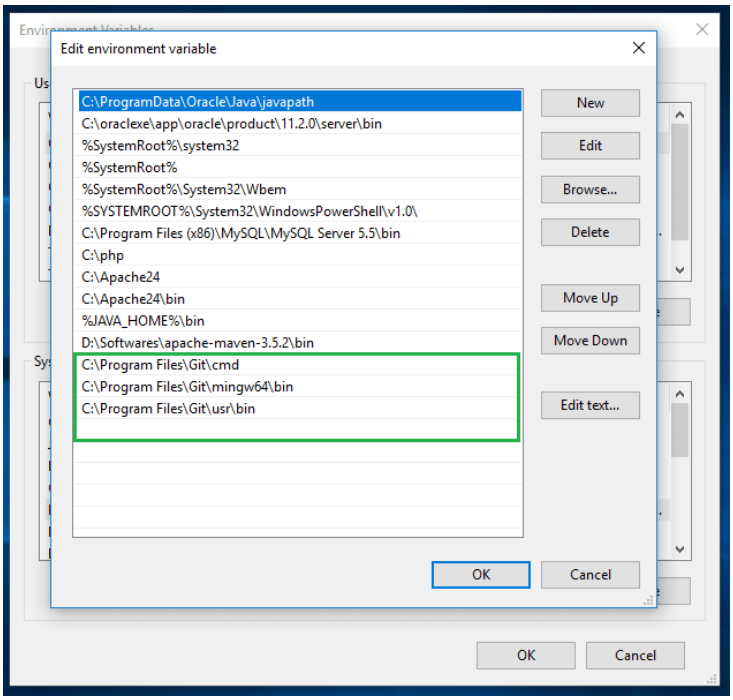
Click on Finish.

Now you can run the Git on Windows 10 operating system.

Check It :

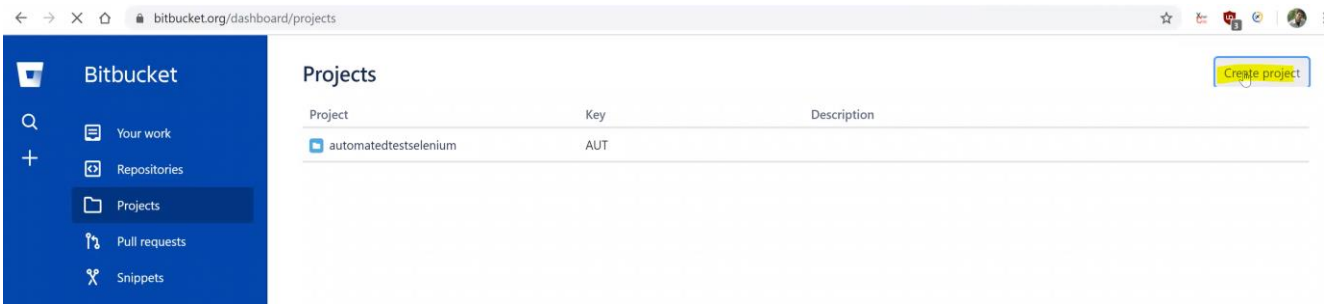
Check whether PATH has been set or not.

Advanced System Settings > Environment Variables > System Variables > select Path and click on Edit. You can see the below git paths.

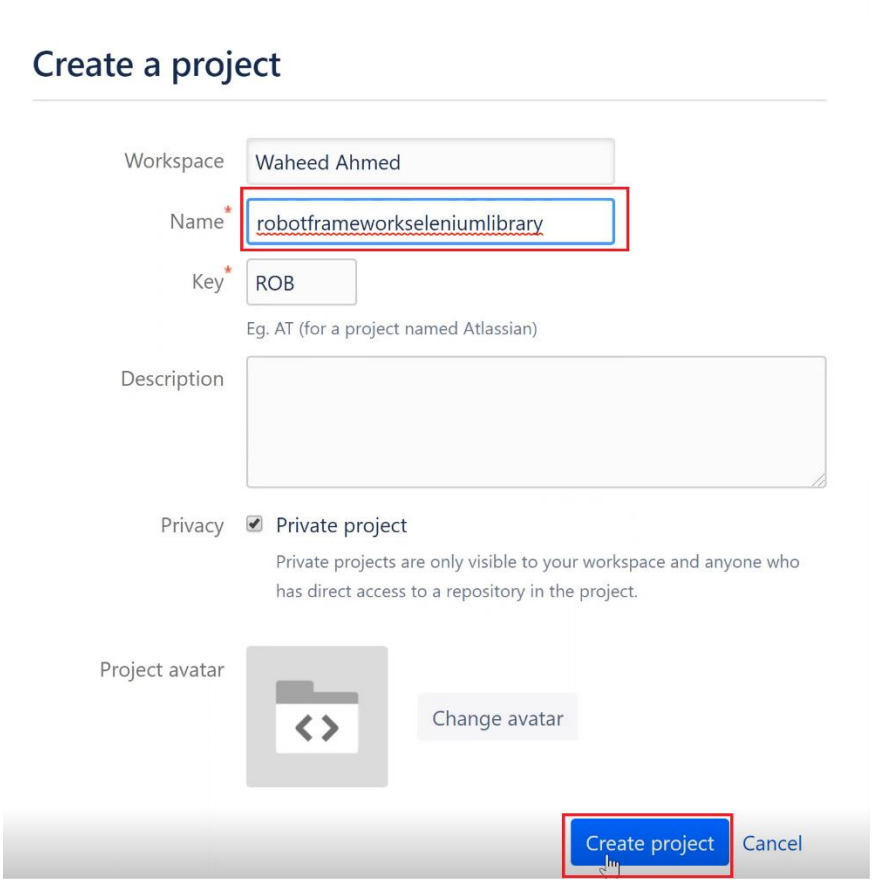


Create Project & Repository:

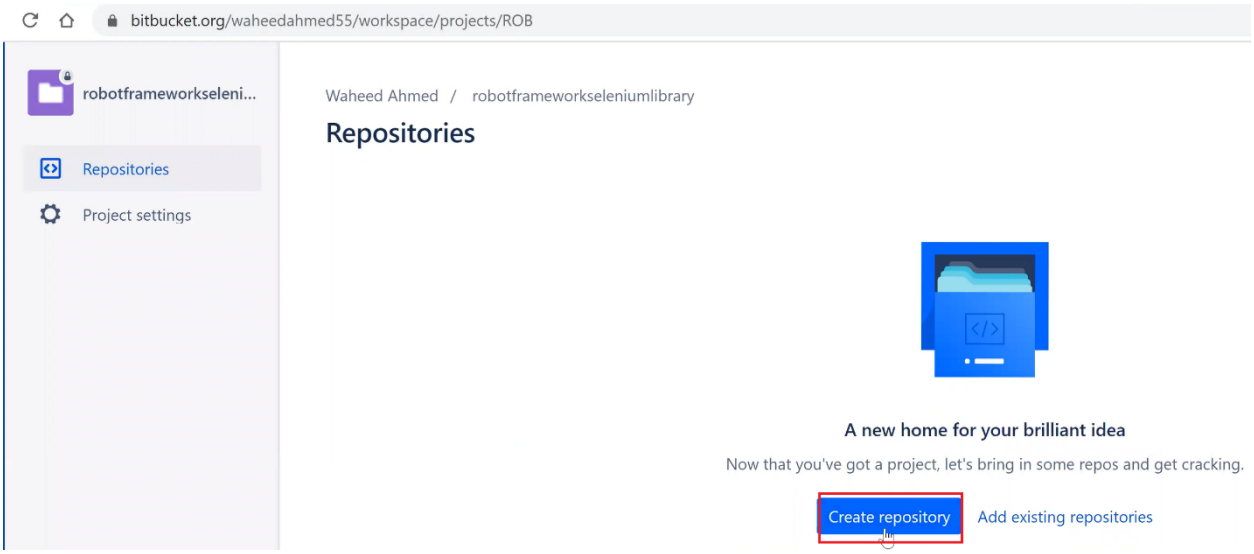
Frist we will create project in bitbucket . Click on Create Project



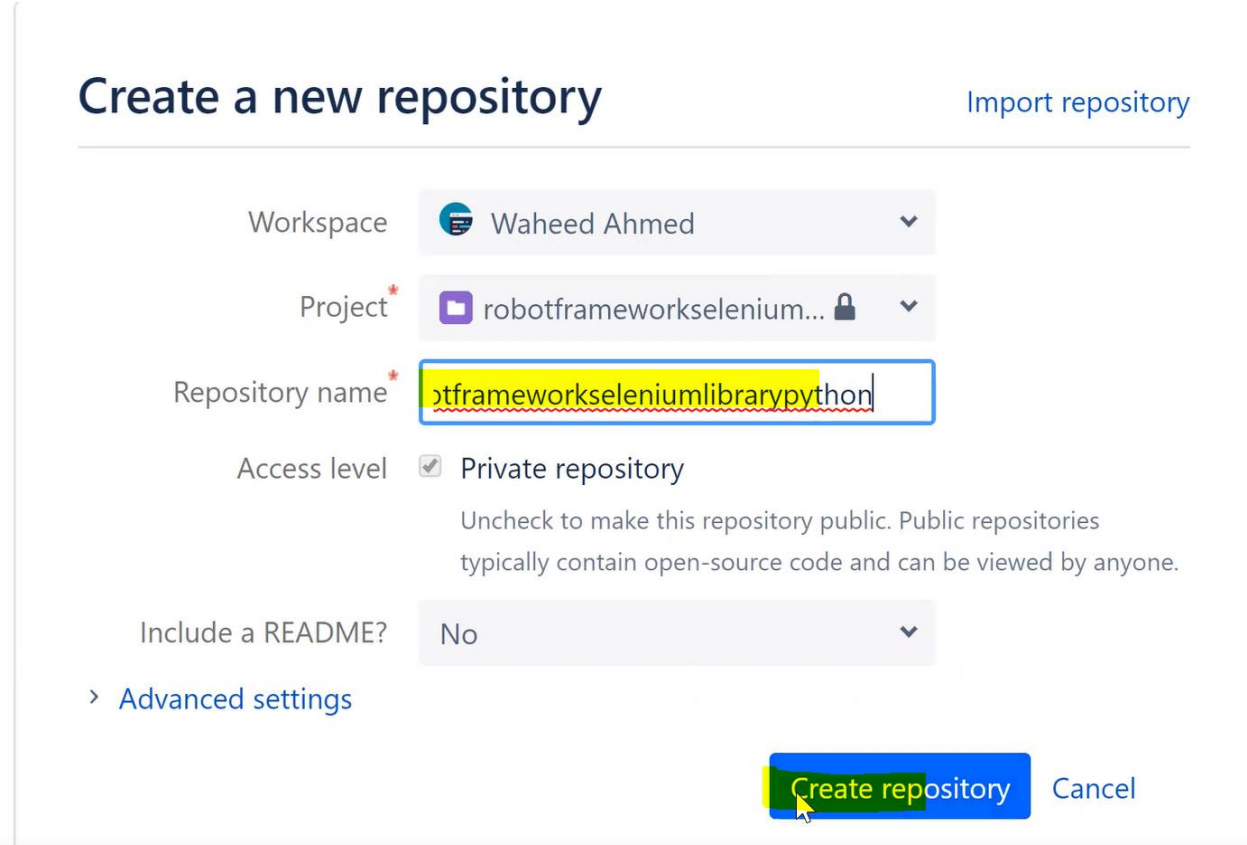
Set Name of the project and click on Create Project



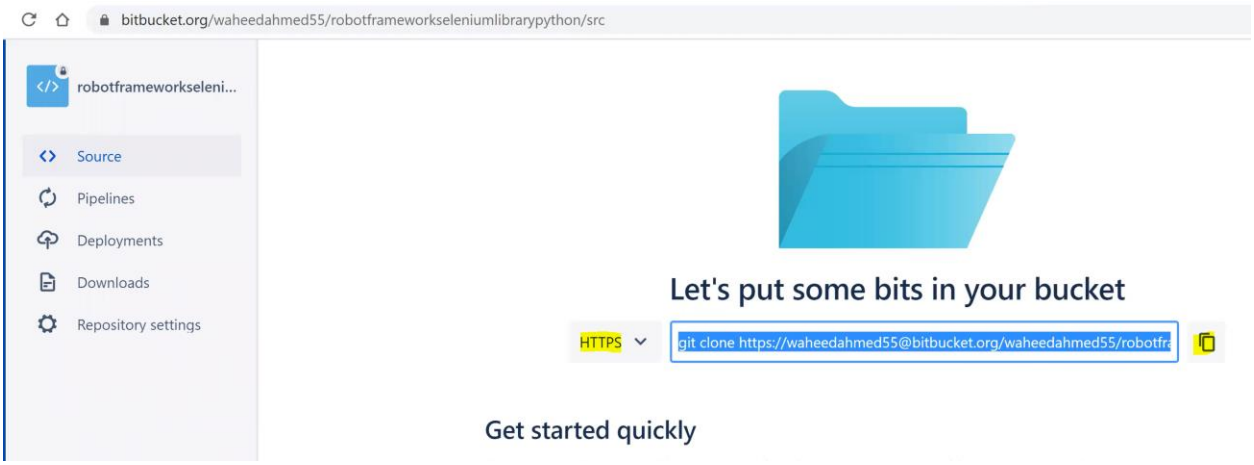
Now lets create Repository in which we will be pushing our test automation code. Click on Create repository



Set Repository name and click on Create repository



Now we have repository



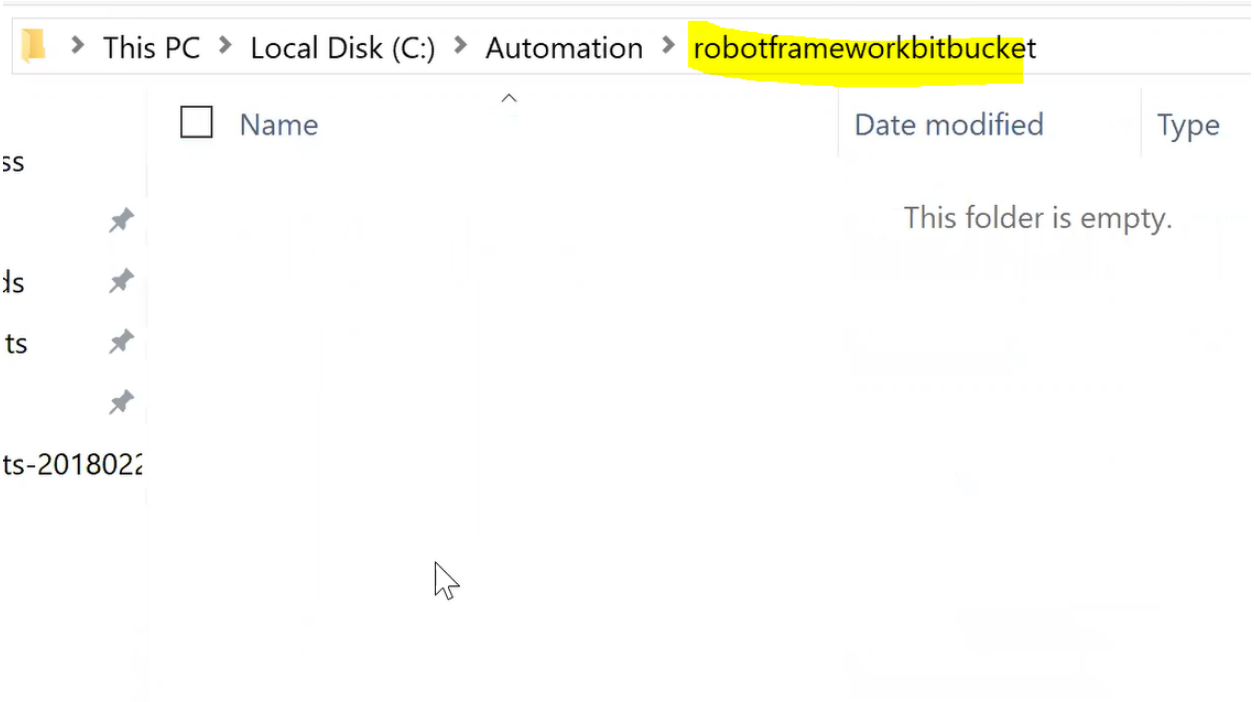
Copy the URL , we will need it in next step

git clone
https://waheedahmed55@bitbucket.org/waheedahmed55/robotframeworkseleniumlibrarypython.git

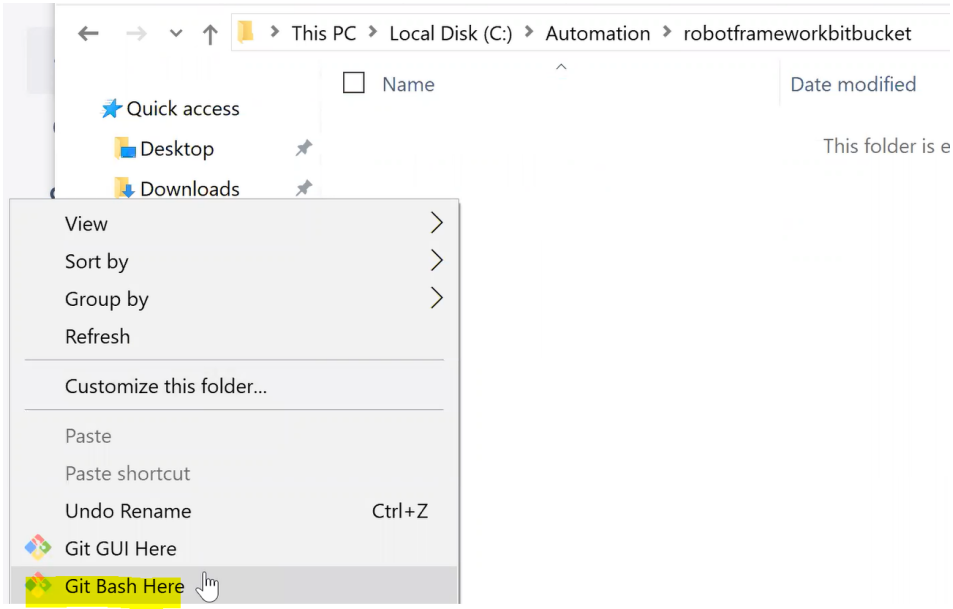
Git bash: Clone and Push the code to repository:

Now we will clone this repository in our local its empty for now , later we will add code and push it.

Before running git clone lets create folder in which we will clone the remote repository we just created in bitbucket. I created folder robotframeworkbitbucket.

