

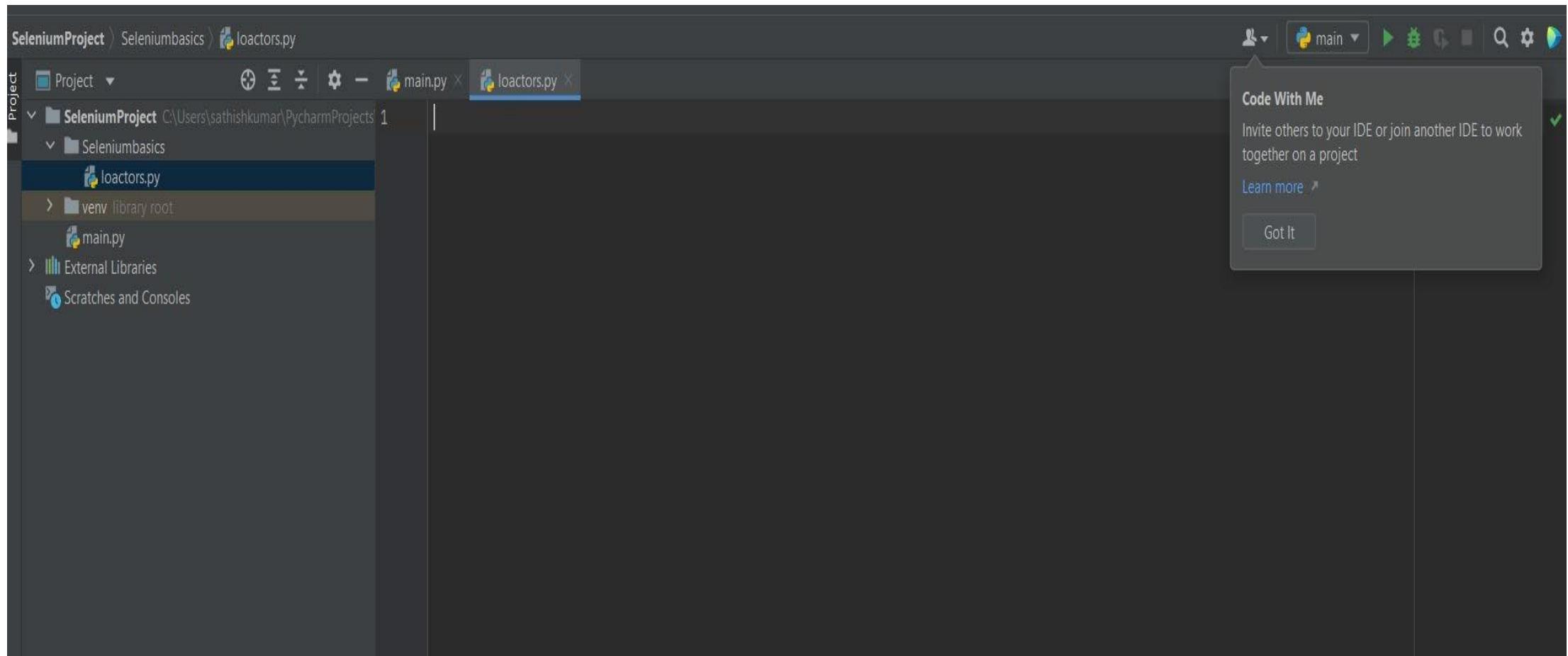
Selenium With Python

Selenium installation

Open the command prompt and type “**Python -m pip install -U Selenium**” press enter

```
C:\Users\sathishkumar>Python -m pip install -U Selenium
Collecting Selenium
  Downloading selenium-3.141.0-py2.py3-none-any.whl (904 kB)
    |██████████| 904 kB 6.4 MB/s
Collecting urllib3
  Downloading urllib3-1.26.6-py2.py3-none-any.whl (138 kB)
    |██████████| 138 kB 6.4 MB/s
Installing collected packages: urllib3, Selenium
Successfully installed Selenium-3.141.0 urllib3-1.26.6
WARNING: You are using pip version 21.2.3; however, version 21.2.4 is available.
You should consider upgrading via the 'C:\Users\sathishkumar\AppData\Local\Programs\Python\Python39\python.exe -m pip install --upgrade pip' command.
```

Sample project with package and python file creation



How to open the browser

- from selenium import webdriver

class locator:

```
driver =  
webdriver.Chrome('D:\Software\chromedriver_win32\chromedriver.exe')
```

To enter the url

```
driver.get("https://en-gb.facebook.com/")
```

How to maximize and close the browser

To Maximize the window (fit to the screen)

```
driver.maximize_window()
```

To Minimize the window

- `driver.minimize_window();`

To set it to particular size

- `driver.set_window_size(300,600)`

To close the browser

- `driver.close()`

To close the session

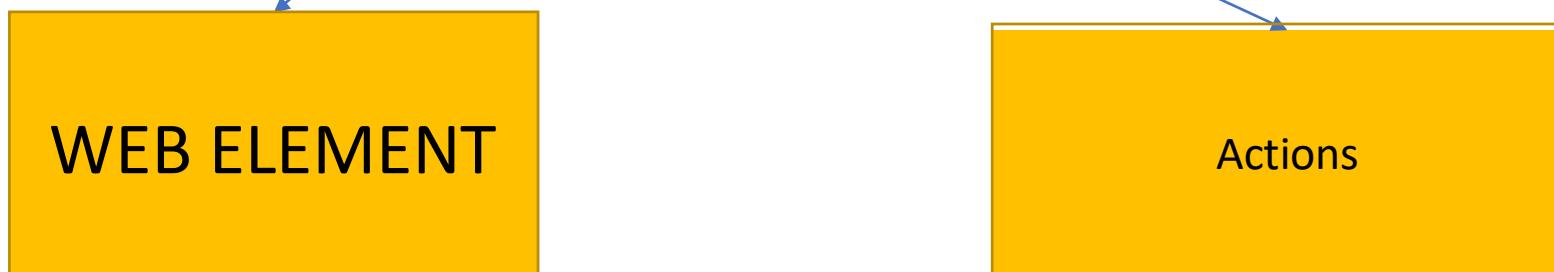
- `self.driver.quit()`

Locators

Method	Syntax	Description
By ID	driver.find_element_by_id(<element name>)	Locates an element using the ID attribute
By name	driver.find_element_by_name(<element name>)	Locates an element using the Name attribute
By class name	driver.find_element_by_classname(<element class>))	Locates an element using the Class attribute
By tag name	driver.find_element_by_tag_name(<htmltagname>))	Locates an element using the HTML tag
By link text	driver.find_element_by_link_text(<linktext >))	Locates a link using link text
By partial link text	driver.find_element_by_partial_link_text(<linktext>))	Locates a link using the link's partial text
By CSS	driver.find_element_by_css_selector(<css selector>))	Locates an element using the CSS selector
By XPath	driver.find_element_by_xpath(<xpath>))	Locates an element using XPath query

