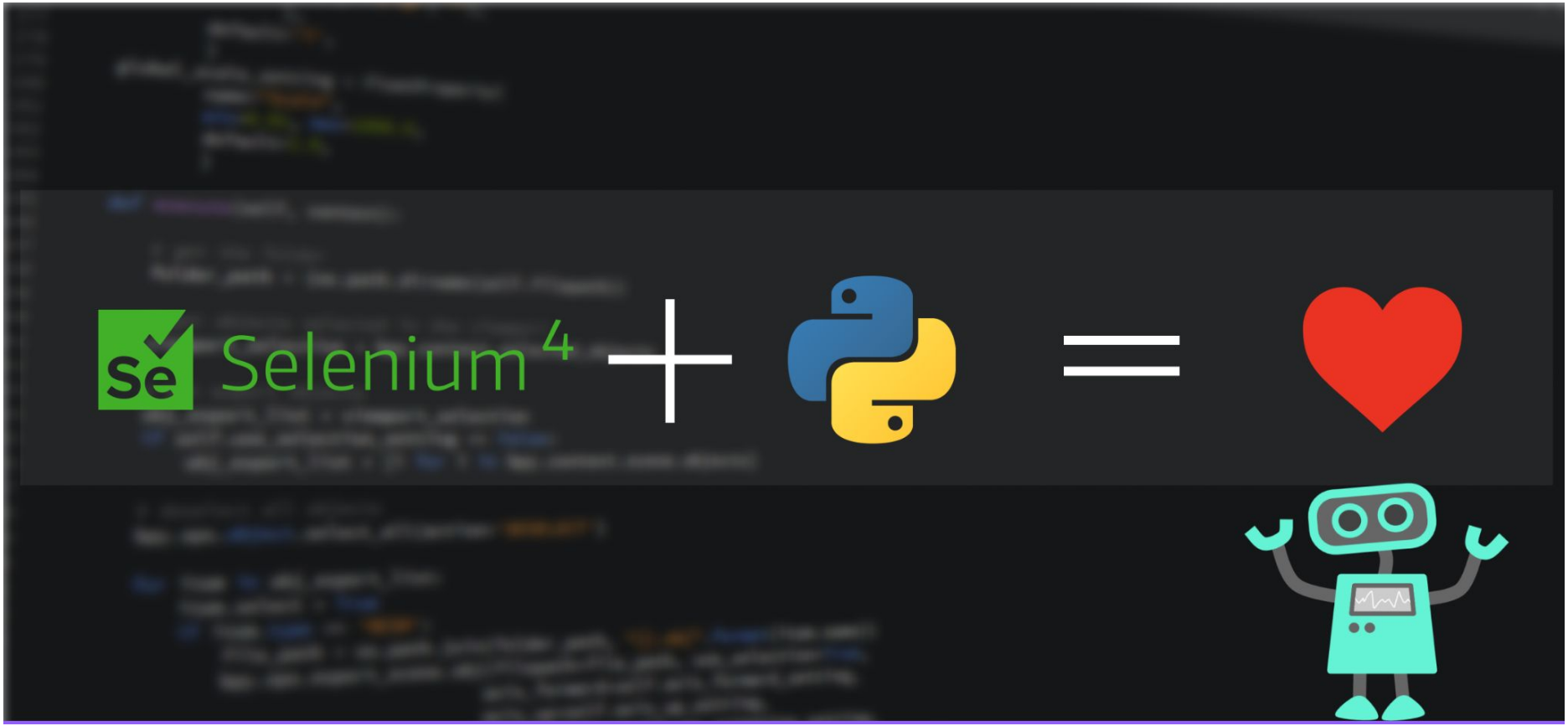
SDJ INTERNATIONAL COLLEGE

Selenium Python
Unit 4

– By Tushar Sir



– By Tushar Sir

findElement & findElements

- Whenever you want to interact with a web page, we require a user to locate the web elements.
- We usually start by finding the HTML elements on the page whenever we plan to automate any web application using WebDriver. Selenium WebDriver defines two methods for identifying the elements, they are findElement and findElements.
    - **findElement:** This command is used to uniquely identify a web element within the web page.
    - **findElements:** This command is used to uniquely identify the list of web elements within the web page.
- There are multiple ways to uniquely identify a web element within the web page such as ID, Name, Class Name, LinkText, PartialLinkText, TagName, and XPath.