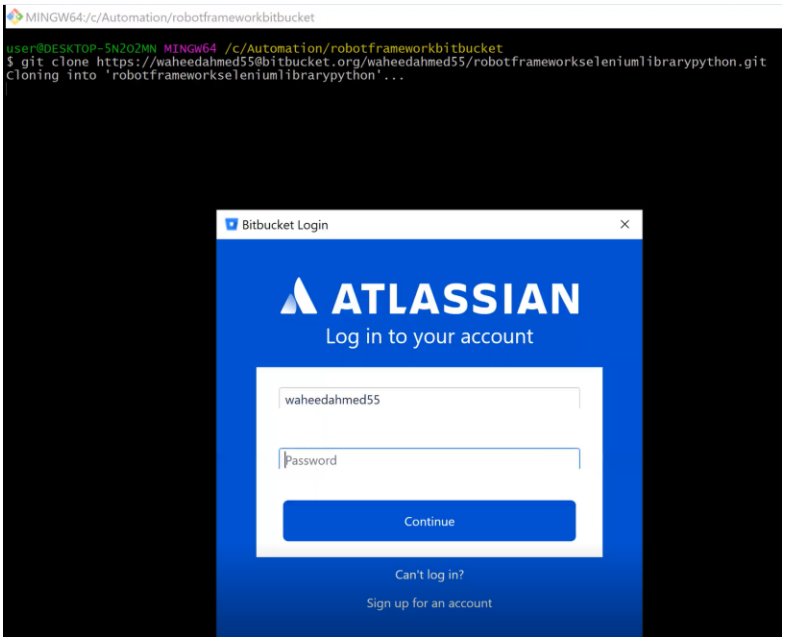


Now right click anywhere in the folder and select Git Bash Here

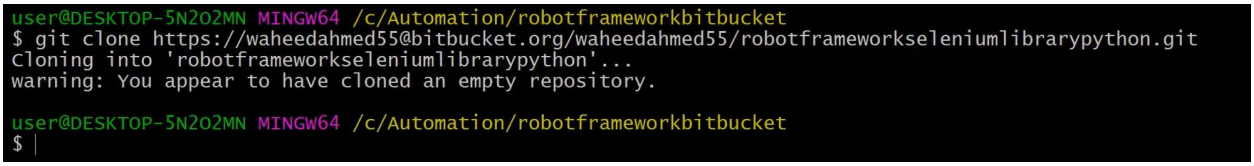


It will launch git command line interface in that folder. Now paste the git clone command which we copied from bitbucket when we created the repository and hit enter . It will clone the remote repository to local

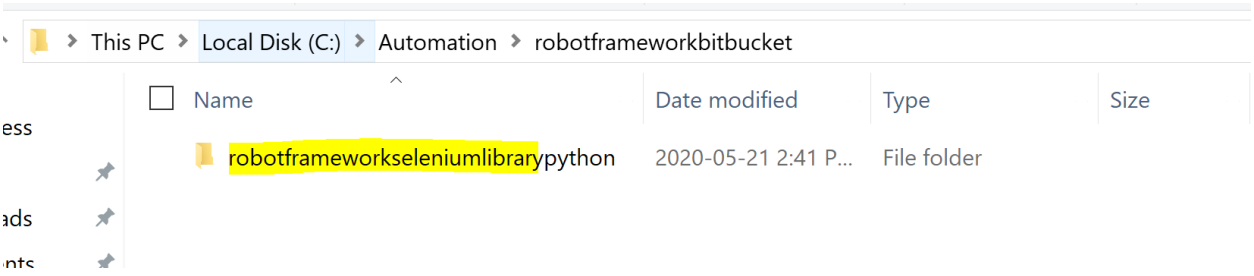
It will prompt you to enter your bitbucket credentials enter them and click



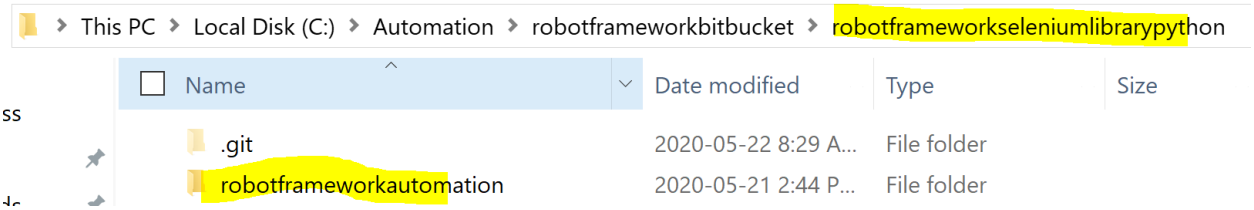
After cloning completes it will display



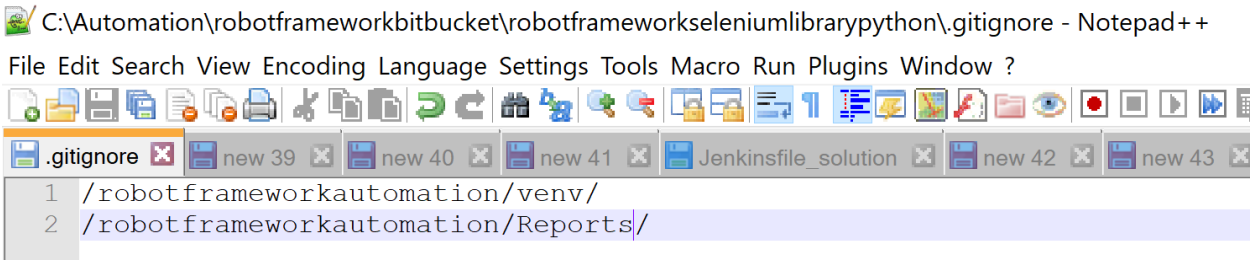
You will see empty folder with name of the repository

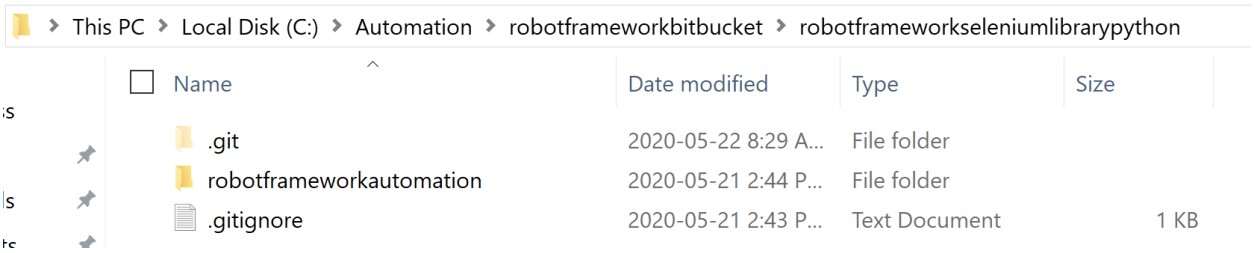


Now we will copy full project folder into this folder . We wont push everything to bitbucket for example we will ignore test results folder etc .

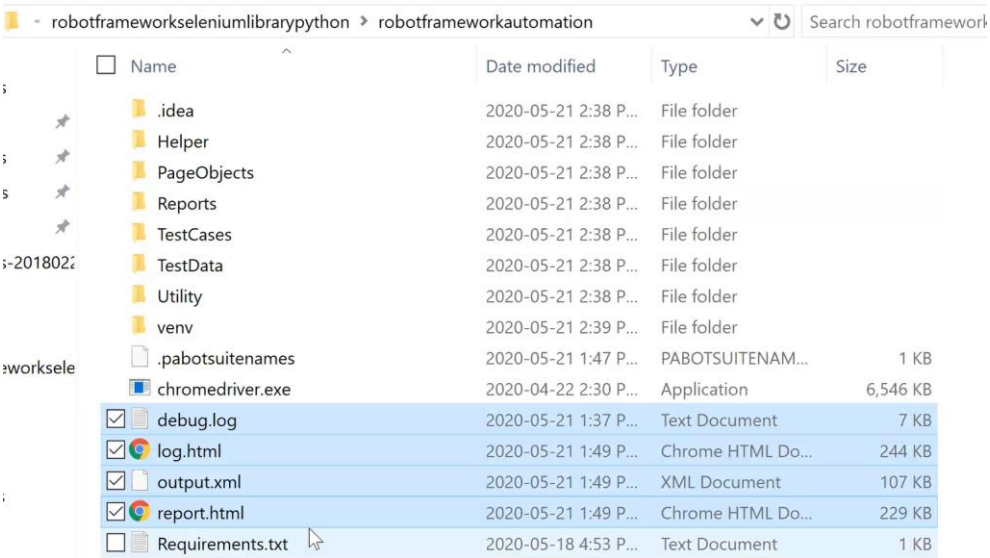


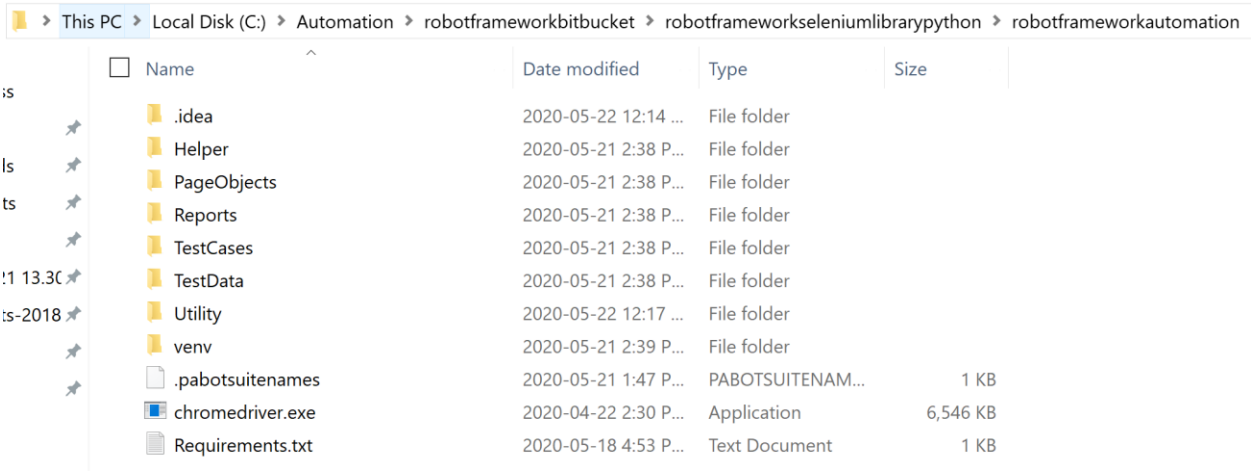
Now we will create .gitignore file here . As you can see I am ignoring Reports folder which has html and other screenshots etc and venv .





Go into robotframeworkautomation folder . I had some extra test reports etc outside the Reports folder I deleted them





Lets go back to Git Command line. We will need to navigate into the folder in which we copied the code

```
user@DESKTOP-5N2O2MN MINGW64 /c/Automation/robotframeworkbitbucket
$ cd robotframeworkseleniumlibrarypython/
```

You will see you are in master .

```
user@DESKTOP-5N2O2MN MINGW64 /c/Automation/robotframeworkbitbucket
$ cd robotframeworkseleniumlibrarypython/

user@DESKTOP-5N2O2MN MINGW64 /c/Automation/robotframeworkbitbucket/robotframeworkseleniumlibrarypython (master)
$
```

Now run git status command. You will see two items in red meaning they aren't yet pushed to remote repository

```
MINGW64:/c/Automation/robotframeworkbitbucket/robotframeworkseleniumlibrarypython

user@DESKTOP-5N2O2MN MINGW64 /c/Automation/robotframeworkbitbucket/robotframeworkseleniumlibrarypython (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        .gitignore
        robotframeworkautomation/

nothing added to commit but untracked files present (use "git add" to track)

user@DESKTOP-5N2O2MN MINGW64 /c/Automation/robotframeworkbitbucket/robotframeworkseleniumlibrarypython (master)
$
```

First we will push .gitignore . Anytime we push the code its three step process first we add and then we commit with message what we are commting meaning our changes and last step is push

git add .gitignore

```
user@DESKTOP-5N2O2MN MINGW64 /c/Automation/robotframeworkbitbucket/robotframeworkseleniumlibrarypython (master)
$ git add .gitignore
```

Next we will commit

git commit -m “Added gitignore file”

```
user@DESKTOP-5N2O2MN MINGW64 /c/Automation/robotframeworkbitbucket/robotframeworkseleniumlibrarypython (master)
$ git commit -m "Added gitIgnore file"
[master (root-commit) 44e59c4] Added gitIgnore file
 1 file changed, 2 insertions(+)
 create mode 100644 .gitignore
```

Next we will push it

git push origin master

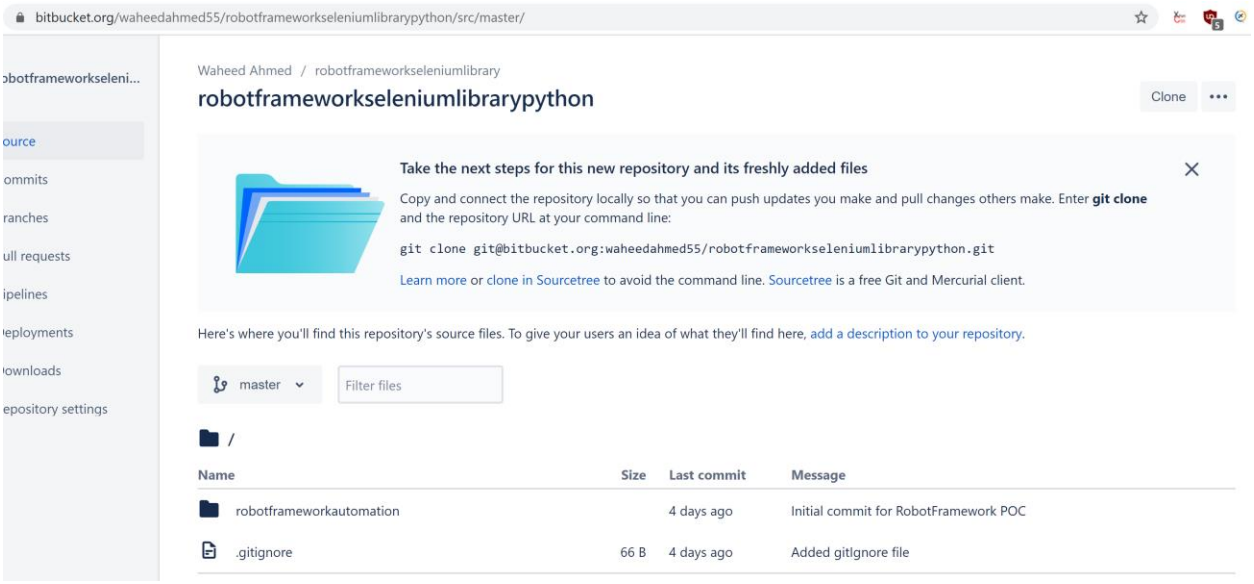
```
user@DESKTOP-5N2O2MN MINGW64 /c/Automation/robotframeworkbitbucket/robotframeworkseleniumlibrarypython (master)
$ git push origin master
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 268 bytes | 268.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://bitbucket.org/waheedahmed55/robotframeworkseleniumlibrarypython.git
 * [new branch]      master -> master
```

Now we will repeat these three steps for the other folder robotframeworkautomation/

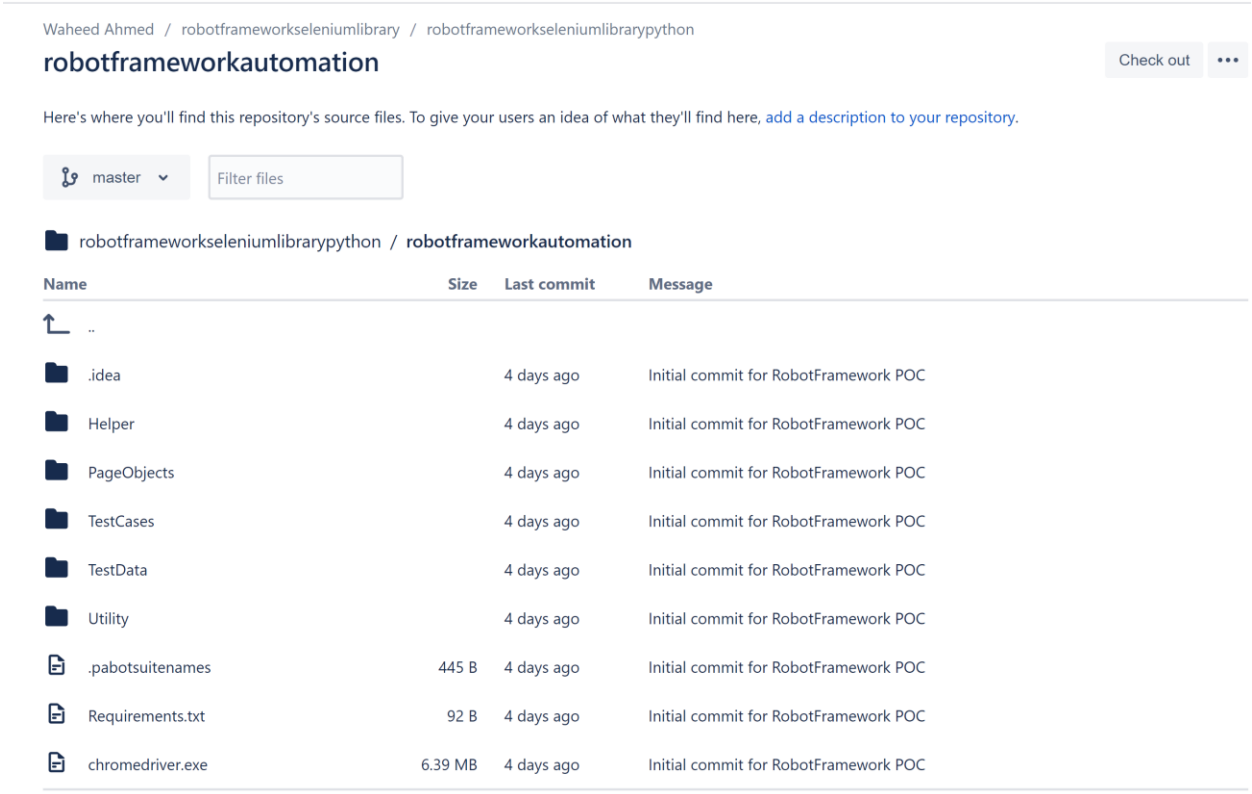
```
user@DESKTOP-5N2O2MN MINGW64 /c/Automation/robotframeworkbitbucket/robotframeworkseleniumlibrarypython (master)
$ git add robotframeworkautomation/
warning: LF will be replaced by CRLF in robotframeworkautomation/.idea/inspectionProfiles/profiles_settings.xml.
The file will have its original line endings in your working directory

user@DESKTOP-5N2O2MN MINGW64 /c/Automation/robotframeworkbitbucket/robotframeworkseleniumlibrarypython (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   robotframeworkautomation/.idea/.gitignore
        new file:   robotframeworkautomation/.idea/inspectionProfiles/profiles_settings.xml
        new file:   robotframeworkautomation/.idea/misc.xml
        new file:   robotframeworkautomation/.idea/modules.xml
        new file:   robotframeworkautomation/.idea/robotframeworkautomation.iml
        new file:   robotframeworkautomation/.pabotsuitenames
        new file:   robotframeworkautomation/Helper/FormHelper.robot
        new file:   robotframeworkautomation/Helper/HandleHelper.robot
        new file:   robotframeworkautomation/Helper/HandleWindowHelper.robot
        new file:   robotframeworkautomation/Helper/HomePageHelper.robot
        new file:   robotframeworkautomation/Helper/LoginHelper.robot
        new file:   robotframeworkautomation/Helper/TableHelper.robot
        new file:   robotframeworkautomation/PageObjects/FormObjects.robot
        new file:   robotframeworkautomation/PageObjects/HandleObjects.robot
        new file:   robotframeworkautomation/PageObjects/HandleWIndowObjects.robot
        new file:   robotframeworkautomation/PageObjects/HomePageObjects.robot
        new file:   robotframeworkautomation/PageObjects/LoginObjetcs.robot
        new file:   robotframeworkautomation/PageObjects/TableObject.robot
        new file:   robotframeworkautomation/Requirements.txt
        new file:   robotframeworkautomation/TestCases/Functional/DataDriven.robot
        new file:   robotframeworkautomation/TestCases/Functional/FillContactForm.robot
        new file:   robotframeworkautomation/TestCases/Regression/HandleAndFrame.robot
        new file:   robotframeworkautomation/TestCases/Regression/HandleTable.robot
        new file:   robotframeworkautomation/TestCases/Smoke/HandleWindow.robot
        new file:   robotframeworkautomation/TestCases/Smoke/LoginTest.robot
        new file:   robotframeworkautomation/TestCases/Smoke/SearchAddToCart.robot
        new file:   robotframeworkautomation/TestData/ApplicationProperties.robot
        new file:   robotframeworkautomation/TestData/TestSheet.xlsx
        new file:   robotframeworkautomation/Utility/SeleniumKeywords.robot
        new file:   robotframeworkautomation/Utility/Setup.robot
        new file:   robotframeworkautomation/chromedriver.exe
```

```
user@DESKTOP-5N2O2MN MINGW64 /c/Automation/robotframeworkbitbucket/robotframeworkseleniumlibrarypython (master)
$ git commit -m "Initial commit for RobotFramework POC"
[master 827828d] Initial commit for RobotFramework POC
 31 files changed, 708 insertions(+)
 create mode 100644 robotframeworkautomation/.idea/.gitignore
 create mode 100644 robotframeworkautomation/.idea/inspectionProfiles/profiles_settings.xml
 create mode 100644 robotframeworkautomation/.idea/misc.xml
 create mode 100644 robotframeworkautomation/.idea/modules.xml
 create mode 100644 robotframeworkautomation/.idea/robotframeworkautomation.iml
 create mode 100644 robotframeworkautomation/.pabotsuitenames
 create mode 100644 robotframeworkautomation/Helper/FormHelper.robot
 create mode 100644 robotframeworkautomation/Helper/HandleHelper.robot
 create mode 100644 robotframeworkautomation/Helper/HandleWindowHelper.robot
 create mode 100644 robotframeworkautomation/Helper/HomePageHelper.robot
 create mode 100644 robotframeworkautomation/Helper/LoginHelper.robot
 create mode 100644 robotframeworkautomation/Helper/TableHelper.robot
 create mode 100644 robotframeworkautomation/PageObjects/FormObjects.robot
 create mode 100644 robotframeworkautomation/PageObjects/HandleObjects.robot
 create mode 100644 robotframeworkautomation/PageObjects/HandleWIndowObjects.robot
 create mode 100644 robotframeworkautomation/PageObjects/HomePageObjects.robot
 create mode 100644 robotframeworkautomation/PageObjects/LoginObjetcs.robot
 create mode 100644 robotframeworkautomation/PageObjects/TableObject.robot
 create mode 100644 robotframeworkautomation/Requirements.txt
 create mode 100644 robotframeworkautomation/TestCases/Functional/DataDriven.robot
 create mode 100644 robotframeworkautomation/TestCases/Functional/FillContactForm.robot
 create mode 100644 robotframeworkautomation/TestCases/Regression/HandleAndFrame.robot
 create mode 100644 robotframeworkautomation/TestCases/Regression/HandleTable.robot
 create mode 100644 robotframeworkautomation/TestCases/Smoke/HandleWindow.robot
 create mode 100644 robotframeworkautomation/TestCases/Smoke/LoginTest.robot
 create mode 100644 robotframeworkautomation/TestCases/Smoke/SearchAddToCart.robot
 create mode 100644 robotframeworkautomation/TestData/ApplicationProperties.robot
 create mode 100644 robotframeworkautomation/TestData/TestSheet.xlsx
 create mode 100644 robotframeworkautomation/Utility/SeleniumKeywords.robot
 create mode 100644 robotframeworkautomation/Utility/Setup.robot
 create mode 100644 robotframeworkautomation/chromedriver.exe

user@DESKTOP-5N2O2MN MINGW64 /c/Automation/robotframeworkbitbucket/robotframeworkseleniumlibrarypython (master)
$ git push origin master
Enumerating objects: 45, done.
Counting objects: 100% (45/45), done.
Delta compression using up to 4 threads
Compressing objects: 100% (42/42), done.
Writing objects: 100% (44/44), 3.44 MiB | 1.11 MiB/s, done.
Total 44 (delta 3), reused 0 (delta 0)
To https://bitbucket.org/waheedahmed55/robotframeworkseleniumlibrarypython.git
   44e59c4..827828d  master -> master
```