Locators

- `username=driver.find_element_by_id("email")`
- `username.send_keys("sathishkumar")`



Locators –Css Selector

With this strategy, the first element with the matching CSS selector will be returned. If no element has a matching CSS selector, a NoSuchElementException will be raised.

Four types of css selector

1. Tag and id
2. Tag and class
3. Tag and Attribute
4. Tag class and Attribute

Locators –Css Selector

Tag and id

Syntax : Tag#id

example: `driver.find_element_by_css_selector("input#gsc-i-id2")`

Tag and class

Syntax : Tag.class

example: `driver.find_element_by_css_selector("input.gsc-i-id2")`

Tag and Attribute

Syntax : Tag[attribute=value]

example: `driver.find_element_by_css_selector("input[name=gsc-i-id2]")`

Tag class and Attribute

Syntax : Tag.class[attribute=value]

example: `driver.find_element_by_css_selector("input.clss[name=gsc-i-id2]")`

Locators –Xpath Selector

XPath is the language used for locating nodes in an XML document.

One of the main reasons for using XPath is when you don't have a suitable id or name attribute for the element you wish to locate. You can use XPath to either locate the element in absolute terms (not advised), or relative to an element that does have an id or name attribute. XPath locators can also be used to specify elements via attributes other than id and name.

There are 2 Types of Xpath:

- Absolute Xpath
- Relative Xpath

Locators – Xpath : Absolute Xpath

Absolute Xpath

- It is the direct way to find the element, but the disadvantage of the absolute XPath is that if there are any changes made in the path of the element then that XPath gets failed.
- The key characteristic of XPath is that it begins with the single forward slash(/) ,which means you can select the element from the root node.

E.g :

`/html/body/div[2]/div[1]/div/h4[1]/b/html[1]/body[1]/div[2]/div[1]/div[1]/h4[1]/b[1]`

Relative Xpath

Relative Xpath starts from the middle of HTML DOM structure. It starts with double forward slash (//). It can search elements anywhere on the webpage, means no need to write a long xpath and you can start from the middle of HTML DOM structure. Relative Xpath is always preferred as it is not a complete path from the root element.

E.G :

`//div[@class='featured-box columnsize1']//h4[1]//b[1]`

Locators –Xpath :Relative Xpath

Using XPath Handling complex & Dynamic elements in Selenium

- 1) Basic Xpath
- 2) Contains()
- 3) Using OR & AND
- 4) Xpath Starts-with
- 5) XPath Text() Function

1. Basic Xpath

XPath expression select nodes or list of nodes on the basis of attributes like ID , Name, Classname, etc. from the XML document as illustrated below.

Xpath=//Tagname[@Attribute='Value']

Locators –Xpath :Relative Xpath

2. Contains

Contains() is a method used in XPath expression. It is used when the value of any attribute changes dynamically

Xpath=//*[contains(@Attribute,'Value')]

3. Using OR & AND

In OR expression, two conditions are used, whether 1st condition OR 2nd condition should be true. It is also applicable if any one condition is true or maybe both. Means any one condition should be true to find the element.

Xpath=//*[@attribute1='value1' **or** @attribute2='value2']

In AND expression, two conditions are used, both conditions should be true to find the element. It fails to find element if any one condition is false.

Xpath=//*[@attribute1='value1' **and** @attribute2='value2']

4. Xpath Starts-with

In this method, the starting text of the attribute is matched to find the element whose attribute value changes dynamically. You can also find elements whose attribute value is static (not changes).

Locators –Xpath :Relative Xpath

Xpath=//Tagname[**starts-with(@attribute,'value')**]

5. XPath Text() Function

The XPath text() function is a built-in function of selenium webdriver which is used to locate elements based on text of a web element. It helps to find the exact text elements and it locates the elements within the set of text nodes. The elements to be located should be in string form.

Xpath=//Tagname[**text()**=‘value’]

XPath axes methods:

These XPath axes methods are used to find the complex or dynamic elements. Below we will see some of these methods.

For illustrating these XPath axes method, we will use the Guru99 bank demo site.

a) Following:

Xpath=//*[@attribute='value']//following::tagname

Locators –Xpath :Relative Xpath

b) Ancestor:

The ancestor axis selects all ancestors element (grandparent, parent, etc.) of the current node as shown in the below screen.

Xpath=//*[text()='Enterprise Testing']//**ancestor**::Tagname

c) **Child:**

Selects all children elements of the current node (Java) as shown in the below screen.

Xpath=//*[@id='java_technologies']//**child**::Tagname

d) **Preceding:**

Select all nodes that come before the current node as shown in the below screen.

Xpath=//*[@type='submit']//**preceding**::input

Locators –Xpath :Relative Xpath

e) Following-sibling:

Select the following siblings of the context node. Siblings are at the same level of the current node as shown in the below screen. It will find the element after the current node.

Xpath=//*[@type='submit']//**following-sibling**::input

f) Parent:

Selects the parent of the current node as shown in the below screen.

Xpath=//*[@id='rt-feature']//**parent**::div

h) Descendant:

Selects the descendants of the current node as shown in the below screen.it identifies all the element descendants to current element (‘Main body surround’ frame element) which means down under the node (child node , grandchild node, etc.).

Xpath=//*[@id='rt-feature']//**descendant**::a

Synchronization

These days, most of the web apps are using AJAX techniques. When a page is loaded by the browser, the elements within that page may load at different time intervals. This makes locating elements difficult: if an element is not yet present in the DOM, a locate function will raise an `ElementNotVisibleException` exception. Using waits, we can solve this issue. Waiting provides some slack between actions performed - mostly locating an element or any other operation with the element.