– By Tushar Sir

Difference between findElement & findElements

FindElement() Method:

- This command is used to access any single element on the web page.
- It will return the object of the first matching element of the specified locator.
- It will throw NoSuchElementException when it fails to identify the element.
- Returns the first matching web element if the locator discovers.
- Throws NoSuchElementException if the element is not found.
- Detects a unique web element.

– By Tushar Sir

Difference between findElement & findElements

FindElements() Method:

- This command is used to uniquely identify the list of web elements within the web page.
- The usage of this method is very limited.
- If the element doesn’t exist on the page then, then it will return value with an empty list.
- Returns a list of multiple matching web elements.
- Returns an empty list if no matching element is found.
- Returns a collection of matching elements.

– By Tushar Sir

Selenium findElement Command

- Find Element command takes in the By object as a parameter and returns an object of type WebElement.
- By object can be used with various locator strategies such as ID, Name, ClassName, link text, XPath, etc.
- **Syntax:**

WebElement elementName = driver.findElement(By.LocatorStrategy("LocatorValue"));

- Locator Value is the unique value using which we can identify the web element. It is the core responsibility of developers and testers to make ensure that web elements are uniquely identified by using certain properties such as ID or Name.
- **Example:**

  WebElement login= driver.findElement(By.linkText("Login"));

– By Tushar Sir

Selenium findElement Command

Locator Strategy can be any of the following values:

- ID
- Name
- Class Name
- Tag Name
- Link Text
- Partial Link Text
- XPath

– By Tushar Sir

Selenium findElements Command

- Selenium findElements command takes in By object as the parameter and returns a list of web elements. It returns an empty list if no elements found using the given locator strategy and locator value.
- **Syntax :**

```
List<WebElement> elementName = driver.findElements(By.LocatorStrategy("LocatorValue"));
```

- **Example :**

```
List<WebElement> listOfElements = driver.findElements(By.xpath("//div"));
```

– By Tushar Sir

How to use Selenium findElement Command

The following application is used for demo purpose: https://www.irctc.co.in/nget/user-registration Scenario

- Open the https://www.irctc.co.in/nget/user-registration for AUT
- Find and click radio button

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

public class SeleniumFindElement {
       public static void main (String [] args){

     System.setProperty("webdriver.chrome.driver","D:\\Drivers\\chromedriver.exe");
     WebDriver driver= new ChromeDriver();
     driver.manage().window.maximize():
     driver.get(:"https://www.irctc.co.in/nget/user-registration");

//Find the radio button for "Male" by using ID and click on it
driver.findElement(By.id("M")).click();