Now if you go back to bitbucket you will see your project:

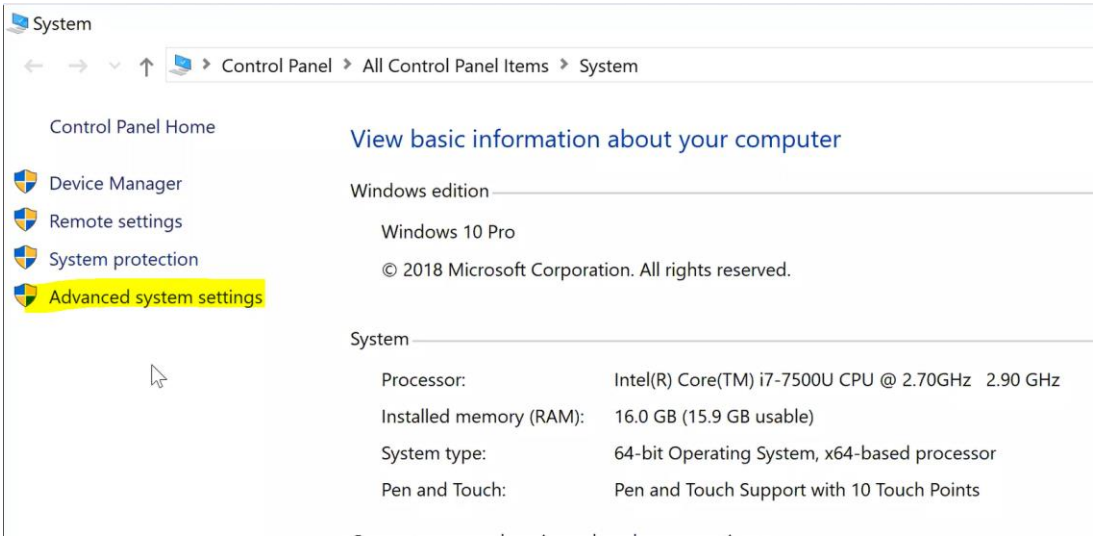


Click on robotautomationframework folder, you will see the code which we just pushed

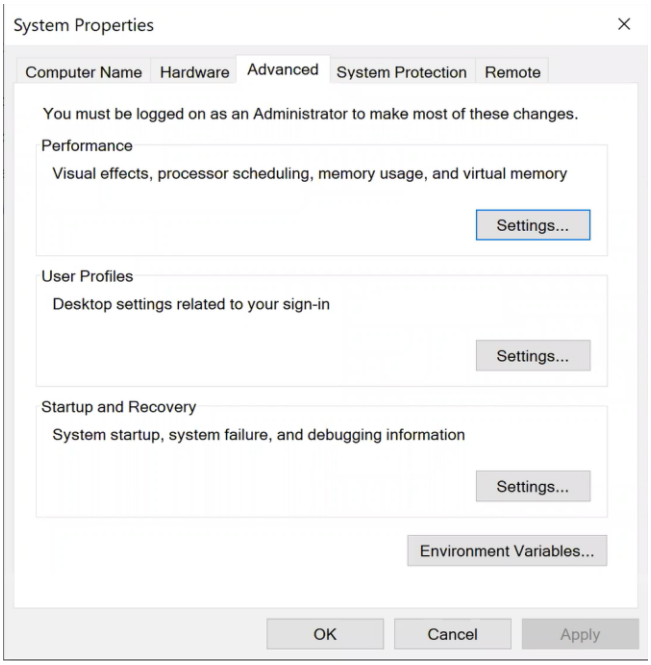


Before we proceed to next steps we will add pip in our PATH under environment variable.

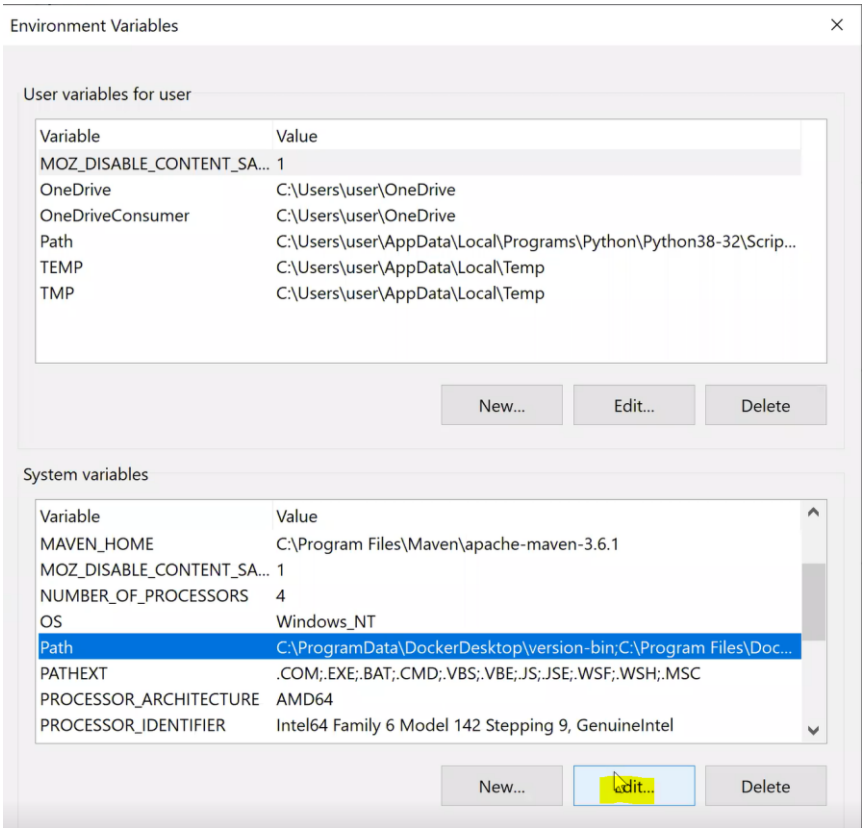
Click on Advance system settings



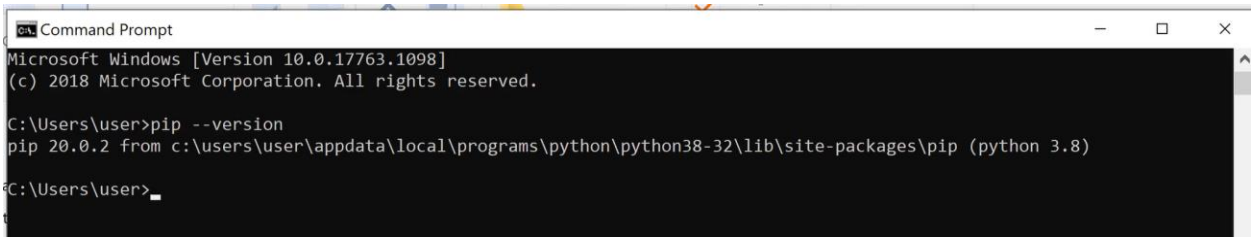
Click on Environment Variables



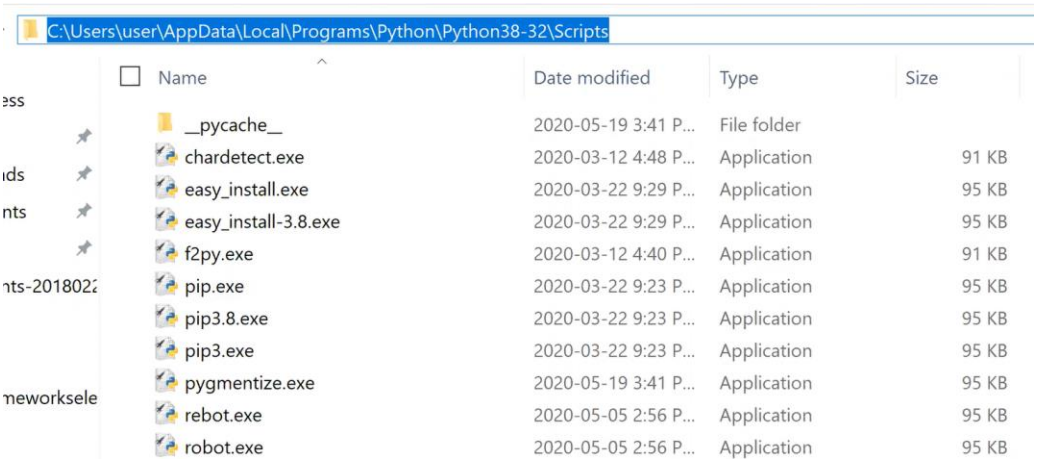
Select Path and click Edit



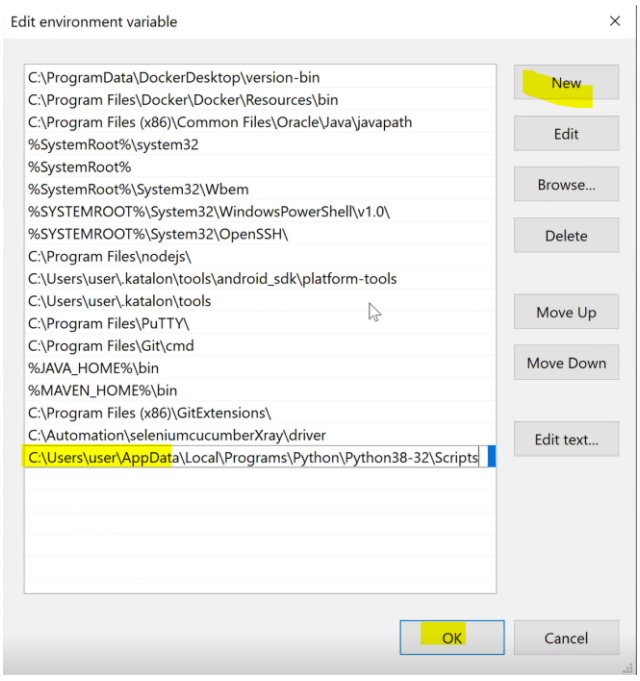
In order to find path to pip



Copy the path



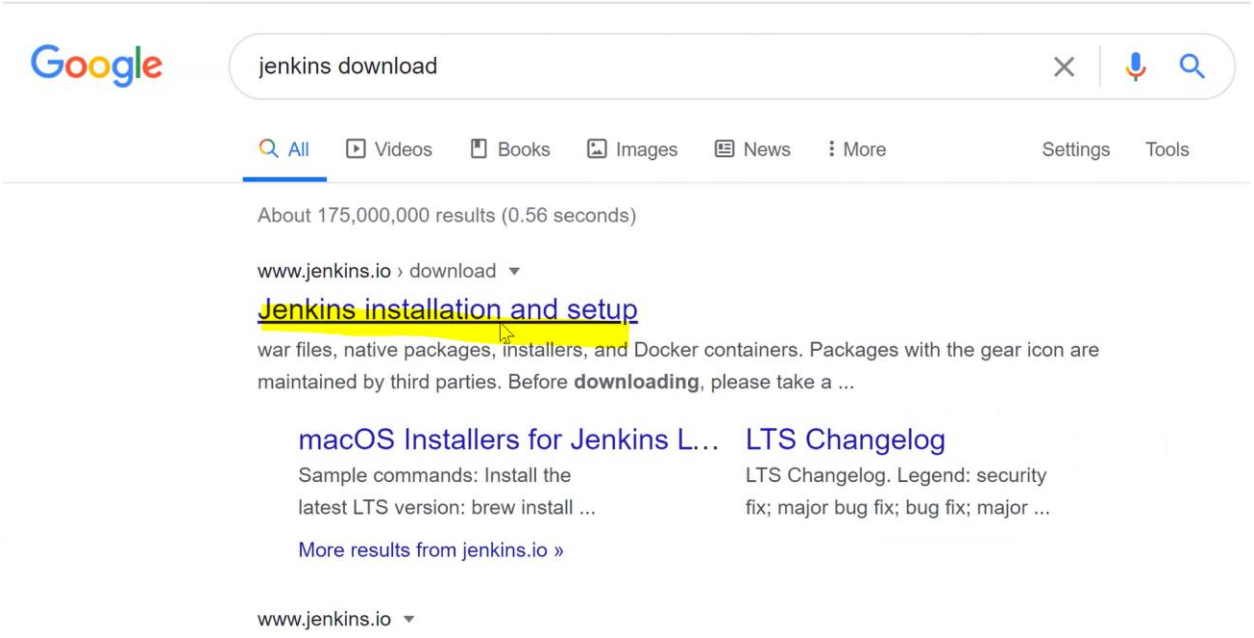
Go back to PATH . Click on New paste the pip path and click on OK



Then click Ok . Then click on cancel

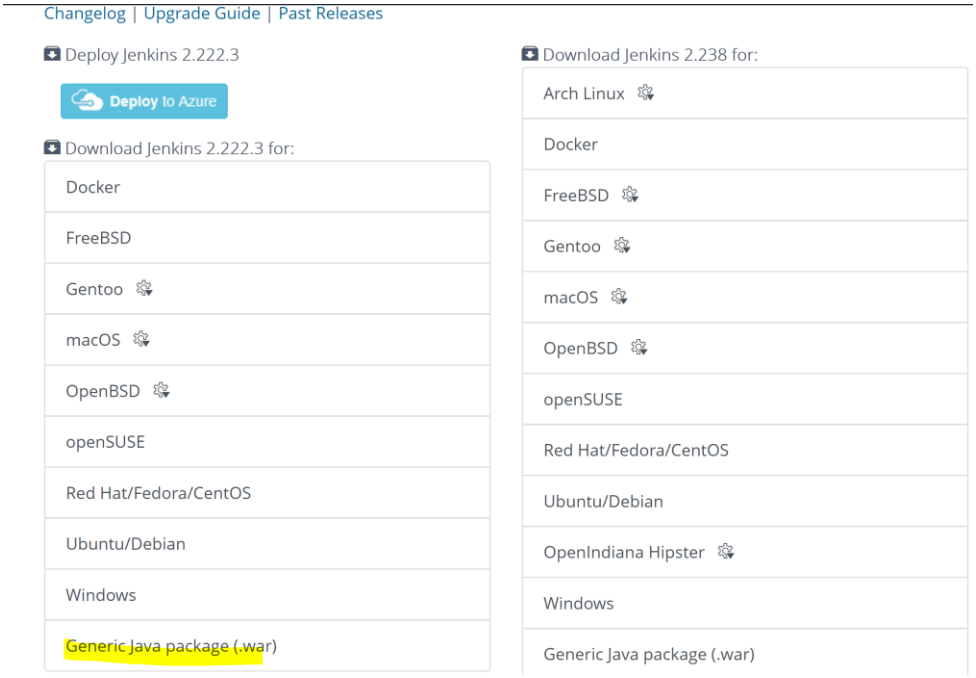
Jenkins: Install & Setup

Go to google & search for Jenkins download



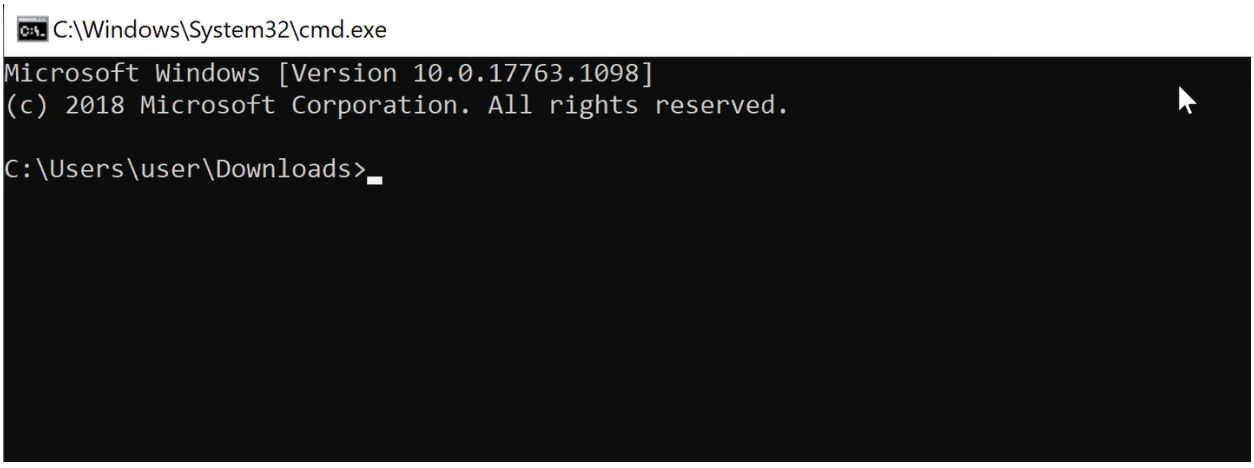
Or you can visit directly https://www.jenkins.io/download/