Selenium Webdriver provides two types of waits

- Implicit Wait
- Explicit Wait

Syncronization – Implicit Wait

IMPLICIT WAIT:

An implicit wait tells WebDriver to poll the DOM for a certain amount of time when trying to find any element (or elements) not immediately available. The default setting is 0 (zero). Once set, the implicit wait is set for the life of the WebDriver object.

Syntax:

```
driver.implicitly_wait(10) # seconds
```

EXPLICIT WAIT:

An explicit wait is a code you define to wait for a certain condition to occur before proceeding further in the code

Syncronization – Explicit Wait

Syntax:

```
from selenium.webdriver.support.ui import WebDriverWait
```

```
from selenium.webdriver.support import expected_conditions as EC
```

```
element = WebDriverWait(driver, 10).until(
```

```
    EC.presence_of_element_located((By.ID, "myDynamicElement")))
```

Synchronization – Explicit Wait

Expected Conditions

There are some common conditions that are frequently of use when automating web browsers. Listed below are the names of each.

Selenium Python binding provides some convenience methods so you don't have to code an `expected_condition` class yourself or create your own utility package for them.

Methods in Expected Conditions

- title_is
- title_contains
- presence_of_element_located
- visibility_of_element_located
- visibility_of
- presence_of_all_elements_located
- text_to_be_present_in_element
- text_to_be_present_in_element_value
- frame_to_be_available_and_switch_to_it
- invisibility_of_element_located
- element_to_be_clickable
- staleness_of
- element_to_be_selected
- element_located_to_be_selected
- element_selection_state_to_be
- element_located_selection_state_to_be
- alert_is_present

Syncronization – time.sleep(sec)

Simply put, the `time.sleep()` takes as arguments the **number of seconds you want the program to halt or to be suspended** before it moves forward to the next step. Let's see this with the help of a simple program, where we simply output the current time with a halt of 5 seconds in between the next execution - hence introducing a difference of 5 seconds between each print of the current time.

The syntax for the function is:

time.sleep(sec)

where sec is the argument that specifies the number of seconds the program is to be suspended for. The function does not return any value.

Dropdown

The Select Class in Selenium is a method used to implement the HTML SELECT tag. The html select tag provides helper methods to select and deselect the elements. The Select class is an ordinary class so New keyword is used to create its object and it specifies the web element location.

There are 2 Type of dropdown to handle

- 1.Single select dropdown
- 2.Multiselect dropdown

Dropdown

Element	Command	Description
Drop-Down Box	<i>selectByVisibleText()</i> / <i>deselectByVisibleText()</i>	selects/deselects an option by its displayed text
	<i>selectByValue()</i> / <i>deselectByValue()</i>	selects/deselects an option by the value of its “value” attribute
	<i>selectByIndex()</i> / <i>deselectByIndex()</i>	selects/deselects an option by its index
	<i>isMultiple()</i>	returns TRUE if the drop-down element allows multiple selection at a time; FALSE if otherwise
	<i>deselectAll()</i>	deselects all previously selected options

Mouse and Keyboard

Mouse actions are nothing but simulating the mouse events like operations performed using Mouse. In Selenium Python, handling Mouse & keyboard events and mouse events (including actions such as Drag and Drop or clicking multiple elements With Control key) are done using the **ActionChains API**.

Mouse Actions :

- mouseMove
- dragAndDrop
- click
- doubleClick
- mouseUp
- mouseDown

Mouse and Keyboard

Keyboard Actions :

- keyDown
- keyUp
- send_keys

perform() is a function that will make the mouse perform an operation on the webpage, if you don't use perform() function then ActionChains don't have any effect on the webpage.

When you call methods for actions on the ActionChains object, the actions are stored in a queue in the ActionChains object. When you call perform(), the events are fired in the order they are queued up.

Mouse action - Methods

Method	Description
<u>click</u>	Clicks an element.
<u>click_and_hold</u>	Holds down the left mouse button on an element.
<u>context_click</u>	Performs a context-click (right click) on an element.
<u>double_click</u>	Double-clicks an element.
<u>drag_and_drop</u>	Holds down the left mouse button on the source element, then moves to the target element and releases the mouse button.
<u>drag_and_drop_by_offset</u>	Holds down the left mouse button on the source element, then moves to the target offset and releases the mouse button.

Mouse action - Methods

<u>move_by_offset</u>	Moving the mouse to an offset from current mouse position.
<u>move_to_element</u>	Moving the mouse to the middle of an element.
<u>move_to_element_with_offset</u>	Move the mouse by an offset of the specified element, Offsets are relative to the top-left corner of the element.
<u>perform</u>	Performs all stored actions.
<u>pause</u>	Pause all inputs for the specified duration in seconds
<u>release</u>	Releasing a held mouse button on an element.
<u>reset_actions</u>	Clears actions that are already stored locally and on the remote end

Keyboard Actions - syntax

Action	Arguments	Description
<code>send_keys(*keys_to_send)</code>	<p><i>keys_to_send</i> – The keys to send. Modifier keys constants can be found in the <i>Keys</i> class.</p>	Send keys to the element that is currently in focus.
<code>key_down(value, element=None)</code>	<p><i>value</i> – Modifier key to send. <i>element</i> – It is an optional argument. It represents the element on which keys need to be sent. If it is not specified, i.e., None, the key is sent to the currently focused element.</p>	Sends a key press without performing the release. It should only be used with modifier keys like Control, Alt, and Shift.
<code>key_up(value, element=None)</code>	<p><i>value</i> – Modifier key to send. <i>element</i> – It is an optional argument. It represents the element on which keys need to be sent. If it is not specified, i.e., None, the key is sent to the currently focused element.</p>	Releases a key. It should only be used with modifier keys like Control, Alt, and Shift.
<code>perform</code>	None	Perform the chain of actions stored in the ActionChains object

Keyboard using pyautogui

It has following method

- The write() Function
- The press(), keyDown(), and keyUp() Functions
- The hold() Context Manager
- The hotkey() Function

The primary keyboard function is write().