       }
}
```

– By Tushar Sir

Locators in Selenium

- Locators in Selenium come into action in the fourth step above after the Selenium WebDriver is initialized and loaded the webpage to be tested.
- A locator enables testers to select an HTML DOM element to act on.
- Knowing how to use different locators correctly is key to building better automation scripts.
- If the test script is not able to identify which element it needs to interact with, the test will fail before it can begin.
- Selenium provides support for these 8 traditional location strategies in WebDriver.

– By Tushar Sir

Locators in Selenium

Locator	Description
class name	Locates elements whose class name contains the search value (compound class names are not permitted)
css selector	Locates elements matching a CSS selector
id	Locates elements whose ID attribute matches the search value
name	Locates elements whose NAME attribute matches the search value
link text	Locates anchor elements whose visible text matches the search value
partial link text	Locates anchor elements whose visible text contains the search value. If multiple elements are matching, only the first one will be selected.
tag name	Locates elements whose tag name matches the search value
xpath	Locates elements matching an XPath expression

– By Tushar Sir

Dynamic Xpath

- XPath, also known as XML Path, is one of Selenium WebDriver's most commonly used locators for navigating through a page's HTML structure.
- It can be used to locate any element in a web page using HTML DOM structure in HTML and XML documents.
- XPath is intended to allow XML document navigation to select individual elements, attributes, or other parts of an XML document for specific processing.
- For example, XPath generates reliable locators, but it is slower in terms of performance than CSS Selector.
- The language XPath is used to select elements in an HTML page. Using XPath, you can find any element on a page based on its tag name, ID, CSS class, and so on. In Selenium, there are two types of XPath.