Scroll down select Generic Java package (.war)



It will start downloading

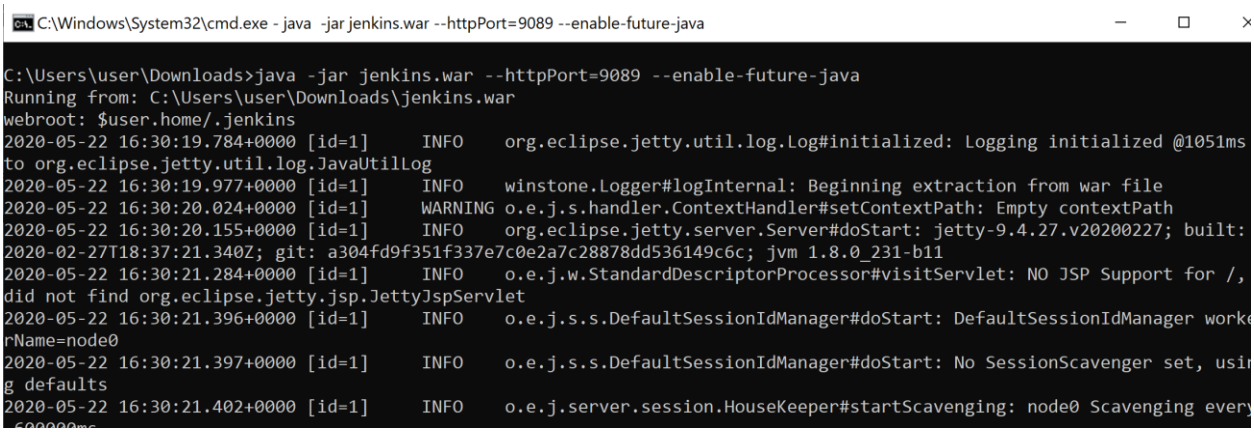
In my case I downloaded into in downloads folder. I launched the command prompt and went to downloads directory



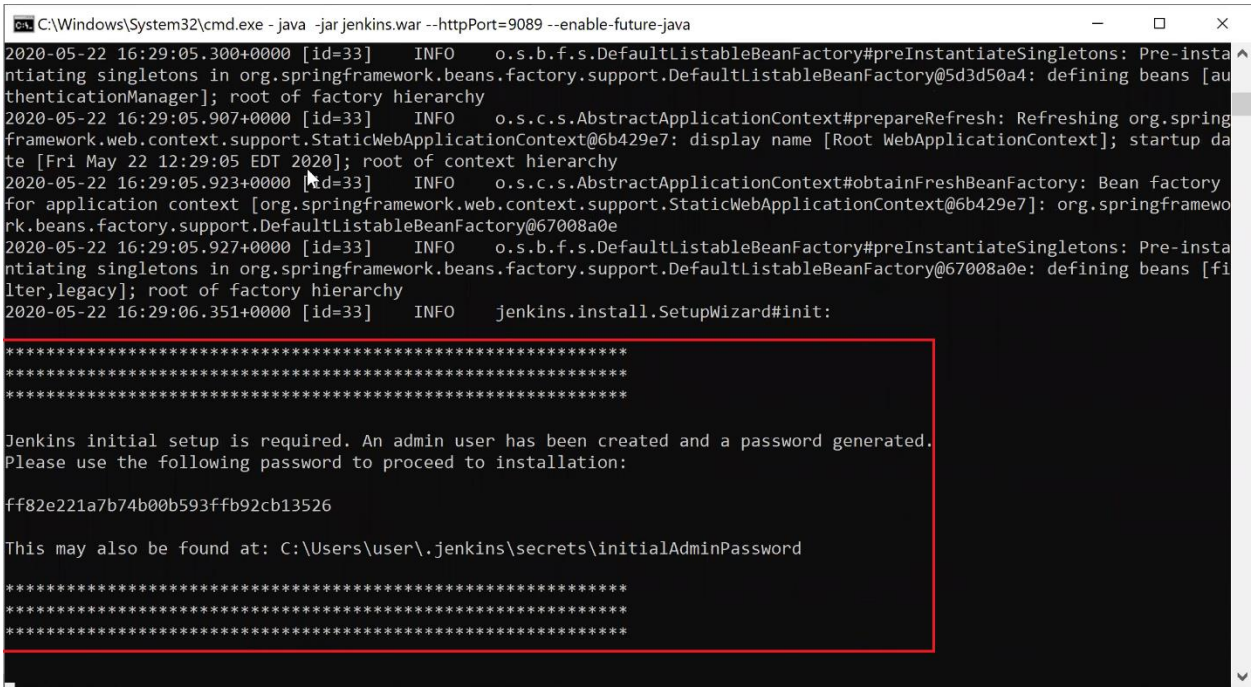
We will launch jenkisn using this war file , unpack using java -jar and launching at port 9089

java -jar jenkins.war --httpPort=9089 --enable-future-java

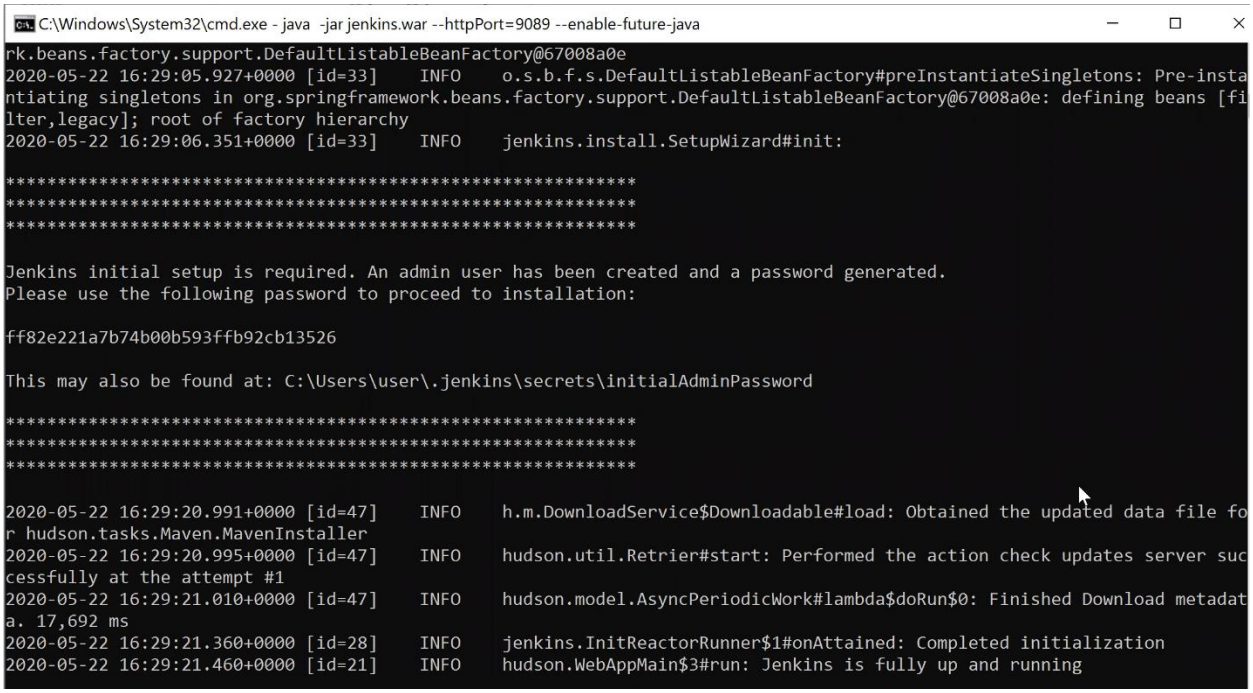
It will start installation



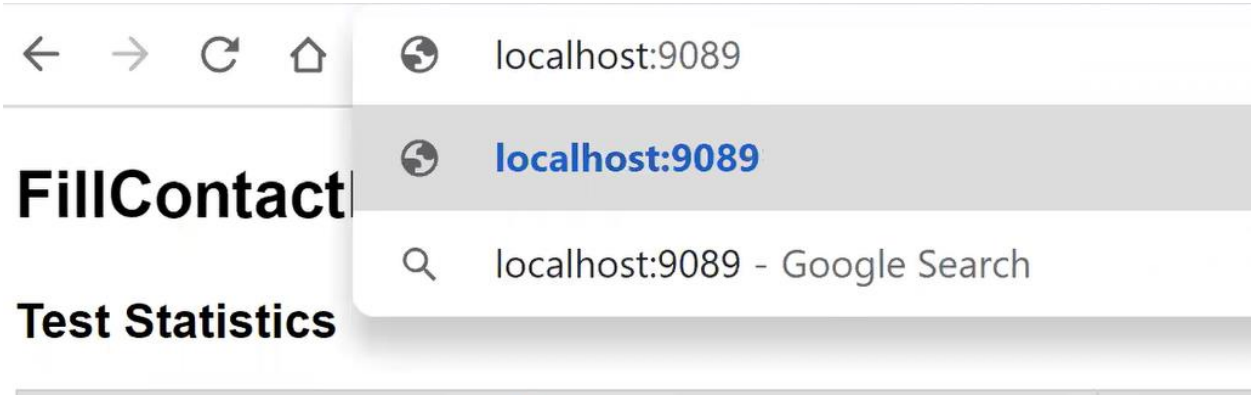
You will see it display location for password that will be needed



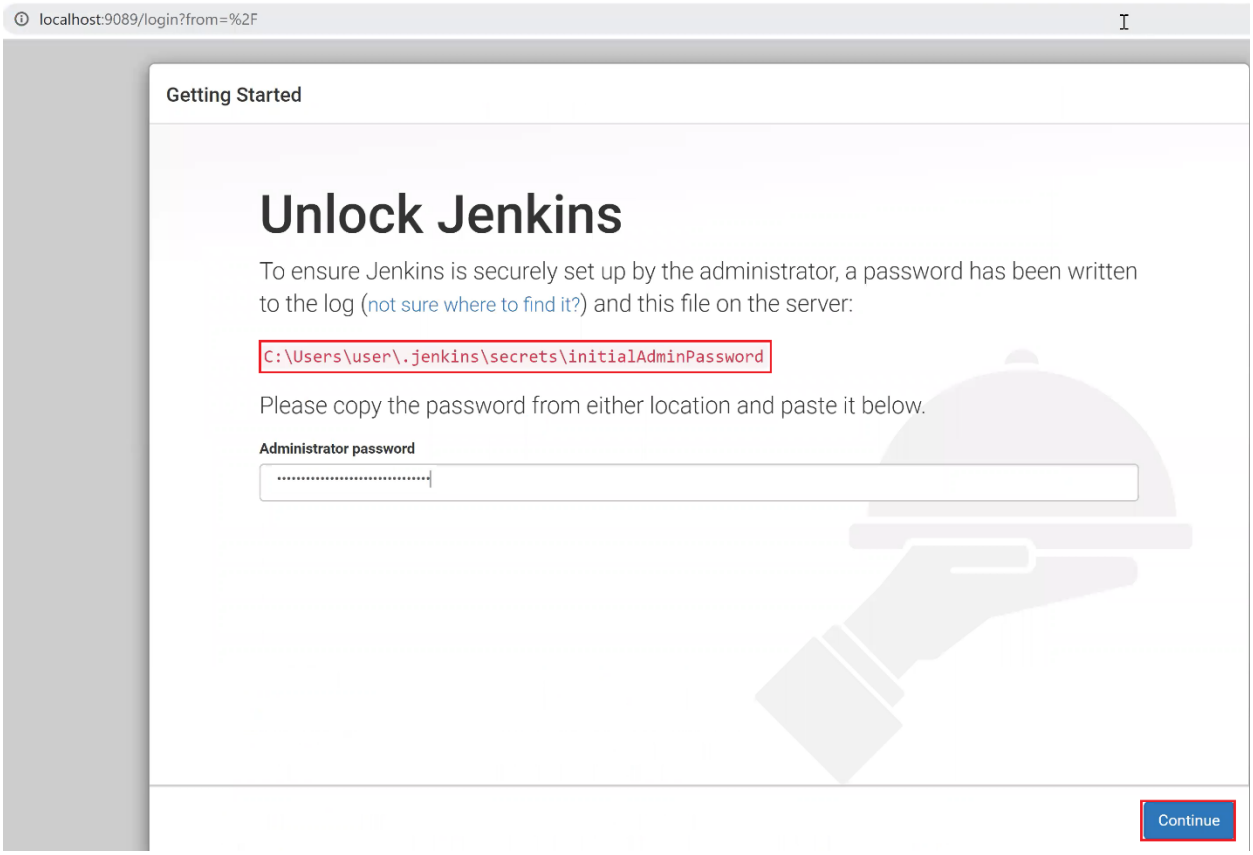
You will see it display Jenkins is fully up and running



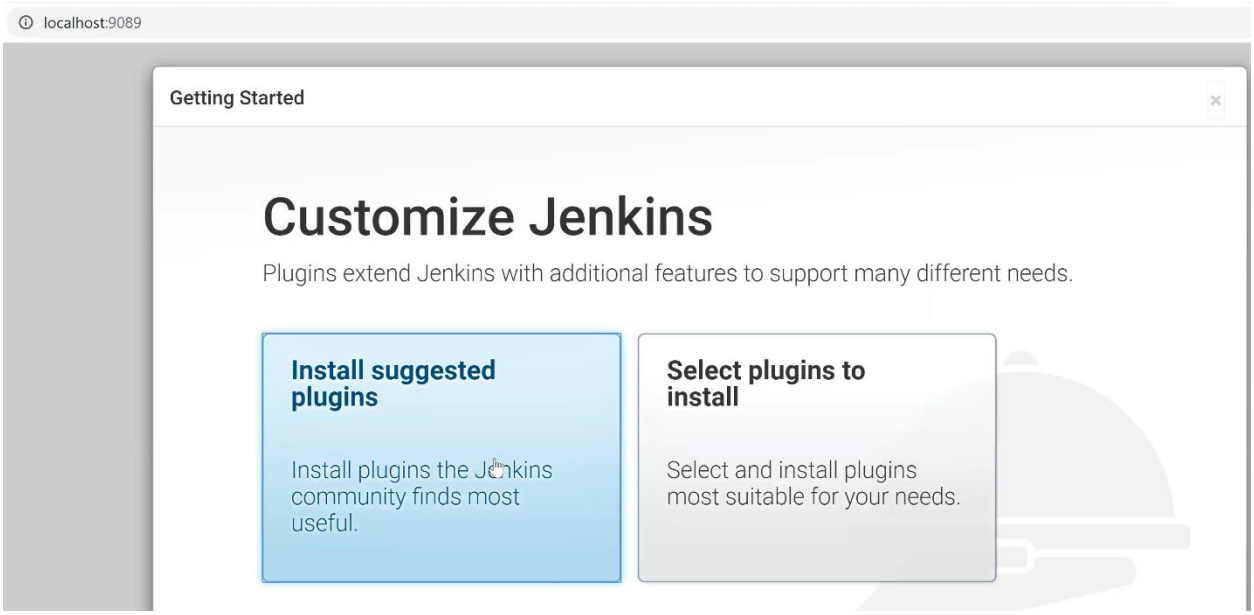
Go to browser navigate to localhost:9089



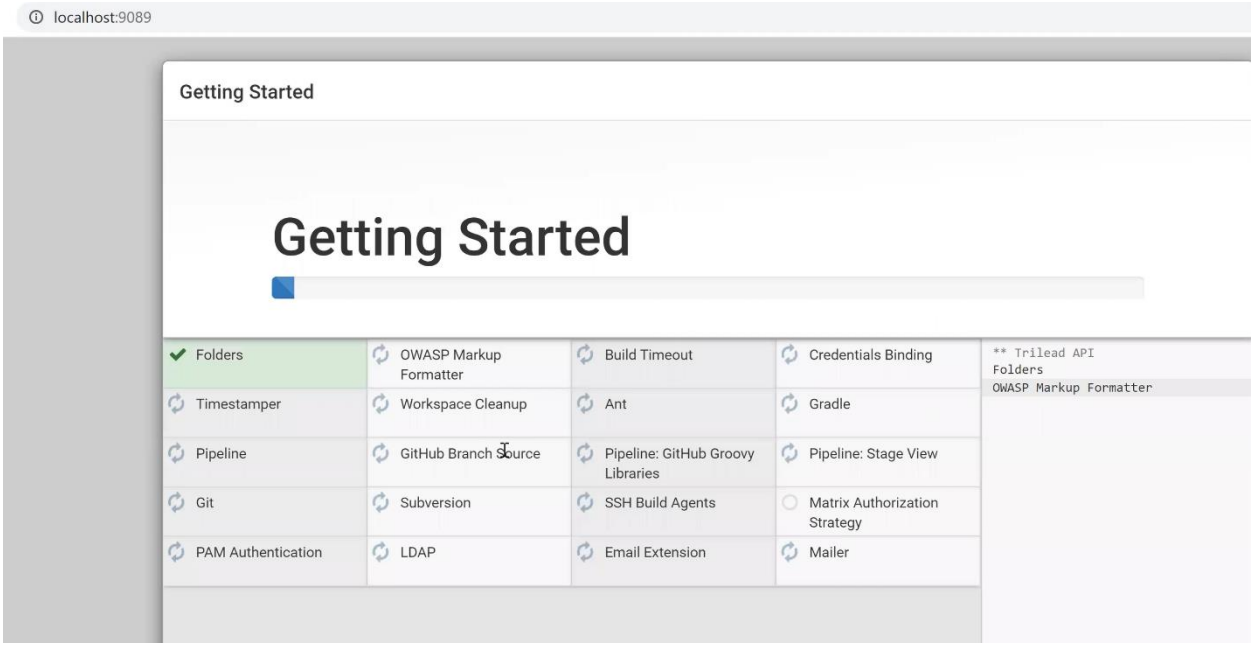
Go to location and copy the Admin password and paste here. Click on Continue