This function will type the characters in the string that is passed. To add a delay interval in between pressing each character key, pass an int or float for the interval keyword argument.

For example:

```
>>> pyautogui.write('Hello world!')          # prints out "Hello world!" instantly  
>>> pyautogui.write('Hello world!', interval=0.25) # prints out "Hello world!" with a quarter second
```

Keyboard using pyautogui

The `press()` function is really just a wrapper for the `keyDown()` and `keyUp()` functions, which simulate pressing a key down and then releasing it up. These functions can be called by themselves. For example, to press the left arrow key three times while holding down the Shift key, call the following:

```
>>> pyautogui.keyDown('shift') # hold down the shift key  
>>> pyautogui.press('left')    # press the left arrow key  
>>> pyautogui.press('left')    # press the left arrow key  
>>> pyautogui.press('left')    # press the left arrow key  
>>> pyautogui.keyUp('shift')   # release the shift key
```

Keyboard using pyautogui

The hold() Context Manager

To make holding a key convenient, the `hold()` function can be used as a context manager and passed a string from the `pyautogui.KEYBOARD_KEYS` such as `shift`, `ctrl`, `alt`, and this key will be held for the duration of the `with` context block. See `KEYBOARD_KEYS`.

```
>>> with pyautogui.hold('shift'):
    pyautogui.press(['left', 'left', 'left'])
```

The hotkey() Function

To make pressing hotkeys or keyboard shortcuts convenient, the `hotkey()` can be passed several key strings which will be pressed down in order, and then released in reverse order. This code:

```
>>> pyautogui.hotkey('ctrl', 'shift', 'esc')
```

Handling Alert

Handle Alert & Pop-up Boxes in Selenium Python

We can broadly categorize the Alerts into following three types.

- i) Simple Alert – **with ok button**
- ii) A Confirmation Alert – **with ok and cancel button**
- iii) Prompt Alert – **with ok ,cancel and textbox**
- We will now discuss how to handle the above three types of alerts in detail.
- Whenever an Alert gets triggered, and a pop-up appears on the web page. The control remains with the parent web page only. So the first task for the Selenium Webdriver is to switch the focus from the parent page to the Alert pop-up. It can be done using following code snippet.

```
alert_obj = driver.switch_to.alert
```

Handling Alert

- After the control has moved to Alert pop-up, we can do different actions on it using the recommended methods as.

`alert_obj.accept()` – used to accept the Alert

`alert_obj.dismiss()` – used to cancel the Alert

`alert.send_keys()` – used to enter a value in the Alert text box.

`alert.text` – used to retrieve the message included in the Alert pop-up.

Verification and Validation commands

Validation : (it will return only Boolean value)

- is_enabled() - Webdriverobject. Is_enabled()
- is_selected() - Webdriverobject. is_selected()
- is_displayed() - Webdriverobject. is_displayed()

Verification (it will return string / list of string value)

- title - Webdriverobject.title
- current_url - Webdriverobject. current_Url
- get_attribute - WebElement.get_attribute("attributename")
- text - WebElement.text
- current_window_handle - Webdriverobject. current_window_handle
- window_handles - Webdriverobject. window_handles

List Handle

```
def listhand(self,actualtext):
    self.driver.get("https://www.makemytrip.com/")
    self.driver.find_element_by_id("fromCity").click()
    totalvalue =self.driver.find_elements_by_xpath("//ul[@role='listbox']/li")
    sizeoflist = len(totalvalue)
    for x in range(1,sizeoflist):
        text_Value =
    self.driver.find_element_by_xpath("//ul[@role='listbox']/li["+str(x)+"]//div[2]").t
ext
        #print(text_Value)
        if text_Value == actualtext :
            self.driver.find_element_by_xpath("//ul[@role='listbox']/li["+str(x)+"]").click()
            break
```

WebTable Handle

```
def table(self,actualtext):
    self.driver.get("http://www.leafground.com/pages/table.html")
    totalrowvalue = self.driver.find_elements_by_xpath("//table[@id='table_id']/tr")
    sizeofrowlist = len(totalrowvalue)
    for x in range(2, sizeofrowlist+1):
        totalcolumnvalue =
    self.driver.find_elements_by_xpath("//table[@id='table_id']/tr[" + str(x) + "]/td")
        sizeofcolumnlist = len(totalcolumnvalue)
        text_Value = self.driver.find_element_by_xpath("//table[@id='table_id']/tr[" +
str(x) + "]/td[1]").text
        print(text_Value)
        if text_Value != actualtext:
            self.driver.find_element_by_xpath("//table[@id='table_id']/tr[" + str(x) +
"]//td[3]//input").click()
            #break
```

How to scroll the page

To perform scroll operation we can use the below methods

#scroll down

```
self.driver.execute_script("window.scrollTo(0, 600)", 0)
```

Pixel size

scroll up

```
self.driver.execute_script("window.scrollTo(0, -600)", 0)  
time.sleep(1)
```

scroll right

```
self.driver.execute_script("window.scrollTo(600, 0)", 0)  
time.sleep(1)
```

scroll left

```
self.driver.execute_script("window.scrollTo(-600, 0)", 0)
```

How to scroll the page

scroll bottom

```
self.driver.execute_script("window.scrollTo(0,  
document.body.scrollHeight);")
```

scroll to a specific element (Majorly used in realtime *)

```
element=self.driver.find_element_by_xpath("//h5[text()='Window']  
//parent::a")
```

```
self.driver.execute_script("arguments[0].scrollIntoView();",  
element)
```

Screenshot

Selenium offers a lot of features and one of the important and useful feature is of taking a screenshot. In order to take a screenshot of webpage `save_screenshot()` method is used. `save_screenshot` method allows user to save the webpage as a png file.

`driver.save_screenshot("image.png")`

Argument :

filename or the full path you wish to save your screenshot to.

Action performed :

The screenshot will be saved in the same directory as the program, if path is provided screenshot will be saved at that location only.

Handling Frames

We can handle frames in Selenium. A frame is an HTML element that keeps a document within another document in a page. HTML has the `<frame>` or `<iframe>` tags for embedding a frame inside a document.