– By Tushar Sir

Dynamic Xpath

Different Approaches to Find Elements using Dynamic XPath in Selenium: -

- **Absolute XPath:**
  - It is the most direct way to find the element, but the disadvantage of “absolute XPath” is that if the element’s path changes, that particular XPath fails.
  - The critical feature of XPath is that it begins with a single forward-slash (/), indicating that you can select an element from the root node using Dynamic XPath.
- **Relative XPath:**
  - A relative XPath is one in which the path begins at a node of your choice rather than the root node.
  - The benefit of using relative XPath is that you don’t have to specify the entire XPath; instead, you can begin in the middle or somewhere in between.

– By Tushar Sir

11 Unique Ways to Create Dynamic XPath in Selenium

Using Single Slash:

- This mechanism is also known as Absolute XPath element discovery. A single slash is used to create an XPath with an absolute path, i.e., the XPath is designed for beginning selection from the document node/start node/parent node.
- **Syntax:**

  html/body/div[1]/div[2]/div[2]/div[1]/form/div[1]/div/div[1]/div/div/input[1]

Using Double Slash:

- This mechanism is also referred to as finding elements with Relative XPath. A double slash is used to create XPath with a relative path, which means that the XPath can begin selection from anywhere in the document. Then, look for the preceding string across the entire page (DOM).
- **Syntax:**

  //form/div[1]/div/div[1]/div/div/input[1]

– By Tushar Sir

11 Unique Ways to Create Dynamic XPath in Selenium

Utilizing a Single Attribute:

- The syntax could be written in two ways, as shown below. HTML Tag inclusion or exclusion. If you want to exclude HTML tags, you must use *.
- **Syntax:**

  //<HTML tag>[@attribute_name='attribute_value']

  or

  //[@attribute_name='attribute_value']*

– By Tushar Sir

11 Unique Ways to Create Dynamic XPath in Selenium

Using Multiple Attributes:

- **Syntax:**