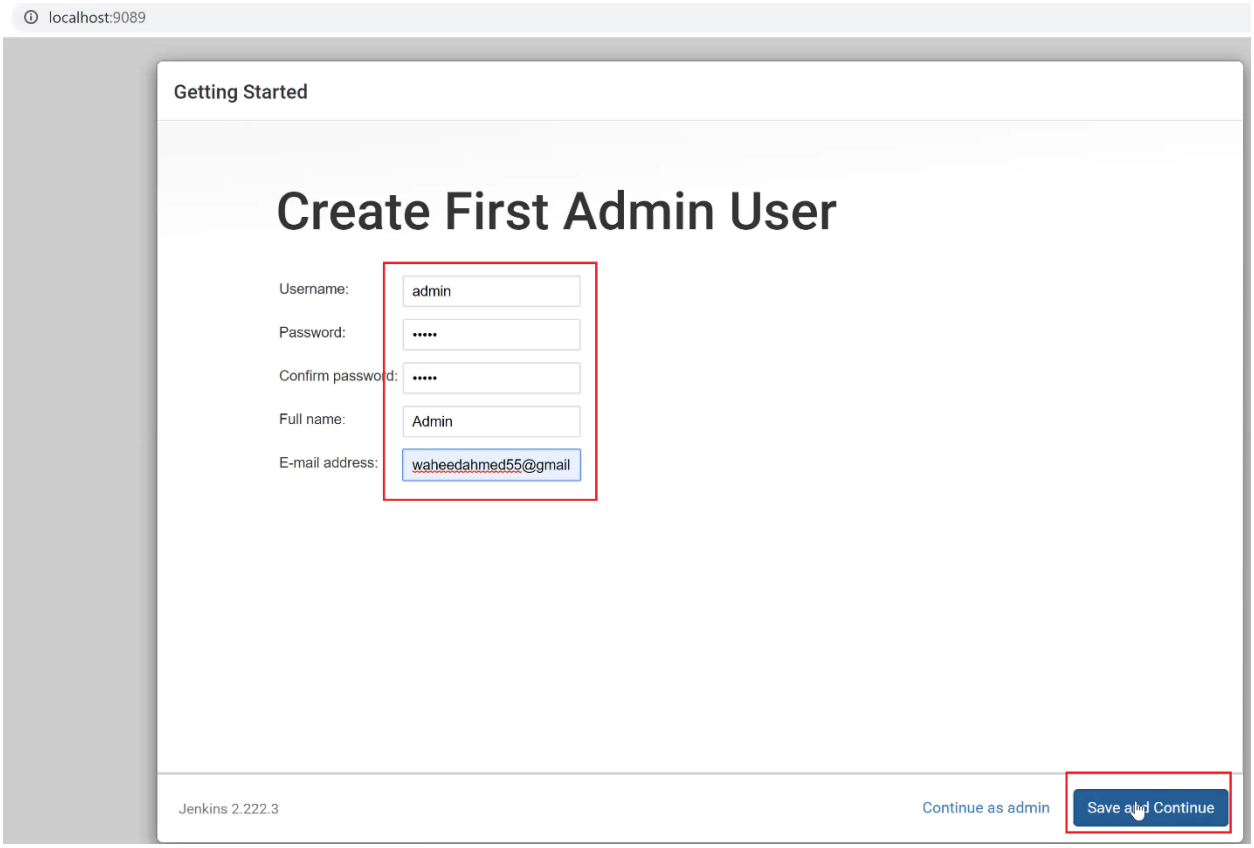
Click on Install suggested plugins



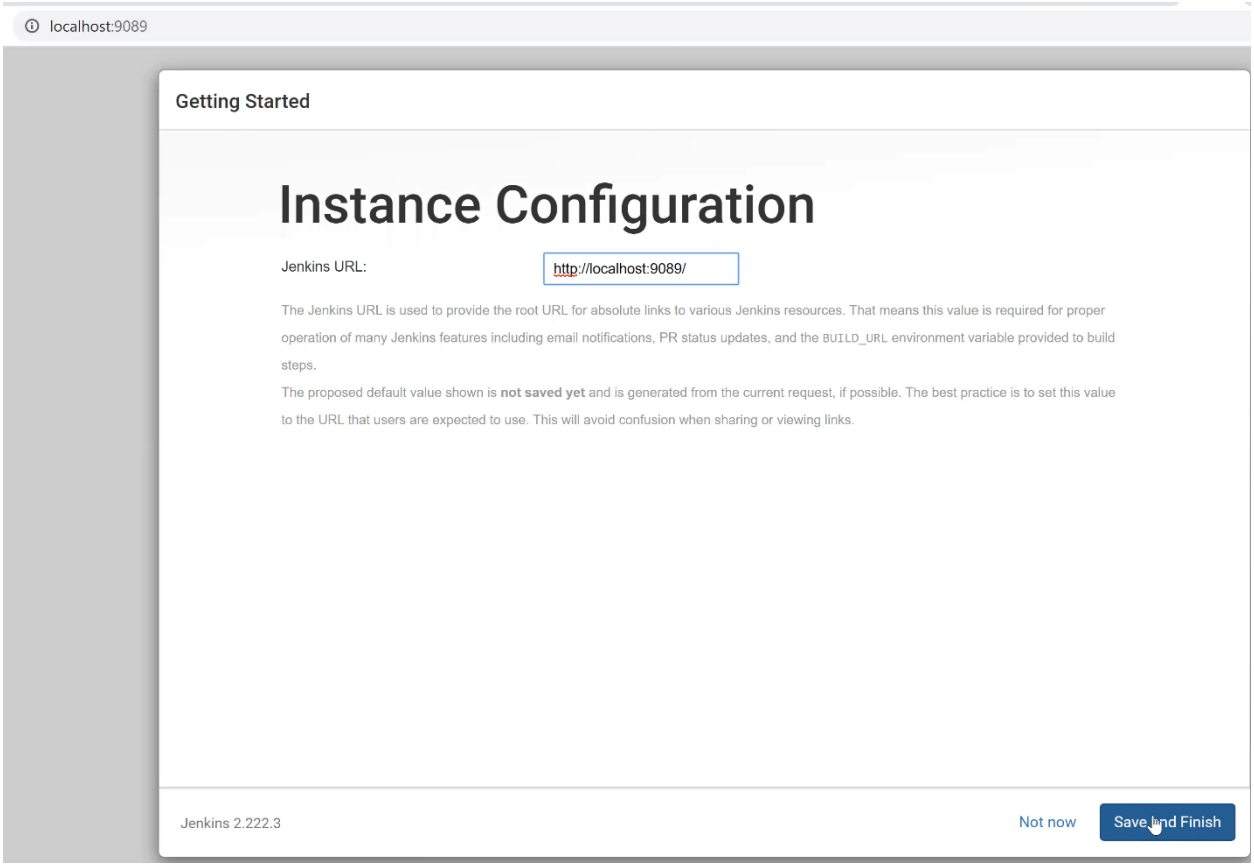
Then it will start installing standard plugins



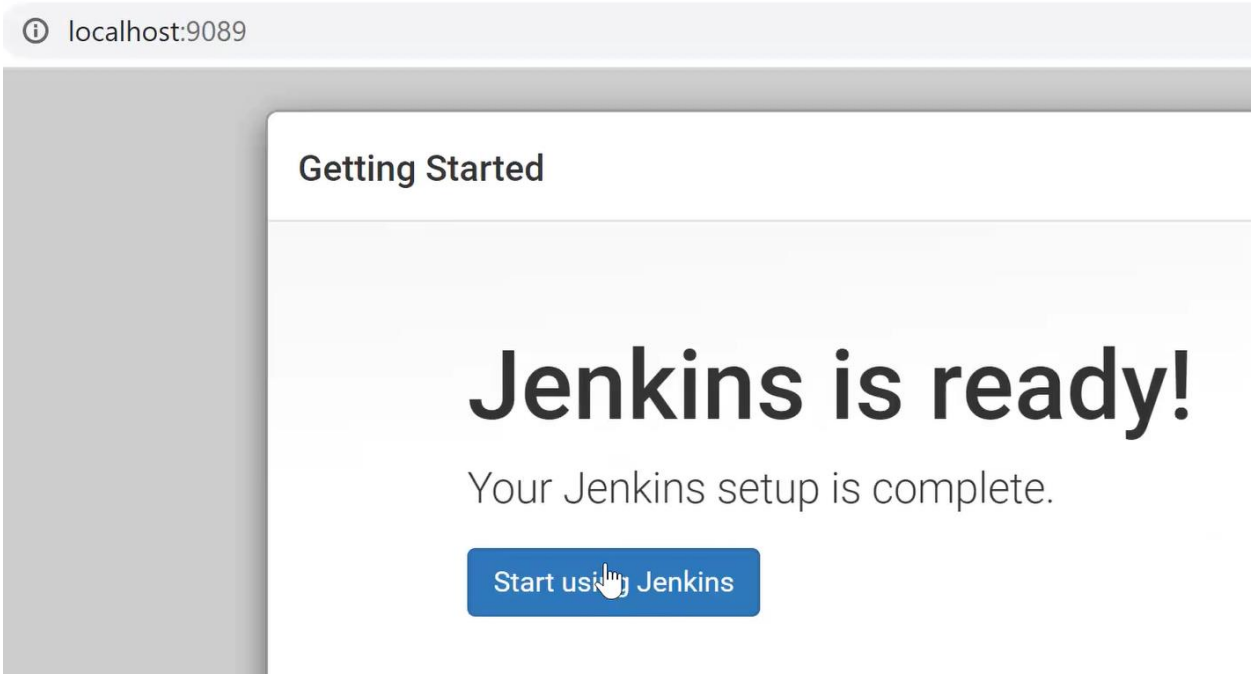
Now set your credentials which you will use to login in to . Click on Save and Continue



Click on Save and Finish

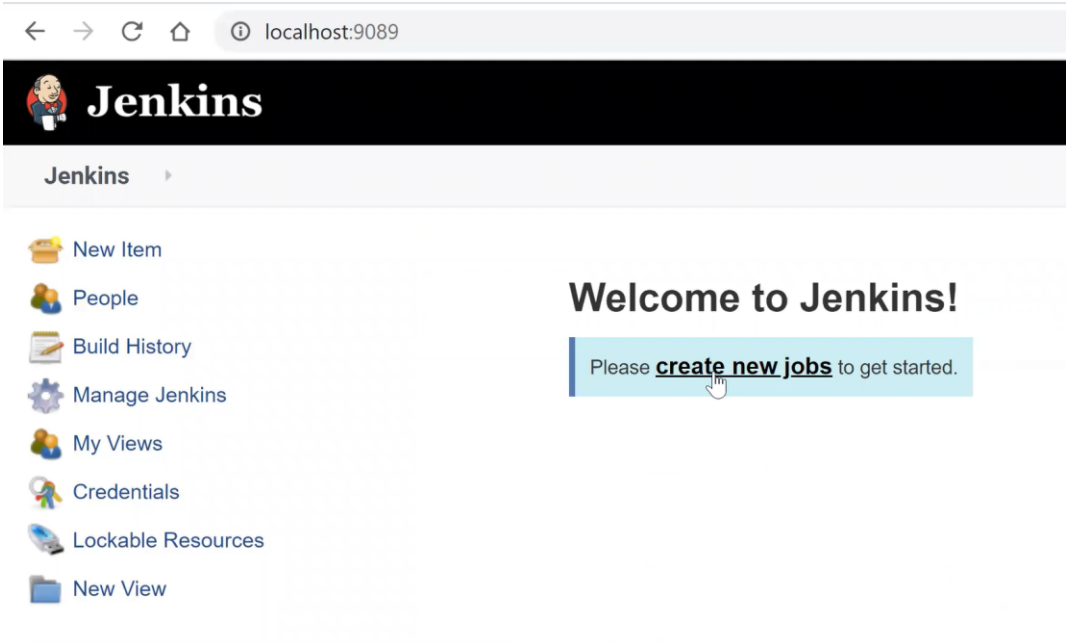


Click on Start using Jenkins

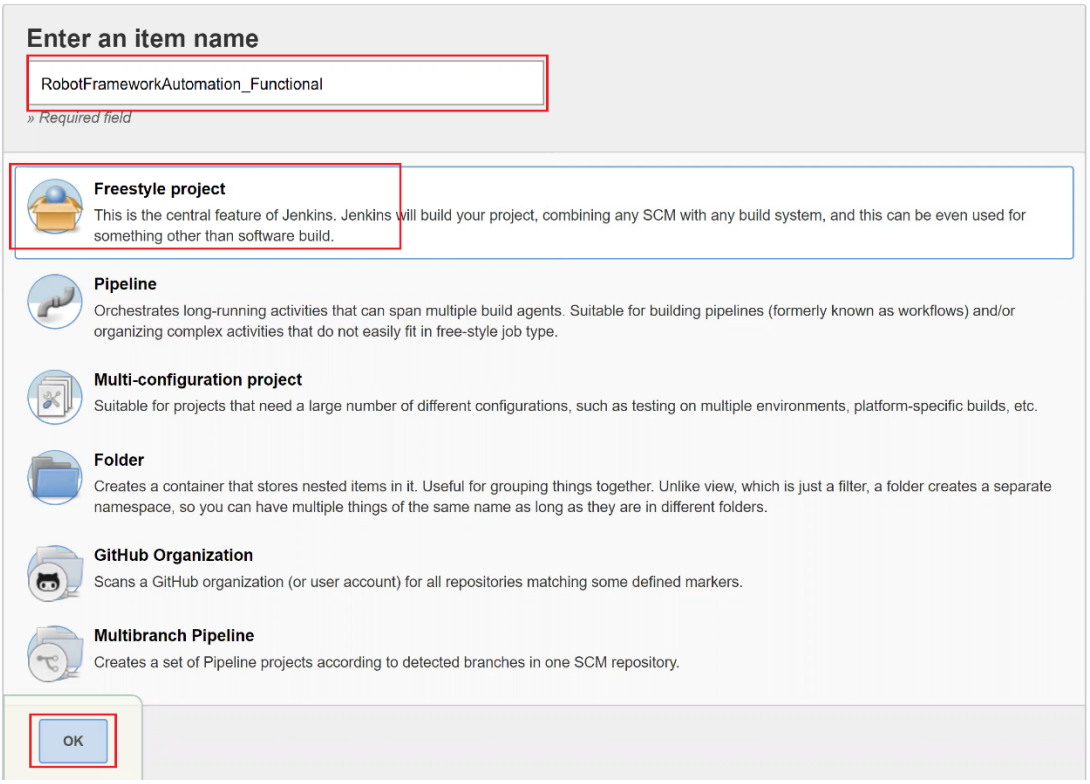


Create Build Jobs in Jenkins:

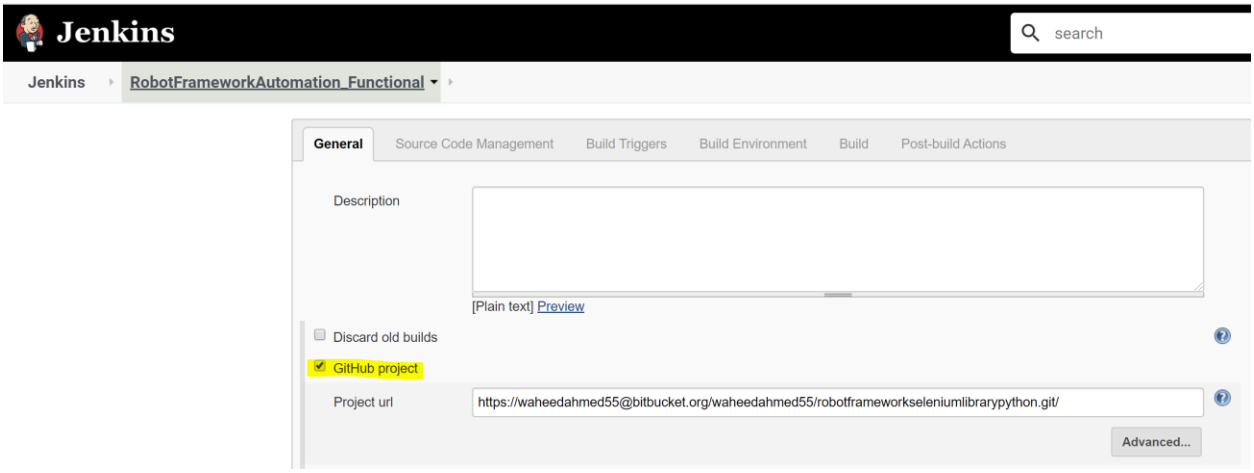
You will see Jenkins home page . We will create two build jobs once for Smoke and other for Functional test suite. Click on create new jobs



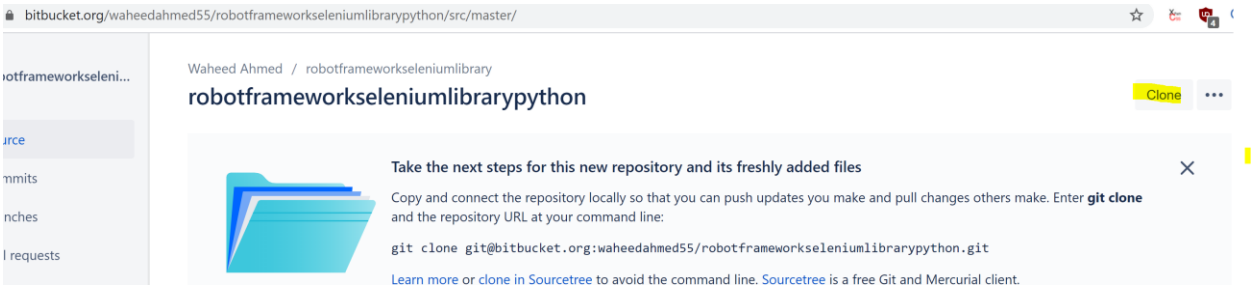
Set build job name and select Freestyle project and Click ok