There are multiple APIs available in Selenium to work with the frames. They are listed below –

1. `switch_to.frame(id)`
2. `switch_to.frame(name)`
3. `switch_to.frame(webelement)`
4. `switch_to.parent_frame()`
5. `switch_to.default_content()`

Handling Frames

switch_to.frame(id)

This method is used to identify a frame with the help of frame id and then switch the focus to that particular frame.

Syntax –

driver.switch_to.frame("frameid") → where frame id is the id attribute present under the frame/iframe tag in HTML.

switch_to.frame(name)

This method is used to identify a frame with the help of frame name and then switch the focus to that particular frame.

Syntax –

driver.switch_to.frame("framename") → where frame name is the name attribute present under the frame/iframe tag in HTML.

Handling Frames

switch_to.frame(webelment) :

This method is used to identify a frame with the help of frame webelement and then switch the focus to that particular frame.

Syntax –

driver.switch_to.frame("frameclassname") → where frameclassname is the name of class attribute present under the frame/iframe tag in HTML.

switch_to.parent_frame()

This method is used to come out of the present frame, then we can access the elements outside that frame and not inside of that frame.

switch_to.default_content()

This method is used to come out of all the frames and switch the focus at the page. Once we move out, it loses the access to the elements inside the frames in the page.

Handling Multiple Windows

We can handle child windows or tabs in Selenium. While working with child windows, we need to always shift the browser focus to the child windows, then perform operation on them.

By default, the focus remains on the first parent window. There are multiple methods available in Selenium which are listed below –

current_window_handle

window_handles

switch_to.window(args)

This method fetches the handle of the present window.

`driver.current_window_handle`

Handling Multiple Windows

window_handles

This method fetches all the handle ids of the windows that are currently open.

Syntax -

```
driver.window_handles
```

```
w = driver.window_handles[2]
```

The above code gives the handle id of the second window opened in the present session.

switch_to.window(args)

This method switches the focus of Selenium to the window name mentioned in the arguments.

Syntax -

```
driver.switch_to.window(childwindow)
```

How to upload the file

1.Upload using send_keys for upload textbox

We can upload files with Selenium using Python. This can be done with the help of the send_keys method. First, we shall identify the element which does the task of selecting the file path that has to be uploaded.

Syntax:

```
Webelement.send_Keys("filepathof the file to be upload")
```

How to upload the file

2.Upload through windows upload popup as below

```
self.driver.find_element_by_xpath("//div[@class='input-file-upload-hover-placeholder']//parent::div)[1]").click()
```

//Copy the value in to clipboard

```
pyperclip.copy("D:\\Besant\\Java basics.pdf")  
time.sleep(2)
```

//Paste the value using keyboard operation

```
pyautogui.hotkey('ctrl', 'v')  
pyautogui.press('enter')
```

How to download the file

Download the file in the default download location as below

```
driver =  
webdriver.Chrome('D:\Software\chromedriver_win32\chromedrive  
r.exe')  
    driver.get("http://www.leafground.com/pages/download.html")  
    driver.maximize_window()  
    driver.find_element_by_link_text("Download Excel").click()
```

How to download the file – in specific folder

2 .Download the file in the specific folder

```
options = webdriver.ChromeOptions();
    prefs = {"download.default_directory" :
"C:\\\\Users\\\\sathishkumar\\\\PycharmProjects\\\\SeleniumProject\\\\Downlo
adfile\\\\"}
options.add_experimental_option("prefs", prefs)
driver =
    webdriver.Chrome('D:\\Software\\chromedriver_win32\\chromedriv
er.exe',options=options)

    self.driver.get("http://www.leafground.com/pages/download.html
")
    self.driver.maximize_window()
    self.driver.find_element_by_link_text("Download Excel").click()
```

Log Files

Log is used for logging the information in to the file for tracking the execution

Types of Logs

- Debug
- Info
- Warning
- Error
- Critical

```
logger= logging.getLogger()
```

Pytest – Testing Framework

Pytest is a python based testing framework, which is used to write and execute test codes. In the present days of REST services, pytest is mainly used for API testing even though we can use pytest to write simple to complex tests, i.e., we can write codes to test API, database, UI, etc.

Advantages of Pytest

The advantages of Pytest are as follows –

- Pytest can run multiple tests in parallel, which reduces the execution time of the test suite.
- Pytest has its own way to detect the test file and test functions automatically, if not mentioned explicitly.
- Pytest allows us to skip a subset of the tests during execution.
- Pytest allows us to run a subset of the entire test suite.
- Pytest is free and open source.
- Because of its simple syntax, pytest is very easy to start with.