  //<HTML tag>[@attribute_name1='attribute_value1'][@attribute_name2='attribute_value2]

  or

  //[@attribute_name1='attribute_value1'][@attribute_name2='attribute_value2]*

– By Tushar Sir

11 Unique Ways to Create Dynamic XPath in Selenium

Using AND:

- **Syntax:**

  //<HTML tag>[@attribute_name1='attribute_value1' and @attribute_name2='attribute_value2]

  or

  //[@attribute_name1='attribute_value1' and @attribute_name2='attribute_value2]*

– By Tushar Sir

11 Unique Ways to Create Dynamic XPath in Selenium

Using OR :

- **Syntax :**

  //<HTML tag>[@attribute_name1='attribute_value1' or @attribute_name2='attribute_value2]

  or

  //[@attribute_name1='attribute_value1' or @attribute_name2='attribute_value2]*

– By Tushar Sir

11 Unique Ways to Create Dynamic XPath in Selenium

Using contains() :

- **Contains()** is a method for identifying an element that changes dynamically and when familiar with some part of the element's attribute value.
- When familiar with the value of an element's attribute (beginning with the specified text), we can use the starts-with() method to identify it.
- **syntax :**

  //<HTML tag>[contains(@attribute_name,'attribute_value')]

  or

  //[contains(@attribute_name,'attribute_value')]*

– By Tushar Sir

11 Unique Ways to Create Dynamic XPath in Selenium

use of text () :

- This mechanism is used to find an element based on a web page's text.
- **Last()** selects the last element (of the specified type) from all input elements.
- **Syntax :**

  //[text()='New look for sign-in coming soon']*

  or

  //[text()='New look for sign-in coming soon']*

– By Tushar Sir

11 Unique Ways to Create Dynamic XPath in Selenium

Using position() :

- The element is chosen from among all input elements based on the position number provided.
- In the following XPath, [@type=’text’] will locate a text field, and function [position()=2] will identify a text field in the second position from the top.
- **Syntax :**

  findElement(By.xpath("(//input[@type='text'])[position()=2]"))