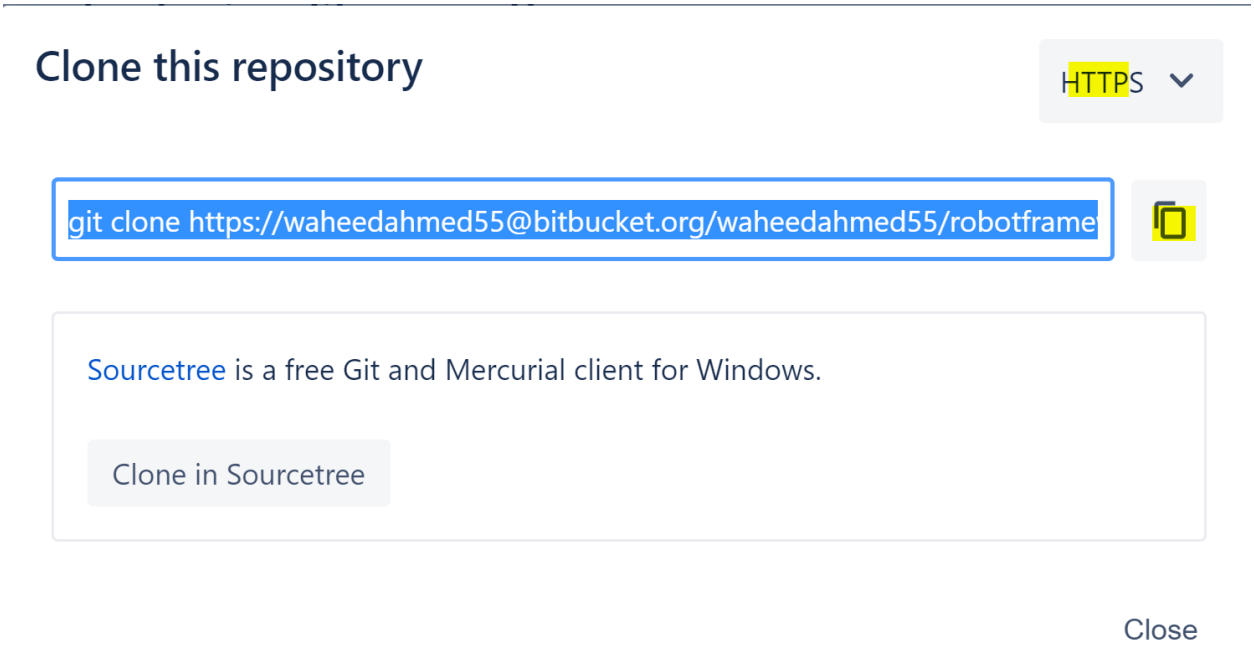
As you click OK it will take you to build configuration page. Select Github project and paste repository URL you can find this from Bitbucket



In Bitbucket click on Clone

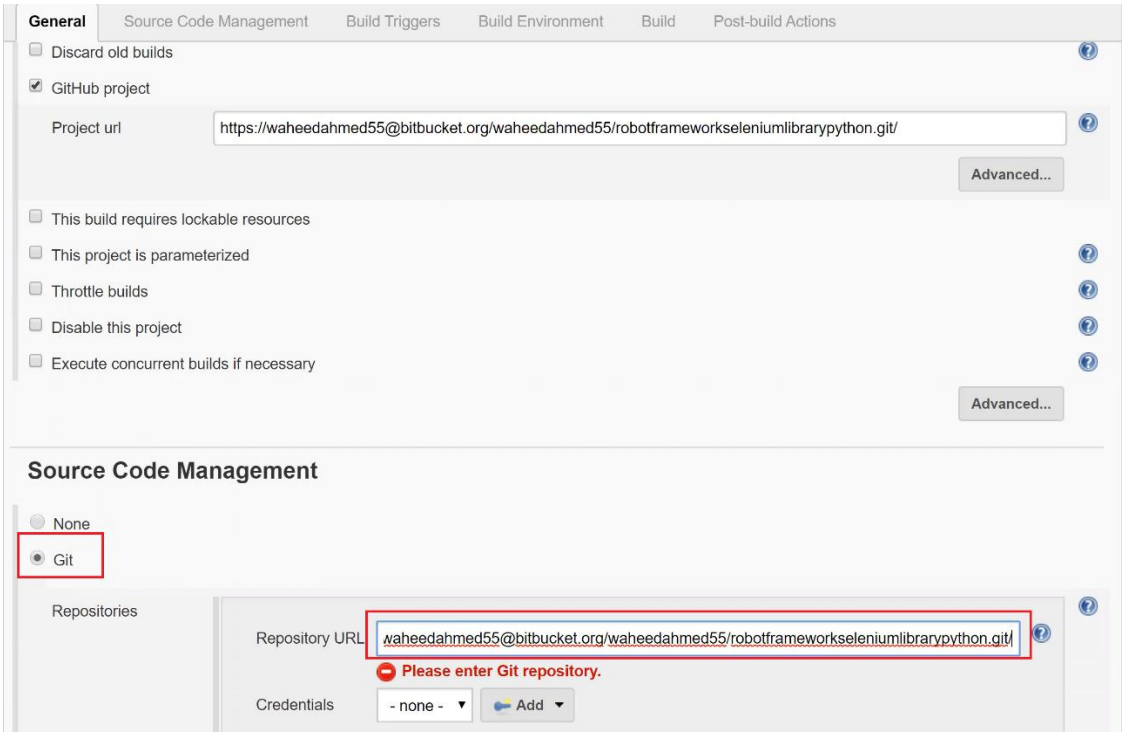


Set to https and copy the url

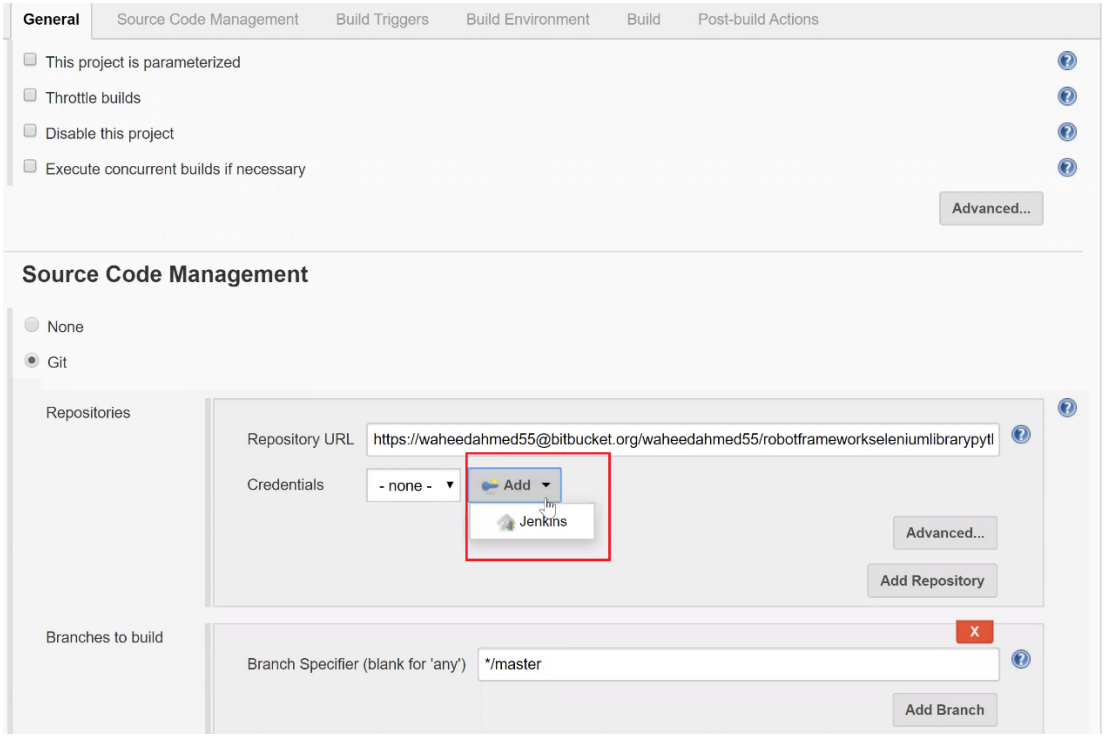


When setting in Jenkins remove git clone part of it.

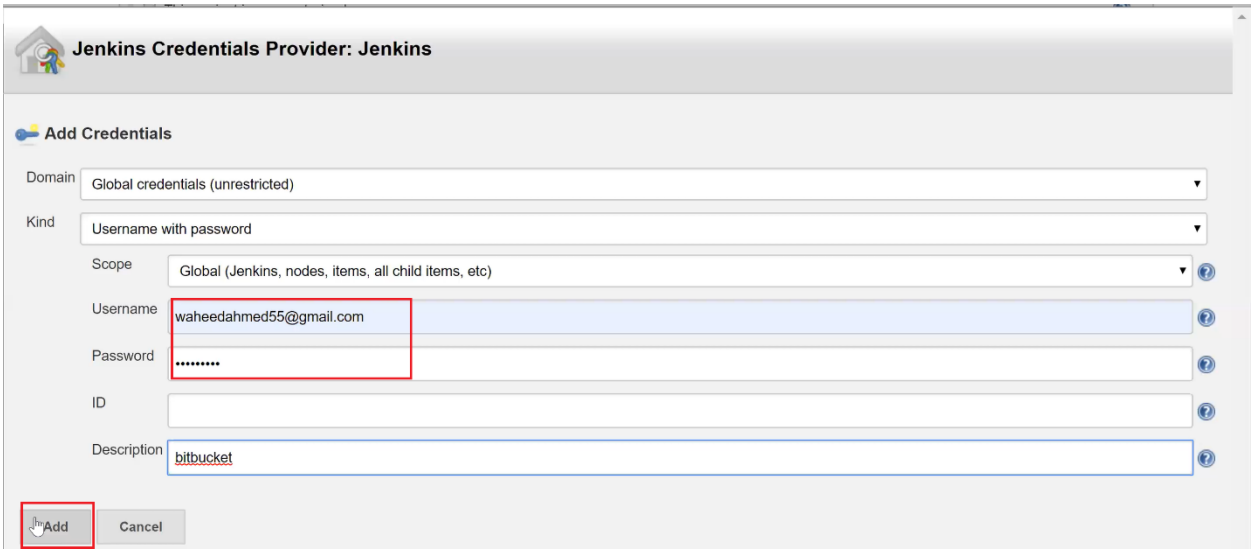
Scroll down to Source Code Management section Select Git and paste the URL



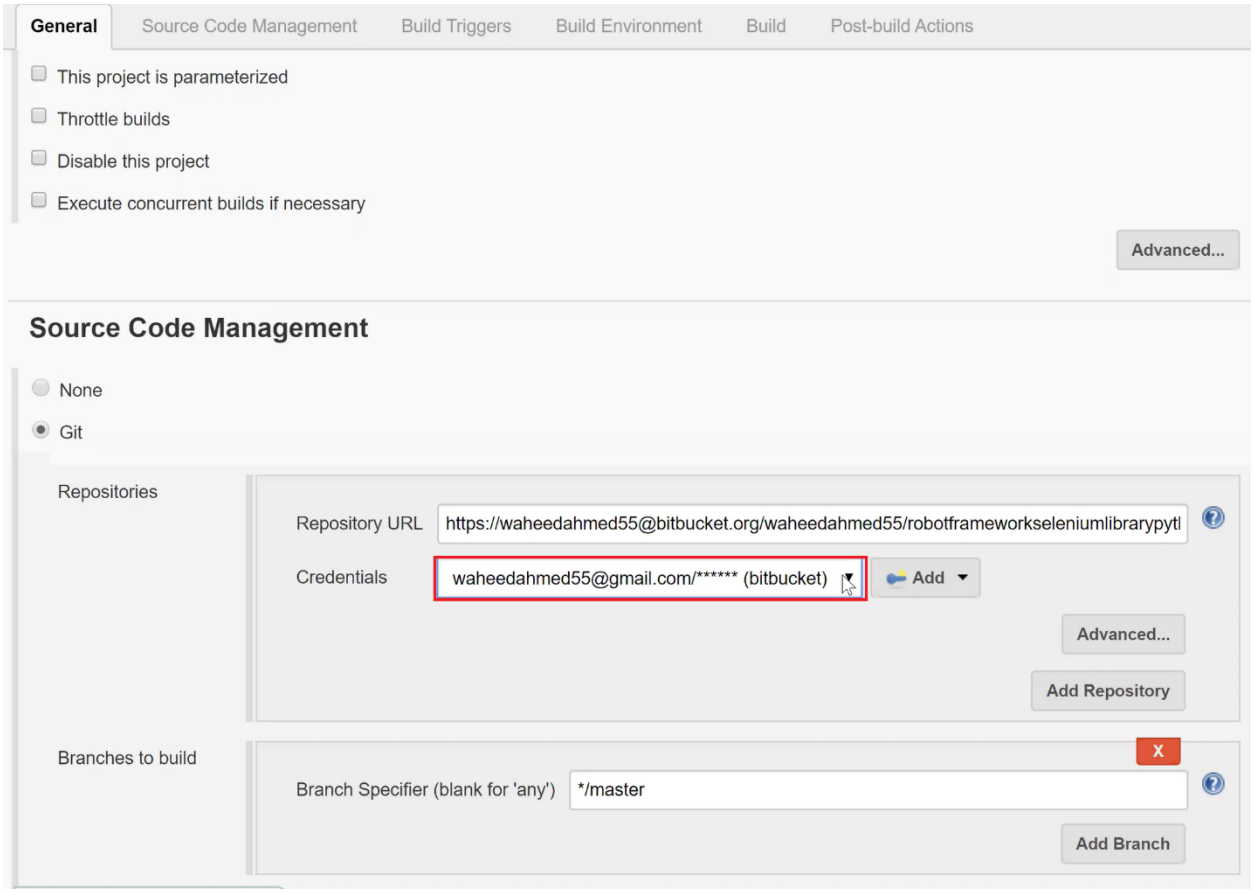
Click on Add -> Select Jenkins . We will be adding credentials of bitbucket here so to allow Jenkins to access the repository



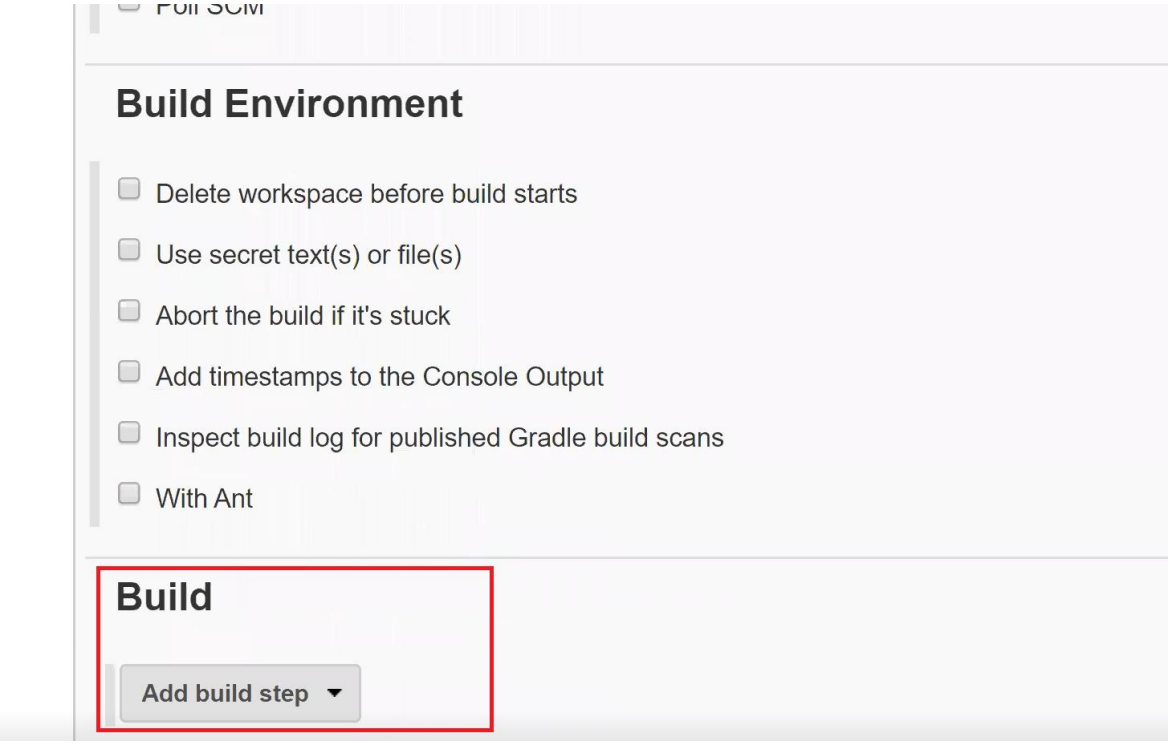
Enter bitbucket username and password and description and Click on Add



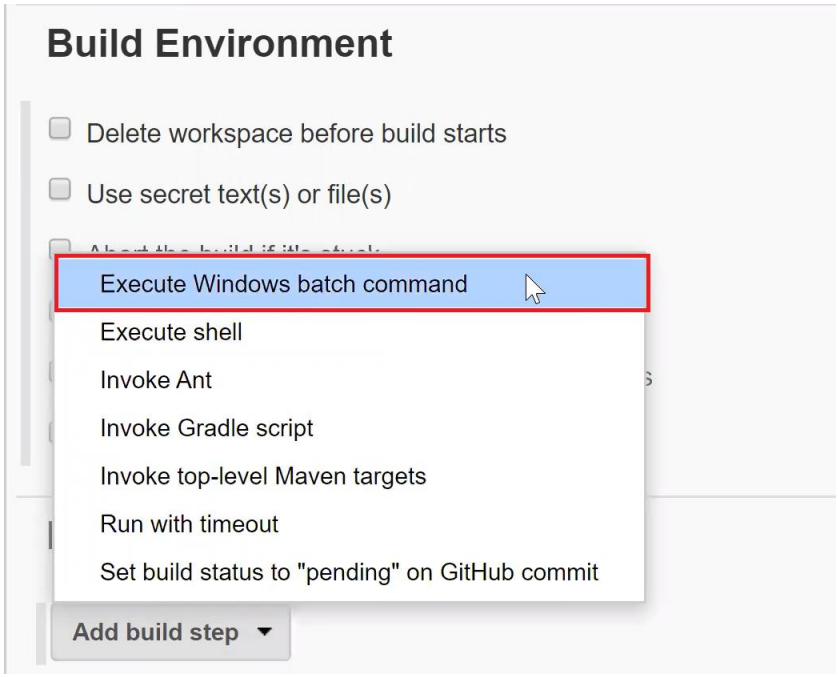
Select the credentials we just created



Scroll down to Build section. Click on Add build step



Select Execute Windows batch command



In Command section add the follow commands. What we are doing in this is first we run dir command once we run the build when this will execute just like on windows it will show list of files and directory

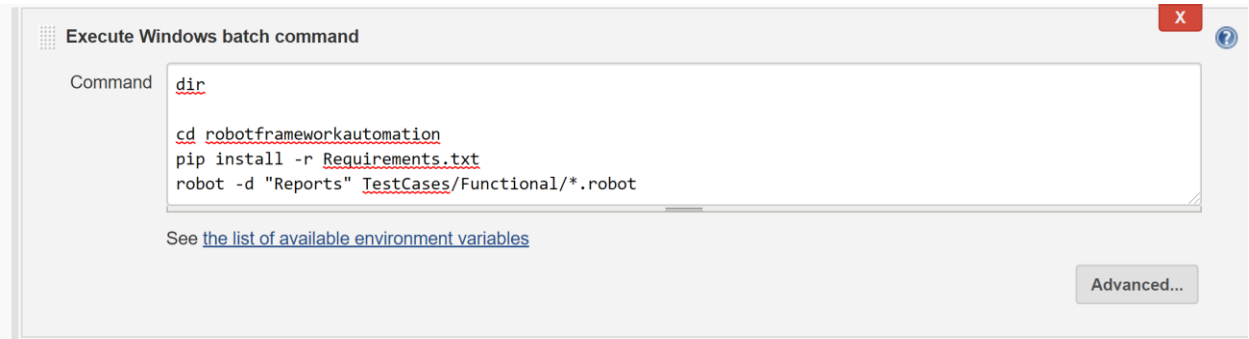
Then we navigate to the project folder . Next we will install all libraries we would need to run our framework which I have listed in Requirments.txtLast but not least command we use to run test cases , in this we will run all functional tests.