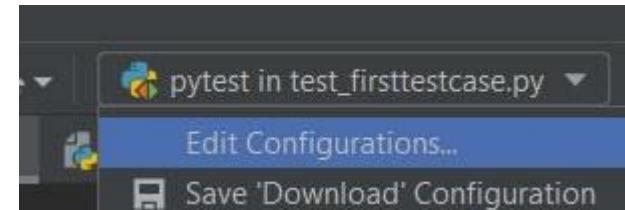
Pytest – Testing Framework

To Run the Pytest we can run by following way

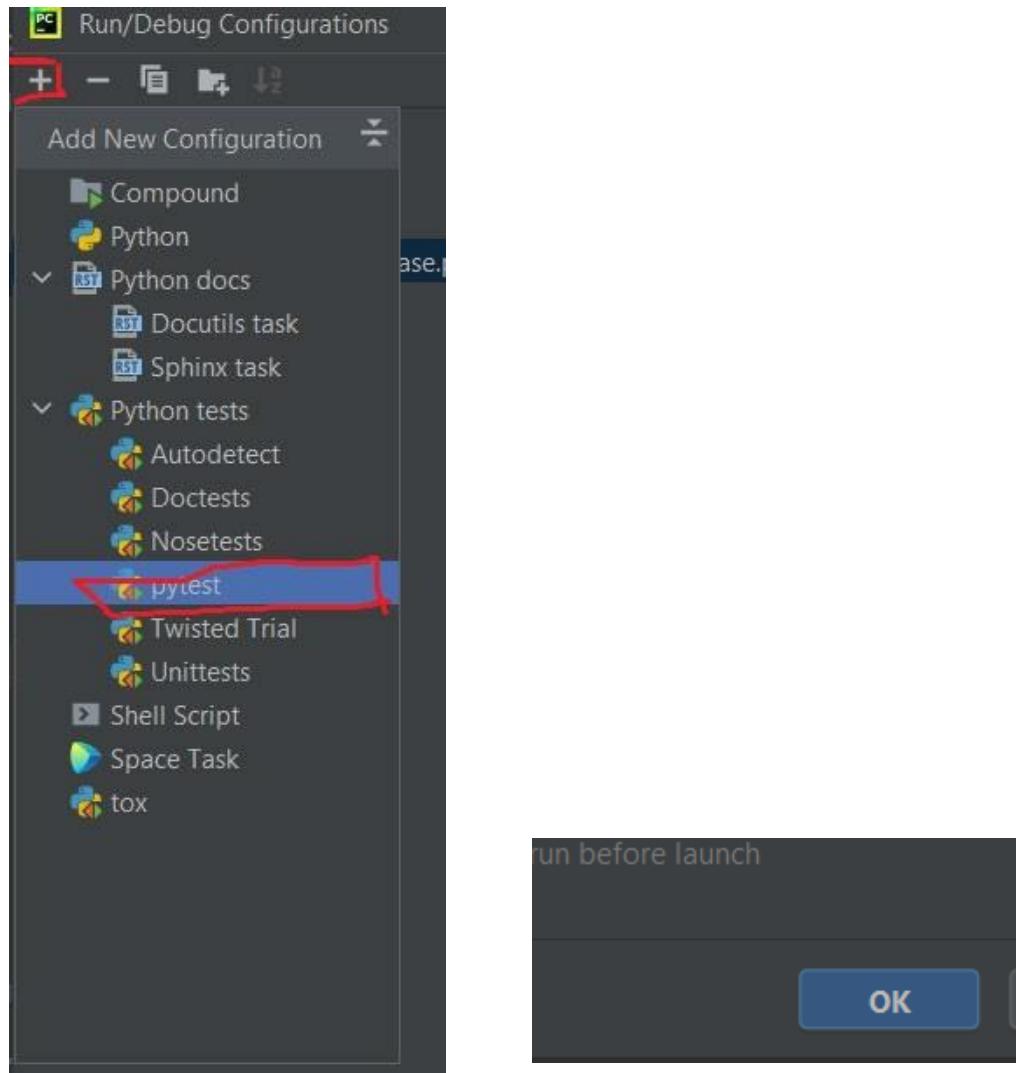
- Run through test runner
- Run through command prompt

Run through test runner

- 1.Create the Python file with test name as prefix
- 2.Create a method with test name as prefix
- 3.Import Pytest
- 4.Click Edit configuration

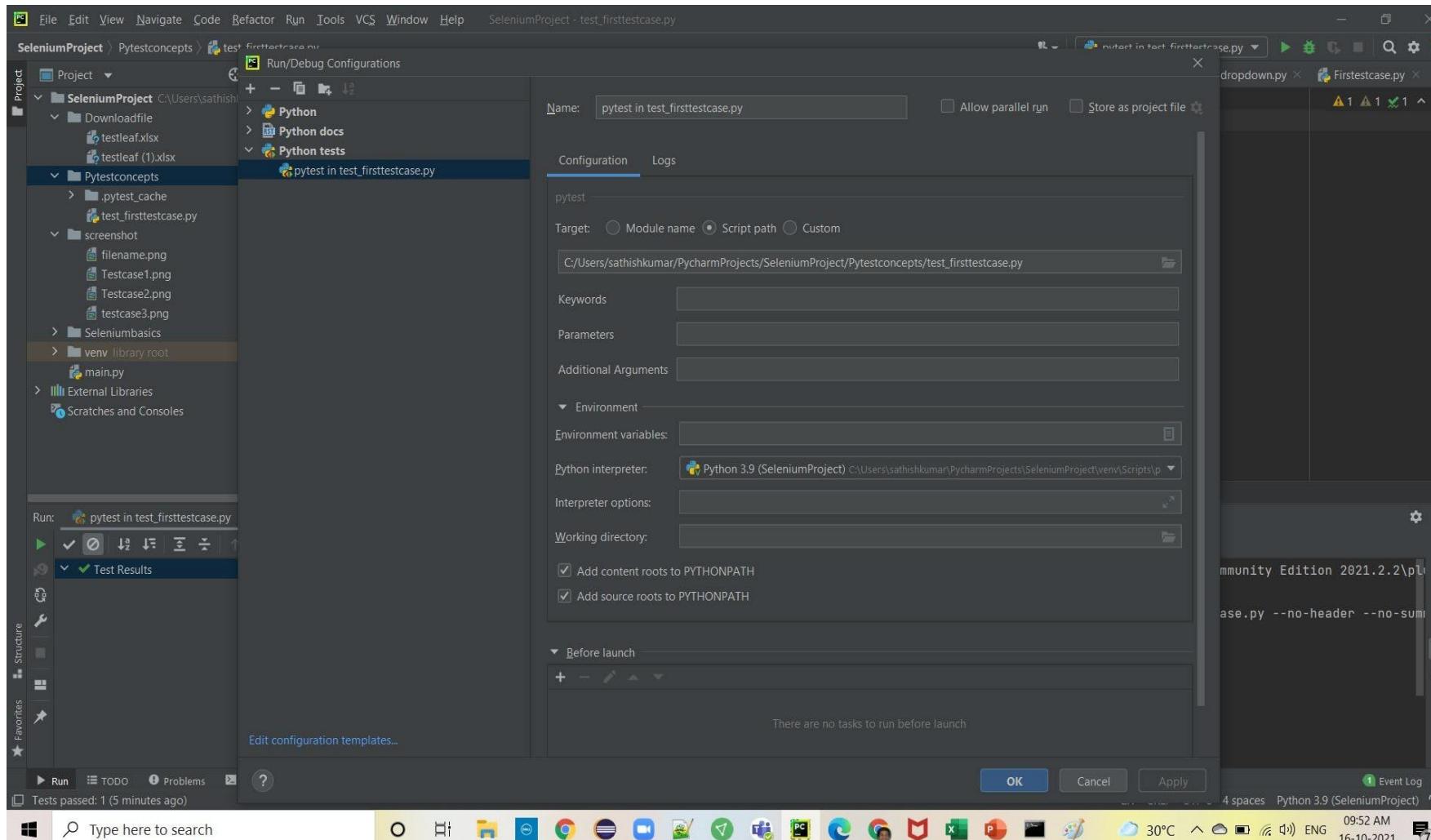


Pytest – Testing Framework



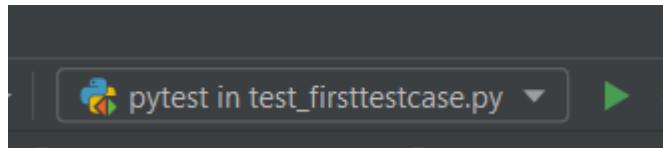
Pytest – Testing Framework

Click Edit configuration and select the respective file under the Target and click ok



Pytest

- Click run button to see the execution



- Output should be like this

```
Tests passed: 1 of 1 test - 0 ms

C:\Users\sathishkumar\PycharmProjects\SeleniumProject\venv\Scripts\python.exe "C:\Program Files\JetBrains\PyCharm Community Edition 2021.2.2\pl
Testing started at 09:46 AM ...

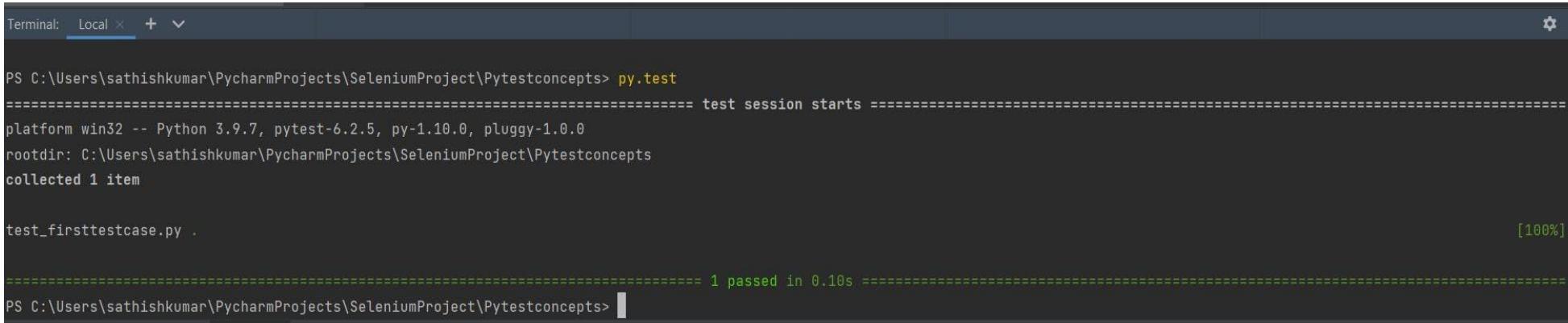
Launching pytest with arguments C:/Users/sathishkumar/PycharmProjects/SeleniumProject/Pytestconcepts/test_first testcase.py --no-header --no-sum

=====
test session starts =====
collecting ... collected 1 item

test_first testcase.py::test_first testcase PASSED [100%] This is test case 1
```

Pytest – command prompt

- Select the file package and right click → open in terminal
- Type **py.test / pytest**
- Then the execution will be completed as like the test runner



The screenshot shows a terminal window titled "Terminal: Local". The command "py.test" is run in the directory "C:\Users\sathishkumar\PycharmProjects\SeleniumProject\Pytestconcepts". The output shows the test session starting, the platform (win32, Python 3.9.7, etc.), the root directory, and that one item was collected from "test_first testcase.py". It then shows the test running, with a progress bar at 100% completion and 1 passed test in 0.10s. The command "py.test" is shown again at the end.

```
PS C:\Users\sathishkumar\PycharmProjects\SeleniumProject\Pytestconcepts> py.test
=====
platform win32 -- Python 3.9.7, pytest-6.2.5, py-1.10.0, pluggy-1.0.0
rootdir: C:\Users\sathishkumar\PycharmProjects\SeleniumProject\Pytestconcepts
collected 1 item

test_first testcase.py .

=====
1 passed in 0.10s
PS C:\Users\sathishkumar\PycharmProjects\SeleniumProject\Pytestconcepts>
```

You can type the following command

- **Pytest -v** (to see all the logs as like in the test runner)
- **Pytest -v -s** (with the print statement details in the output)
- **Pytest filename.py -v -s** (Run the Test cases in the specific file)
- **Pytest -k testcasenamecontainsword -v -s** (Run the test which contains the testcase name)

Pytest – command propmt

Note :

- Test case name should be unique or else the corresponding should not execute
- Test case name =Function name

Pytest provides two ways to run the subset of the test suite.

- Select tests to run based on substring matching of test names.

Pytest –k testcasenamecontainsword –v –s (Run the test which contains the testcase name)

- Select tests groups to run based on the markers applied.

Py.test –m groupname –v -s

Group the test cases

`@pytest.mark.groupname` -----→ just above each testcase to group in to that

Run in command prompt by following command

Py.test –m groupname –v -s

Pytest – Mark and Fixtures

There are 2 types of groups

- 1. Custom mark → user defined group name
- 2. Predefined mark → predefined group name

Below are few custom groups

`@pytest.mark.skip` – to skip that test case

`@pytest.mark.xfail` – this will run but will not included in the report

Fixtures :

Fixtures are functions, which will run before each test function to which it is applied. Fixtures are used to feed some data to the tests such as database connections, URLs to test and some sort of input data. Therefore, instead of running the same code for every test, we can attach fixture function to the tests and it will run and return the data to the test before executing each test.

`@pytest.fixture()`

Pytest - Fixtures

- A test function can use a fixture by mentioning the fixture name as an input parameter.
- This is basically used to execute few code before each test
- E.g
- import pytest
- @pytest.fixture
- def **input_value**():
- input = 39
- return input
- def test_divisible_by_3(**input_value**):
- assert input_value % 3 == 0

Pytest – Fixtures

- If we want to execute certain action post the test case then we have use the following under yield in the fixtures

- @pytest.fixture
- def **input_value()**:

```
    input = 39
```

```
    return input
```

Yield

```
    Print("I am post yield")
```

Pytest – Fixtures (Conftest-file)

- We can define the fixture functions in this file to make them accessible across multiple test files.