dir

cd robotframeworkautomation

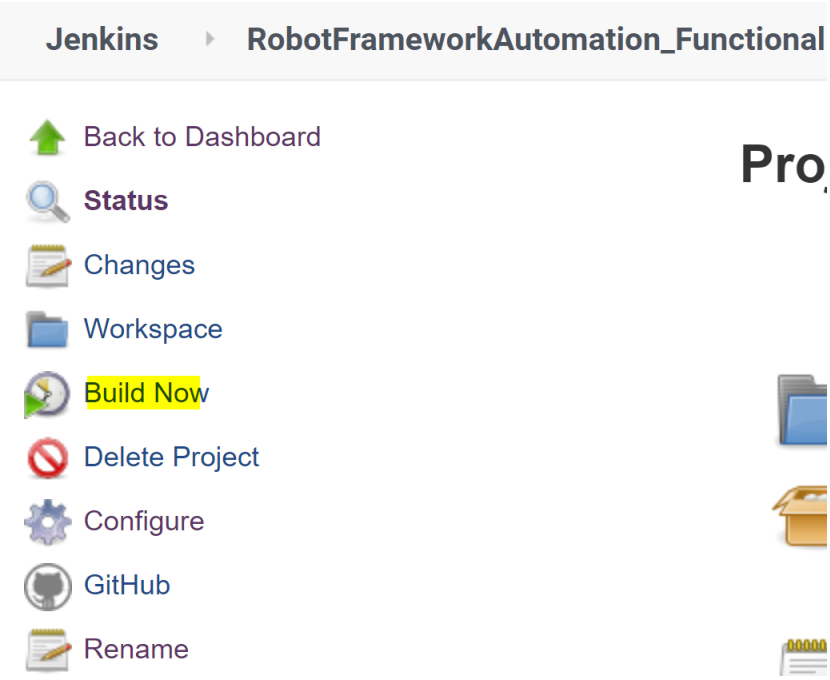
pip install -r Requirements.txt

robot -d "Reports" TestCases/Functional/*.robot

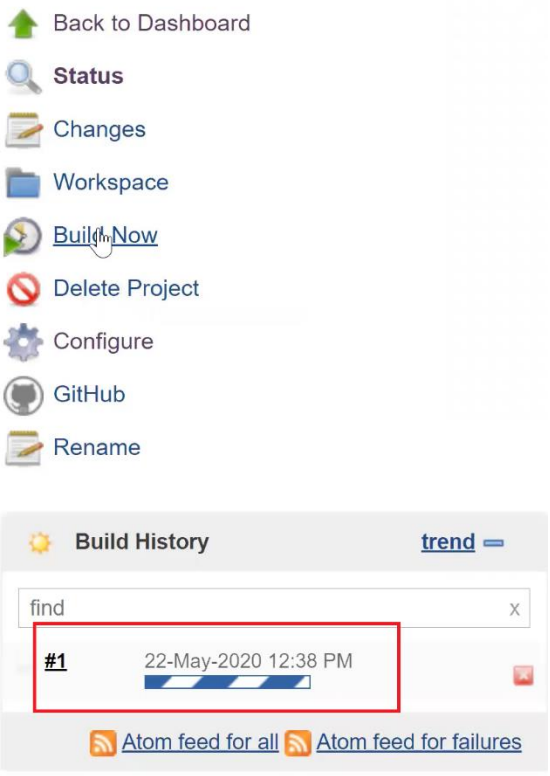


Now lets test this configuration quickly , scroll down first click on Apply and then click on Save

Click on Build Now . You will see browser is launched and it will start running test cases



It will trigger the build as you can see blow. Click on #1



Then click on Console Output

You will see it will run both suites we had under Functional directory

RobotFrameworkAutomation_Functional › #4

```
| PASS |
------------------------------------------------------------------------------
Perform action on TestData2                                           TestData2
| PASS |
------------------------------------------------------------------------------
Perform action on TestData3                                           TestData3
| PASS |
------------------------------------------------------------------------------
Perform action on TestData4                                           TestData4
| PASS |
------------------------------------------------------------------------------
DataDriven & FillContactForm.DataDriven                               | PASS |
4 critical tests, 4 passed, 0 failed
4 tests total, 4 passed, 0 failed
==============================================================================
[ ERROR ] Error in file 'C:\Users\user\.jenkins\jobs\RobotFrameworkAutomation_Functional\workspace\robotframeworkautomation\TestCases\Functional\FillContactForm.ro
ModuleNotFoundError: No module named 'ExcelLibrary'
Traceback (most recent call last):
  None
PYTHONPATH:
  C:\Users\user\AppData\Local\Programs\Python\Python38-32\Scripts\robot.exe
  c:\users\user\appdata\local\programs\python\python38-32\python38.zip
  c:\users\user\appdata\local\programs\python\python38-32\DLLs
  c:\users\user\appdata\local\programs\python\python38-32\lib
  c:\users\user\appdata\local\programs\python\python38-32
  C:\Users\user\AppData\Roaming\Python\Python38\site-packages
  c:\users\user\appdata\local\programs\python\python38-32\lib\site-packages
DataDriven & FillContactForm.FillContactForm
==============================================================================
Fill contact form for A1                                              | PASS |
------------------------------------------------------------------------------
Fill contact form for A2                                              | PASS |
------------------------------------------------------------------------------
Fill contact form for A3                                              | PASS |
------------------------------------------------------------------------------
DataDriven & FillContactForm.FillContactForm                          | PASS |
3 critical tests, 3 passed, 0 failed
3 tests total, 3 passed, 0 failed
==============================================================================
DataDriven & FillContactForm                                          | PASS |
7 critical tests, 7 passed, 0 failed
7 tests total, 7 passed, 0 failed
==============================================================================
Output:  C:\Users\user\.jenkins\jobs\RobotFrameworkAutomation_Functional\workspace\robotframeworkautomation\Reports\output.xml
Log:     C:\Users\user\.jenkins\jobs\RobotFrameworkAutomation_Functional\workspace\robotframeworkautomation\Reports\log.html
Report:  C:\Users\user\.jenkins\jobs\RobotFrameworkAutomation_Functional\workspace\robotframeworkautomation\Reports\report.html

C:\Users\user\.jenkins\jobs\RobotFrameworkAutomation_Functional\workspace\robotframeworkautomation>exit 0
Archiving artifacts
Finished: SUCCESS
```

Now we will create one more build for Smoke test suite, everything will remain same except the Commandw e will change the execution to Smoke directory

dir

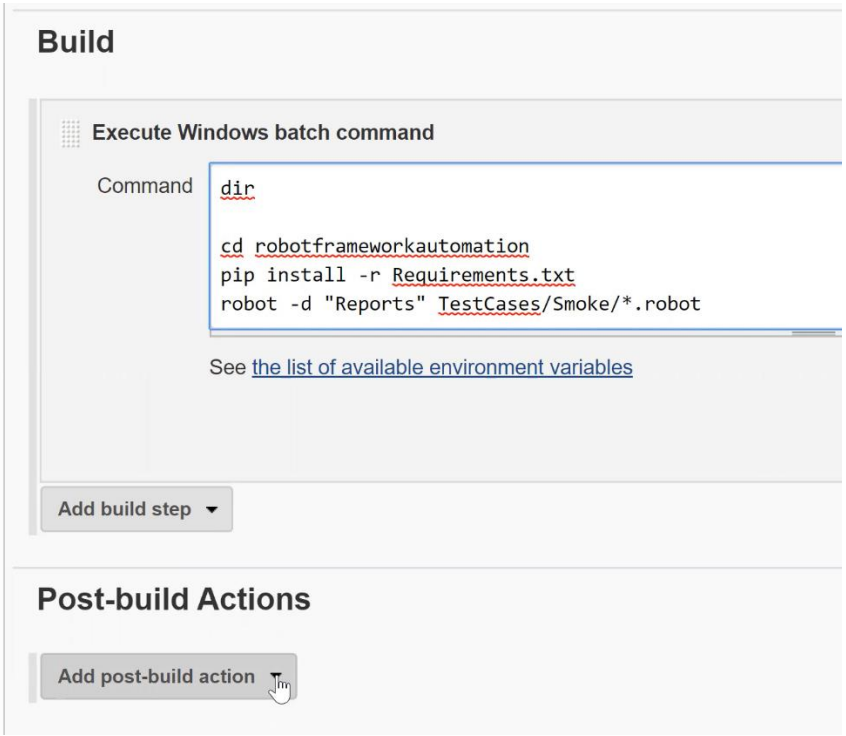
cd robotframeworkautomation

pip install -r Requirements.txt

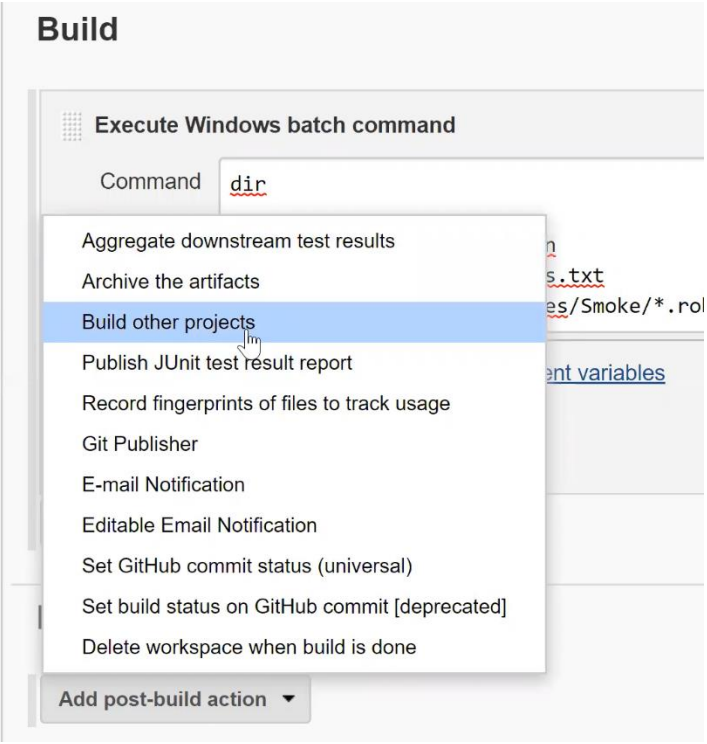
robot -d "Reports" TestCases/Smoke/*.robot

Next we will link this build to Functional build job we just created and condition if all tests under Smoke suites than only it will trigger build job for running functional tests.

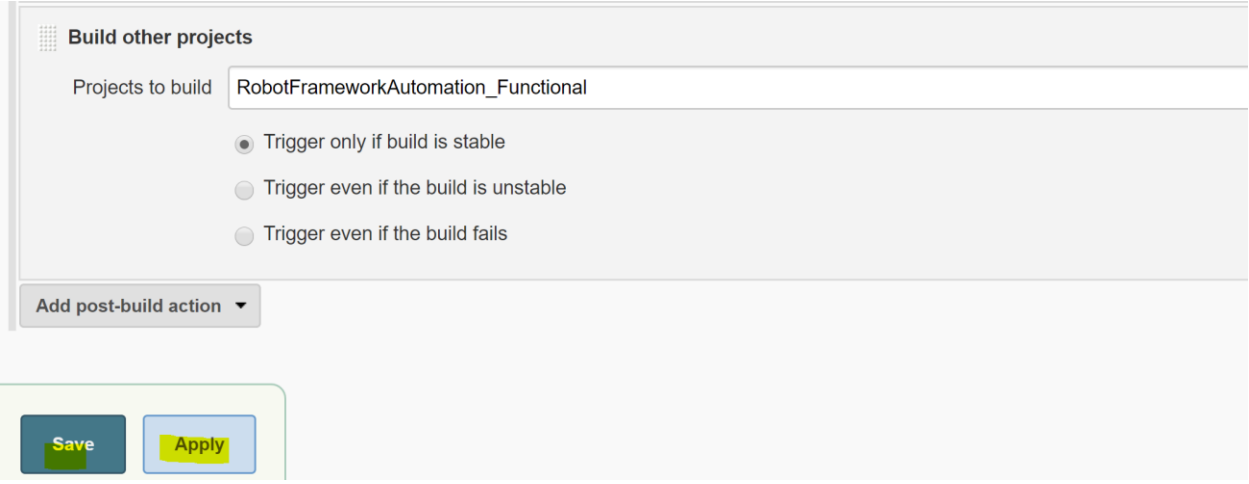
Scroll down to Post-build Actions. Click on the button



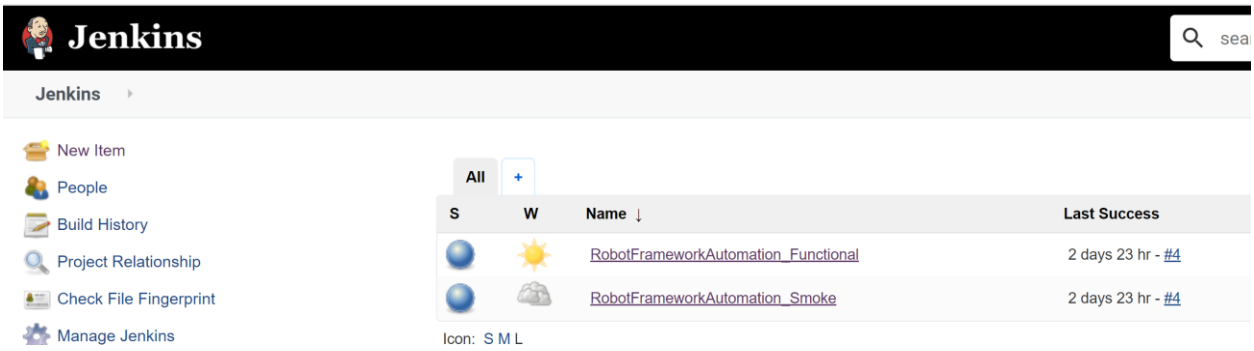
Select Build other projects



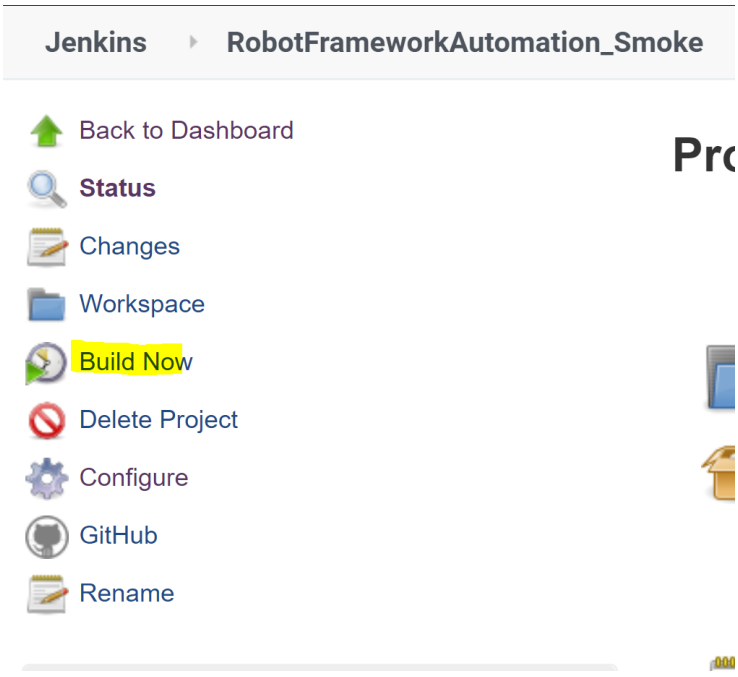
In Projects to build field start typing name of the build job of Functional you will see suggested options select . Now we have set option Trigger only if build is stable Click Apply and Save



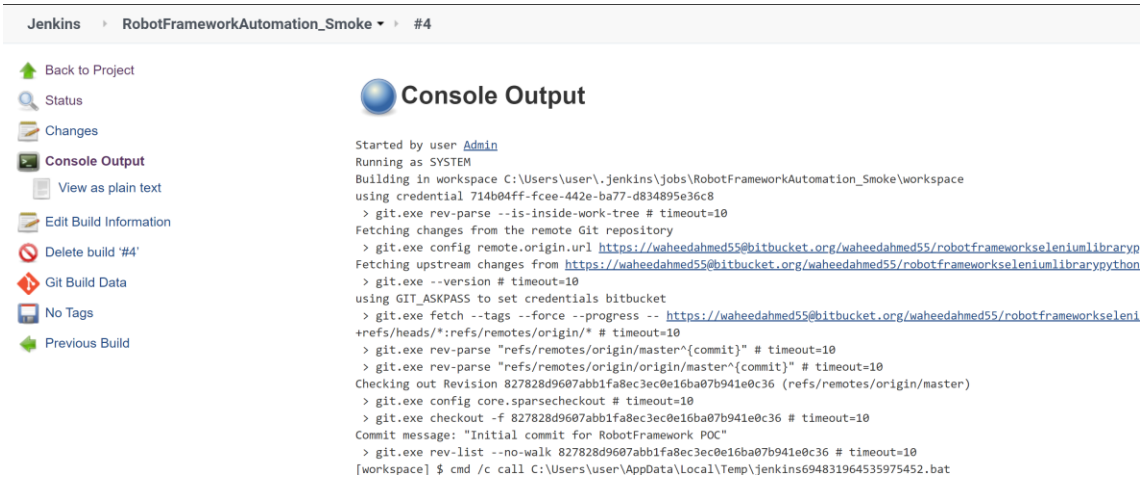
You should see on Jenkins Home page two builds . We have linked both of them by setting trigger condition if Smoke build passes then only trigger Functional build job. Now click on Smoke build



Click on Build Now and go to Builds Console Output



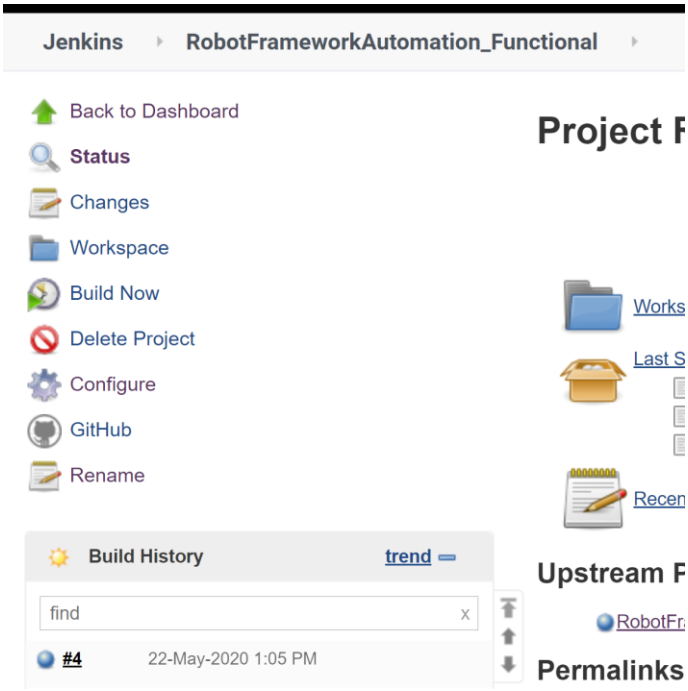
You will see Jenkins making access to repository in bitbucket