File name should be conftest.py

It should contains only fixtures so the each pytest file should not have the fixtures

How to optimize the fixtures in testcase

Create a class underneath that create a test cases

Now provide the self as parameter to each test case

Then define the following syntax on above class

`@pytest.mark.usefixtures("fixturefilename")`

Pytest – Fixtures (Conftest-file)

How to run the fixture before the class but not each test cases then

Go to conftestfile and update as below

```
@pytest.fixture(scope="class")
```

Data Driven Framework :

Fixture is also help to pass the data in to the test cases

If u want to pass a set of data with one time then use the below in fixtures

```
@pytest.fixture()
```

```
def dataload()
```

```
    return["sathish","kumar","1990"]
```

Pytest – Fixtures (Conftest-file) -parameterization

Go to the class where your test cases contains and use the below syntax just above the class

```
@pytest.mark.usefixtures("dataload")  
Class Test_classname  
    def test_testcase1(self,dataload):  
        print(dataload[0])
```

Multiple data set: (pick each test on each run)

Under the Fixtures

```
@pytest.fixture(params=["chrome","firefox","IE"])  
def crbrws(request)  
    return request.param
```

Note : here we have to provide request so that we can get the param value

Pytest – Fixtures (Conftest-file) -parameterization

Go to the class where your test cases contains and use the below syntax just above the class

Class classname

```
def test_TestCase1(self, crbrws):  
    print(crbrws)
```

Other way :

Multiple data set: (pick each test on each run)

Under the Fixtures

```
@pytest.fixture(params=[("chrome","sathishkumar","password"),("firefox","s  
athish","kumar"),("IE","thirdvalue")])  
def crbrws(paramvalue)  
    return paramvalue.param
```

Pytest – Fixtures (Conftest-file) -parameterization

Go to the class where your test cases contains and use the below syntax just above the class

```
@pytest.mark.fixture("crbrws")  
Class Test_classname  
    def test_testcase1(self , crbrws):  
        print(crbrws[1])
```

Reports

1. Html

Install pytest-html

To run

Pytest --html=reportname.html

2 : Html – Reporter

Install pytest-html-reporter

To run

Pytest –html-report=reportname.html

To create the report in specific folder

Pytest --html-report=./Report/reportname.html

To open this report in a browser then do as below,

Right click the HTML report – Open in → Browser → chrome

Run the program based on commanline

So in the commanline you have to use the below command

```
Py.test -- browser_name browsername
```

For set the browser value :

```
def pytest_addoption(parser):
    parser.addoption(
        "--cmdopt", action="store", default="type1", help="my option: type1 or
type2"
    )
```

For get the browser value:

```
@pytest.fixture
def cmdopt(request):
    return request.config.getoption("--cmdopt")
```

Page object Model Framework

- What Is Page Object Model?
- Page Object Model is a design pattern where the core focus is on reducing code duplication and minimization of the effort involved in code update/maintenance. Under the Page Object Model, page classes are created for all the webpages that are a part of the Automation Under Test (AUT).
- This makes the code more modular since locators/elements used by the test suites/test scenarios are stored in a separate class file & the test cases (that contain the core test logic) are in a different file. Hence, any change in the web UI elements will require minimal (or no changes) in the test scenarios since locators & test scripts are stored separately.
- Implementation based on the Page Object Model (POM) contains the below crucial elements:
- **Page Object Element (Page Class/Page Object)** – The Page Class is an object repository for the WebElements/Web UI Elements of the web-pages under test. It also contains an implementation of the interfaces/methods to perform operations on these web elements. It is loosely based on the fundamentals of Object-Oriented Programming.
- **Test Cases** – As the name suggests, test cases contain the implementation of the actual test scenarios. It uses page methods/methods in the page class to interact with the page's UI elements. If there is a change in the UI of the web page, only the Page Class needs to be updated, and the test code remains unchanged.

Page object Model Framework

- Why Use Page Object Model (POM)?
- As a software project progresses, the complexity of the development code and the test code increases. Hence, a proper project structure must be followed when developing automation test code; else, the code might become unmanageable.
- A web product (or project) consists of different web pages that use various WebElements (e.g., menu items, text boxes, checkboxes, radio buttons, etc.). The test cases will interact with these elements, and the code complexity will increase manifold if Selenium locators are not managed in the right way.
- Duplication of source code or duplicated locators' usage can make the code less readable, resulting in increased overhead costs for code maintenance. For example, your team needs to test the login functionality of an e-commerce website. Using Automation testing with Selenium, the test code can interact with the web page's underlying UI or locators. What happens if the UI is revamped or the path of the elements on that page change? The automation test scenarios will fail as the scenarios would no longer find those locators on the web page.
- Suppose you follow the same test development approach for more web-pages. In that case, a considerable effort has to be spent on updating the locators that might be scattered (or duplicated) across different files. This approach is also error-prone as developers need to find & update the path of the locators. In such scenarios, the Page Object Model can be advantageous as it offers the following:
 - Eases the job of code maintainability
 - Increases the readability and reusability quotient

Page object Model Framework

- Advantages Of Page Object Model
- Now that you are aware of the basics of the Page Object Model, let's have a look at some of the core advantages of using that design pattern:
- **Increased Reusability** – The page object methods in different POM classes can be reused across different test cases/test suites. Hence, the overall code size will reduce by a good margin due to the increase in the page methods' reusability factor.
- **Improved Maintainability** – As the test scenarios and locators are stored separately, it makes the code cleaner, and less effort is spent on maintaining the test code.
- **Minimal impact due to UI changes** – Even if there are frequent changes in the UI, changes might only be required in the Object Repository (that stores the locators). There is minimal to no impact on the implementation of the test scenarios.
- **Integration with multiple test frameworks** – As the test implementation is separated from the repository of page objects (or page classes), we can use the same repository with different test frameworks. For example, Test Case – 1 can use the Robot framework, Test Case – 2 can use the pytest framework, etc. single test suite can contain test cases implemented using different test frameworks.

Page object Model Framework