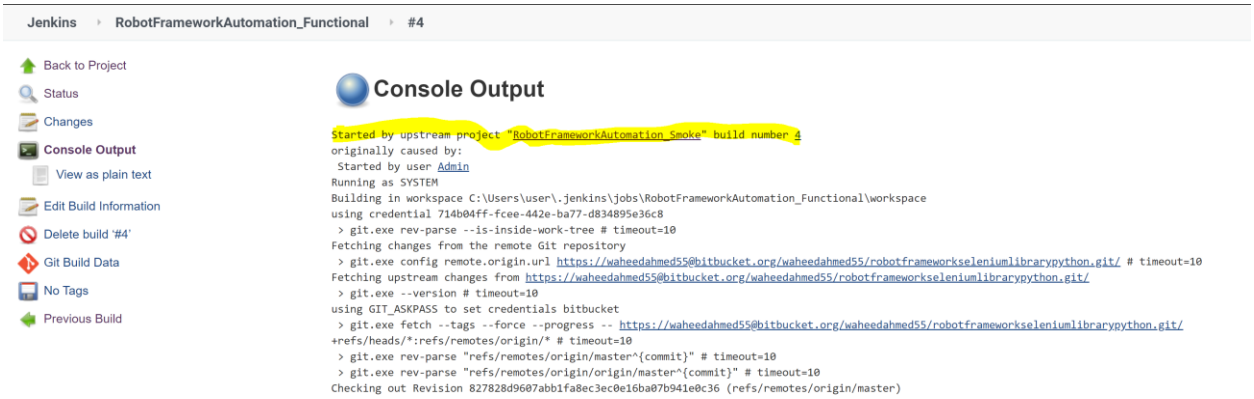
If you scroll to bottom of the page, if all smoke test cases passes , their will be trigger even to functional build job. You can click on the hyper link

```
==============================================================================
Output:  C:\Users\user\.jenkins\jobs\RobotFrameworkAutomation_Smoke\workspace\robotframeworkautomation\Reports\output.xml
Log:     C:\Users\user\.jenkins\jobs\RobotFrameworkAutomation_Smoke\workspace\robotframeworkautomation\Reports\log.html
Report:  C:\Users\user\.jenkins\jobs\RobotFrameworkAutomation_Smoke\workspace\robotframeworkautomation\Reports\report.html

C:\Users\user\.jenkins\jobs\RobotFrameworkAutomation_Smoke\workspace\robotframeworkautomation>exit 0
Archiving artifacts
Triggering a new build of RobotFrameworkAutomation_Functional
Finished: SUCCESS
```

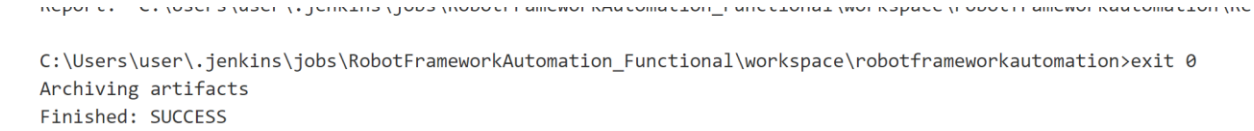
Click on build #4 . Click on Console output



It will display at top that this build got triggered by Robotframeworkautomation_Smoke



Once all test cases under Functional passes you will see success



Common Keywords & Testcase Control:

Now lets say we have keywords which we will use repetitively not only within same test class but also in different classes for those keywords what we will do is we will create CommonHelper.robot file under Helper directory.

Example : In below screenshot you can see in HandleAndFrame.robot test case class we used keyword

Navigate to the site ${URL_Of_website} in multiple test cases

```
*** Test Cases ***
Verify Alert
    Navigate to the site    ${AlertPageUrl}
    Click on the alert button
    Accept the alert

Verify Alert with cancel
    Navigate to the site    ${AlertPageUrl}
    Cllick on Alert with Cancel link
```

Which we defined/implemented in its HandleHelper.robot class

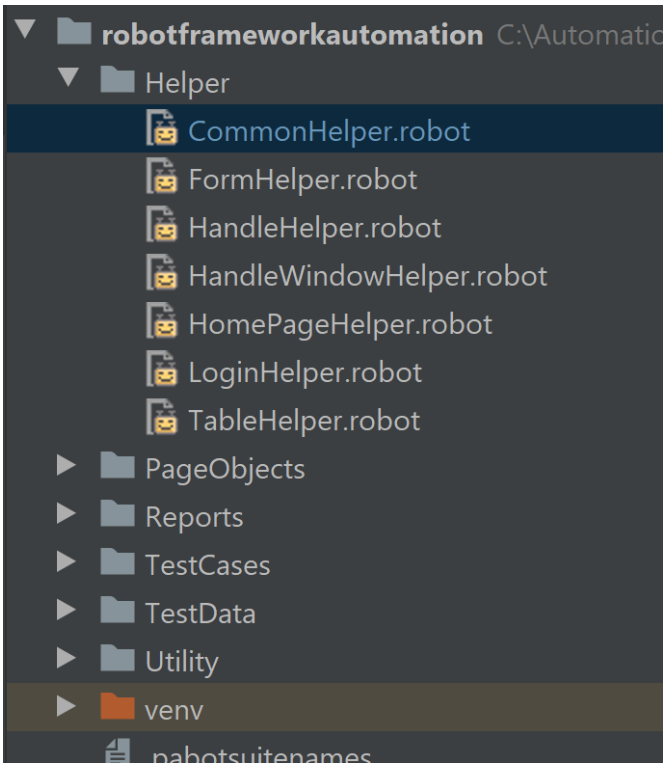
HandleHelper.robot

```
1   *** Settings ***
2   Resource    ../PageObjects/HandleObjects.robot
3   Resource    ../Utility/SeleniumKeywords.robot
4   Library     BuiltIn
5
6   *** Keywords ***
7   Navigate to the site
8       [Arguments]     ${URL}
9       Navigate to the page    ${URL}
10
11  Click on the alert button
12      sleep   3s
13      Click On Button   ${NormalAlert}
14      sleep   3s
15  Accept the alert
16      Handle the alert box    ACCEPT
```

Similarly, we other keywords like Login and Logout Application.

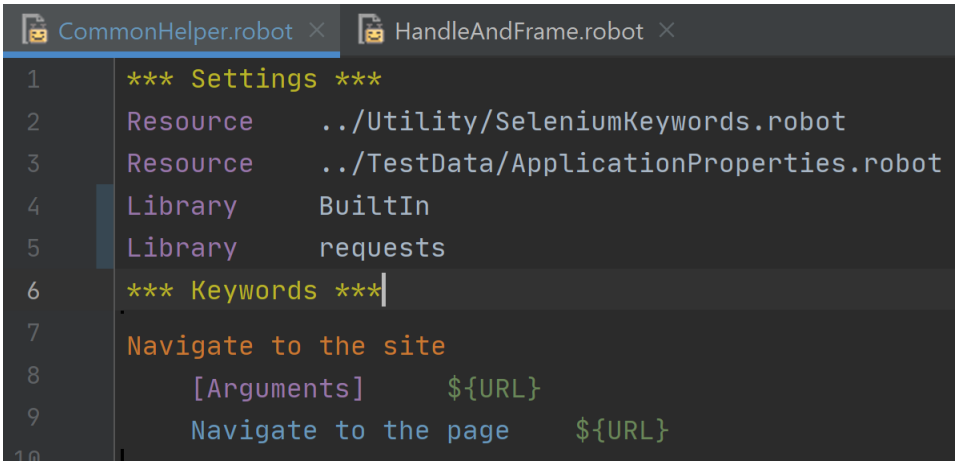
For example on this website http://practice.automationtesting.in/my-account/ we have to login every time before start test case and log out at the end of test case now we have to structure our flow in such a way we can achieve that at same time goal is to re-use those login and out keywords in multiple different test cases classes.

Lets gets started create CommonHelper.robot class under Helper folder:

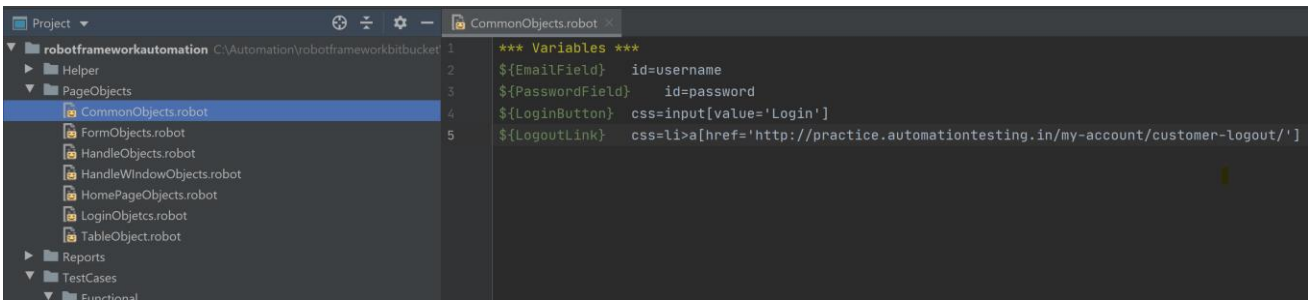


Now as usual we will have our Settings section in it and import SeleniumKeywords.robot and ApplicationProperties.robot along with other libraries required for console out etc.

Following to it will be Keywords section that's where we will add the implementation of common keywords like Navigate to the site for this just copy the code from HandleHelper.robot (remove from it) and paste in CommonHelper.robot just like you can see below:



Next I will write the Login and Logout the keywords which I will use in our Test Setup and Test Teardown. Since the objects which we will interact with for example username field , password field or login button will be common thus we will create CommonObjects.robot class under PageObjects directory:



Yes we will need all of these page objects to be passed as argument to respective keywords.

As you can see below I added the Login to the application keyword and arguments which we will be passing from test case , for login we just need page object of user name field , username text , password field, password text, and Login button.

CommonHelper.robot

```
*** Settings ***
Resource    ../Utility/SeleniumKeywords.robot
Resource    ../TestData/ApplicationProperties.robot
Library     BuiltIn
Library     requests
*** Keywords ***
Login to the application
    [Arguments]     ${userNameField}    ${userName}     ${PasswordField}    ${Password}     ${LoginButton}
    Set Value For Input Field    ${userNameField}  ${userName}
    Set Value For Input Field     ${PasswordField}  ${Password}
    Click On Page Element     ${LoginButton}

Log out the application
    [Arguments]     ${LogoutLink}
    Click On Page Element  ${LogoutLink}

Navigate to the site
    [Arguments]     ${URL}
    Navigate to the page    ${URL}
```

And for Log out the application we just need to pass Logout button page object from test case.

Now lets look at test class I created UserDefinedKeywords.robot under Regression folder. As you can see below I have imported CommonHelper.robot CommonObjects.robot in Settings section. Also note I have used the Login to the Application keyword in Test Setup and Test Teardown

UserDefinedKeywords.robot

```
*** Settings ***
Resource    ../../Utility/Setup.robot
Resource    ../../Helper/CommonHelper.robot
Resource    ../../TestData/ApplicationProperties.robot
Resource    ../../PageObjects/CommonObjects.robot
Suite Setup  Launch browser and navigate to   ${NewURL}
Test Setup  Login to the application    ${EmailField}   ${UserName}     ${PasswordField}    ${NewPass}  ${LoginButton}
Test Teardown  Log out the application  ${LogoutLink}
Suite Teardown  Close All Browser Window

*** Test Cases ***
Login Keyword
    log to console  Login Keyword

Logout Keyword
    log to console  Logout Keyword
```

Now before ou run let me know explain how It will be executed this important when deciding the flow of test cases choice is your either you want to login and logout before every test cases which I did in this example using Test Setup and Test Teardown or you just want to Login once using Test Setup and Logout using Suite Teardown.

Now once you run the class it will launch the browser and navigate to site and then it will perform login right after it will execute the first Test case which is Login Keyword and after its done executing all steps in this test case it will go to Test Teardown which is going to call common keyword Log out the application and perform logout. Now next again before starting the Logout Keyword test case it will execute Test Setup which is calling common keyword Login to the application and then it will run the Logout Keyword test cases and once done executing all steps in it, it will go to Test Teardown which will call common keyword Logout the application. So this is how we achieved the reusability of the common keyword at same time how to set flow of test cases.

Console Output:

```
(venv) C:\Automation\robotframeworkbitbucket\robotframeworkseleniumlibrarypython\robotframeworkautomation>robot -d "Reports" TestCases/Regression/UserDefinedKeywords.robot
==============================================================================
UserDefinedKeywords
==============================================================================

DevTools listening on ws://127.0.0.1:53420/devtools/browser/3bcf987c-9855-4fba-aef6-a0f5f8f6a0fa
headless
Login Keyword                                                 .Login Keyword
Login Keyword                                                 | PASS |
------------------------------------------------------------------------------
Logout Keyword                                                .Logout Keyword
Logout Keyword                                                | PASS |
------------------------------------------------------------------------------
[19824:220:0602/161952.263:ERROR:broker_win.cc(55)] Error reading broker pipe: The pipe has been ended. (0x6D)
UserDefinedKeywords                                           | PASS |
2 critical tests, 2 passed, 0 failed
2 tests total, 2 passed, 0 failed
==============================================================================
Output:  C:\Automation\robotframeworkbitbucket\robotframeworkseleniumlibrarypython\robotframeworkautomation\Reports\output.xml
Log:     C:\Automation\robotframeworkbitbucket\robotframeworkseleniumlibrarypython\robotframeworkautomation\Reports\log.html
Report:  C:\Automation\robotframeworkbitbucket\robotframeworkseleniumlibrarypython\robotframeworkautomation\Reports\report.html
```

Now lets move to HandleAndFrame.robot example in this I have removed the implementation from HandleHelper.robot class and imported the CommonHelper in HandleAndFrame.robot test class.

As you can see below:

HandleHelper.robot (removed the Navigate to the site)

HandleAndFrame.robot HandleHelper.robot

```
*** Settings ***
Resource    ../PageObjects/HandleObjects.robot
Resource    ../Utility/SeleniumKeywords.robot
Library     BuiltIn

*** Keywords ***
Click on the alert button
    sleep   3s
    Click On Button   ${NormalAlert}
    sleep   3s
Accept the alert
    Handle the alert box    ACCEPT

Click on the alert with Cancel button
    sleep   3s
    Click On Button   ${AlertWithCancel}
    sleep   3s

Cllick on Alert with Cancel link
    Click On Page Element   ${AlertWithCancelLink}
```

HandleAndFrame.robot : So you can see I mported the CommonHelper.robot to use common keyword Navigate to the site

HandleAndFrame.robot

```
1   *** Settings ***
2   Resource    ../../Utility/Setup.robot
3   Resource    ../../Helper/HandleHelper.robot
4   Resource    ../../Helper/CommonHelper.robot
5   Resource    ../../TestData/ApplicationProperties.robot
6   Suite Setup  Open the browser in
7   Test Setup  Go back to homepage
8   Test Teardown  Capture screenshot if test failed   Dummy
9   Suite Teardown  Close All Browser Window
10
11  *** Test Cases ***
12  Verify Alert
13      Navigate to the site    ${AlertPageUrl}
14      Click on the alert button
15      Accept the alert
16
17  Verify Alert with cancel
18      Navigate to the site    ${AlertPageUrl}
19      Cllick on Alert with Cancel link
20      Click on the alert with Cancel button
21      Accept the alert
22      Click on the alert with Cancel button
23      Dismiss the alert
```

Code: HandleAndFrame.robot

```
*** Settings ***
Resource    ../../Utility/Setup.robot
Resource    ../../Helper/HandleHelper.robot
Resource    ../../Helper/CommonHelper.robot
Resource    ../../TestData/ApplicationProperties.robot
Suite Setup  Open the browser in
Test Setup  Go back to homepage
Test Teardown  Capture screenshot if test failed   Dummy
Suite Teardown  Close All Browser Window

*** Test Cases ***
Verify Alert
    Navigate to the site    ${AlertPageUrl}
    Click on the alert button
    Accept the alert

Verify Alert with cancel
    Navigate to the site    ${AlertPageUrl}
    Cllick on Alert with Cancel link
    Click on the alert with Cancel button
    Accept the alert
    Click on the alert with Cancel button
    Dismiss the alert

Verify Alert with Textbox
    Navigate to the site    ${AlertPageUrl}
    Click on Alert with Textbox link
    Click on the alert with textbox button
    Enter text in alert     Dummy Alert Text

Verify Frame
    Navigate to the site    ${FrameUrl}
    Switch to the frame
    Enter text in the field     Dummy Alert Text
    Come out from frame
```

Code: UserDefinedKeywords.robot

```
*** Settings ***
Resource    ../../Utility/Setup.robot
Resource    ../../Helper/CommonHelper.robot
Resource    ../../TestData/ApplicationProperties.robot
Resource    ../../PageObjects/CommonObjects.robot
Suite Setup  Launch browser and navigate to   ${NewURL}
Test Setup  Login to the application   ${EmailField}  ${UserName}    ${PasswordField}   ${NewPass}
${LoginButton}
Test Teardown  Log out the application  ${LogoutLink}
Suite Teardown  Close All Browser Window

*** Test Cases ***
Login Keyword
   log to console  Login Keyword

Logout Keyword
   log to console  Logout Keyword
```

Code: CommonObjects.robot

```
*** Variables ***
${EmailField}   id=username
${PasswordField}    id=password
${LoginButton}  css=input[value='Login']
${LogoutLink}   css=li>a[href='http://practice.automationtesting.in/my-account/customer-logout/']
```

Code: CommonHelper.robot

```
*** Settings ***
Resource    ../Utility/SeleniumKeywords.robot
Resource    ../TestData/ApplicationProperties.robot
Library     BuiltIn
Library     requests
*** Keywords ***
Login to the application
   [Arguments]    ${userNameField}   ${userName}    ${PasswordField}   ${Password}    ${LoginButton}
   Set Value For Input Field   ${userNameField}  ${userName}
   Set Value For Input Field    ${PasswordField}  ${Password}
   Click On Page Element    ${LoginButton}

Log out the application
   [Arguments]    ${LogoutLink}
   Click On Page Element  ${LogoutLink}

Navigate to the site
   [Arguments]    ${URL}
   Navigate to the page   ${URL}
```

Code: ApplicationProperties.robot

```
*** Variables ***
${URL}    https://demo.nopcommerce.com/
${UserName}    waheedahmed55@gmail.com
${InvalidPassword}    test1
${Password}    test123
${SearchItem}    Laptop
${AlertPageUrl}    http://demo.automationtesting.in/Alerts.html
${FrameUrl}    http://demo.automationtesting.in/Frames.html
${TableUrl}    https://info.sice.indiana.edu/~hrosenba/Demo/Demo4.html
${ContactFormURL}    https://amjottawaeast-27647.web.app
${WindowURL}    http://demo.automationtesting.in/Windows.html
${NewPass}    LH9n@E@36zSqHp7
${NewURL}    http://practice.automationtesting.in/my-account/
${RegisterURL}    http://demo.automationtesting.in/Register.html
```