  or

  findElement(By.xpath("(//input[@type='text'])[2]"))

– By Tushar Sir

11 Unique Ways to Create Dynamic XPath in Selenium

Using an index :

- We could get to the nth element by putting the index position in square brackets. Then, we were able to identify the Last Name field using the XPath below.
- **Syntax :**

  findElement(By.xpath("//label[2]"))

Using previous XPath axes :

- Except for ancestors, attribute nodes, and namespace nodes, this selects all nodes that appear before the current node in the document.

– By Tushar Sir

Which is highly efficient as well as reliable, dynamic XPATH or DYNAMIC CSS selector?

- Most of the automation engineers on this planet would recommend using CSS Selector to find web elements.
- Finding changing web elements through CSS selector are faster than dynamic XPath.
- Followings are few of logic in bullets:
  - Each browser has different engines for XPath which make elements inconsistent.
  - CSS is always same irrespective of browsers.
  - Sometimes XPath becomes complex, whereas CSS is always simple.
  - Dynamic CSS Selector performs better.

– By Tushar Sir

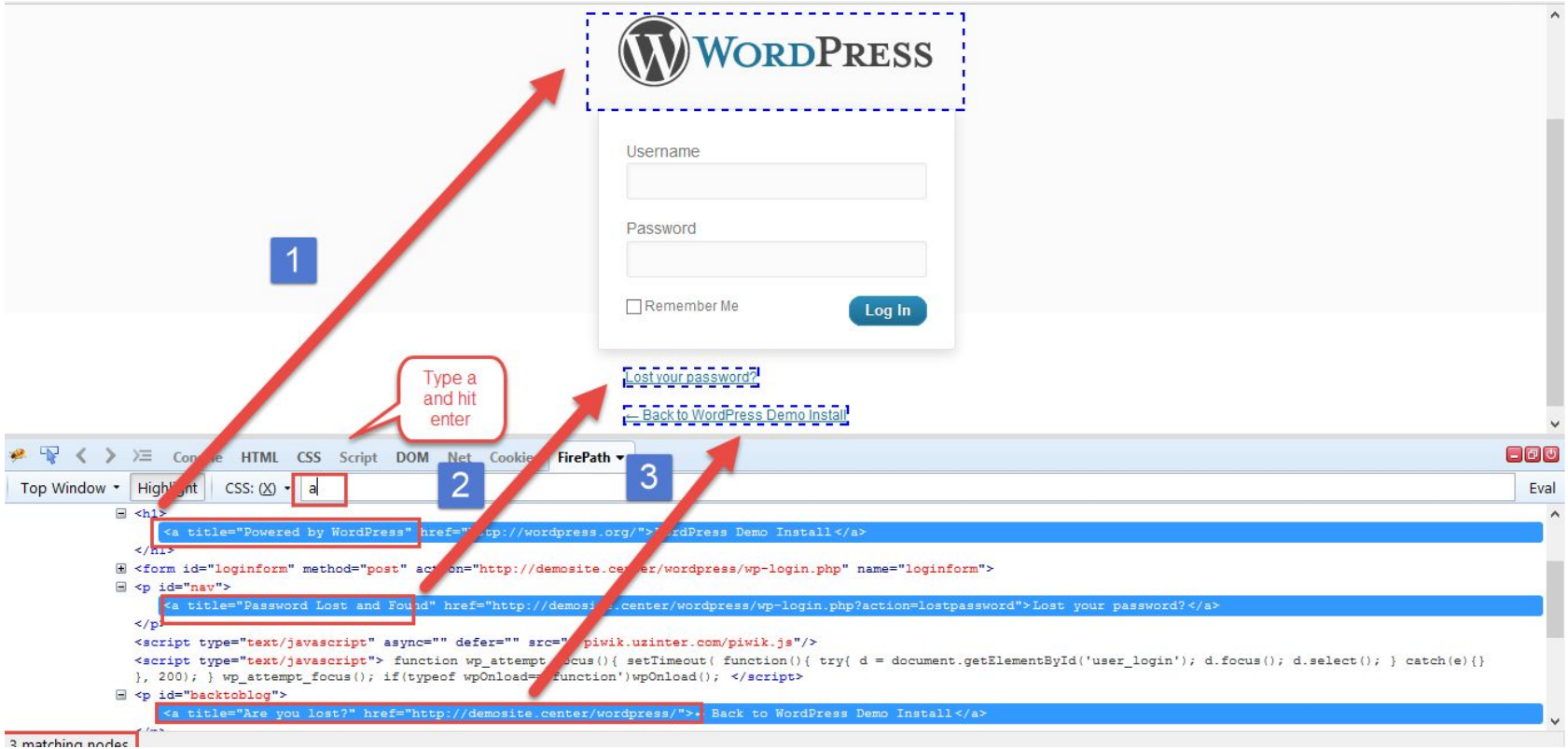
Dynamic CSS

- Symbol used while writing CSS selector in Selenium Webdriver :

	attribute	Symbol used
1	Using id	use # symbol
2	Using class name	use . symbol
3	Using attribute	tagname[attribute='value']
4	Using multiple attribute	tagname[attribute1='value1'] [attribute2='value2']
5	Contains	* symbol
6	Starts with	^ symbol
7	Ends with	$ symbol

– By Tushar Sir

Dynamic CSS

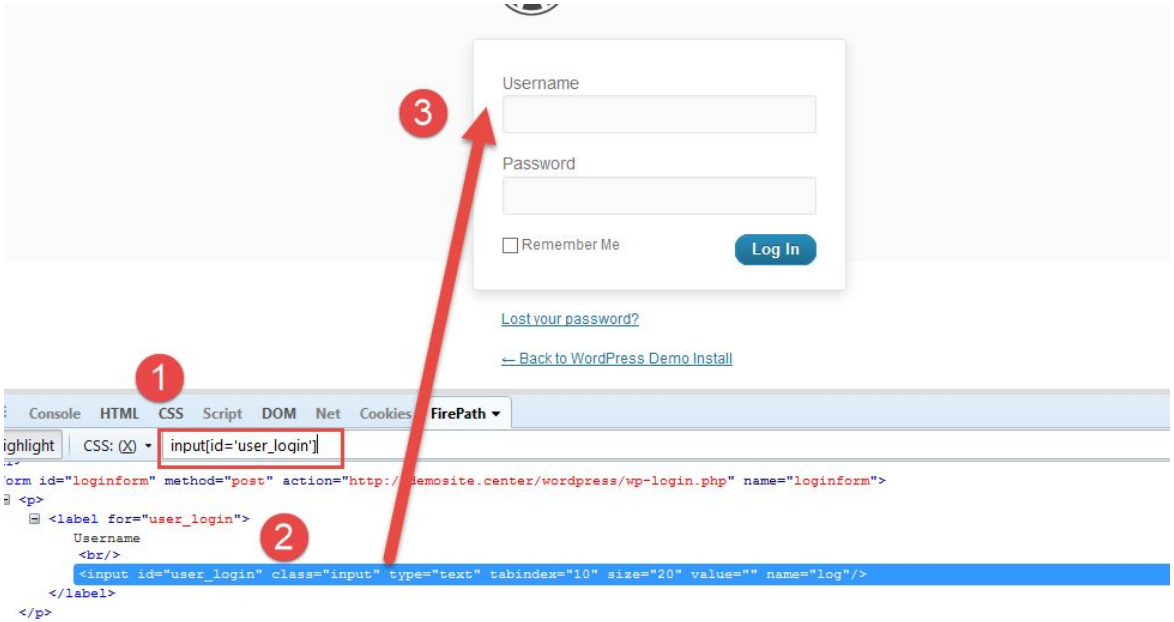


– By Tushar Sir

Dynamic CSS

Find CSS Selector using Single Attribute :

- **Syntax-** *tagname[attribute='value']*
- **Example-** *input[id='user_login']*

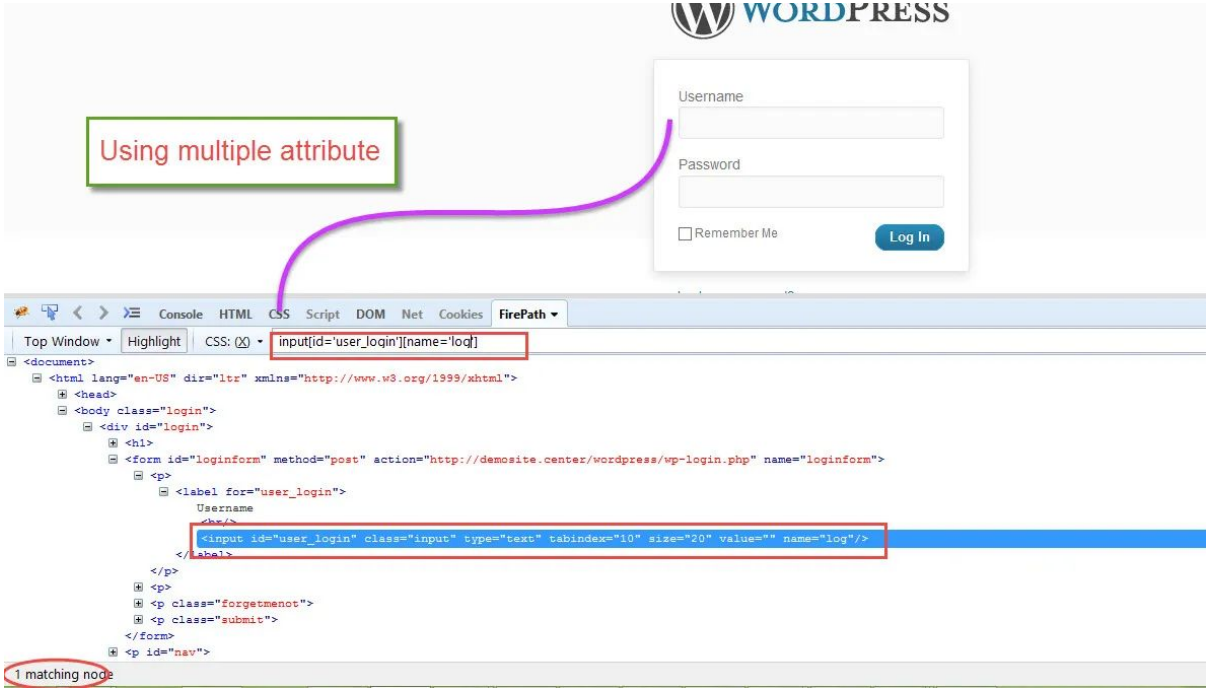


– By Tushar Sir

Dynamic CSS

Find CSS using Multiple attributes :

- **Syntax-** *tagname[attribute1='value1'][attribute2='value2']*



– By Tushar Sir

Dynamic CSS

Find CSS using id and Class name :

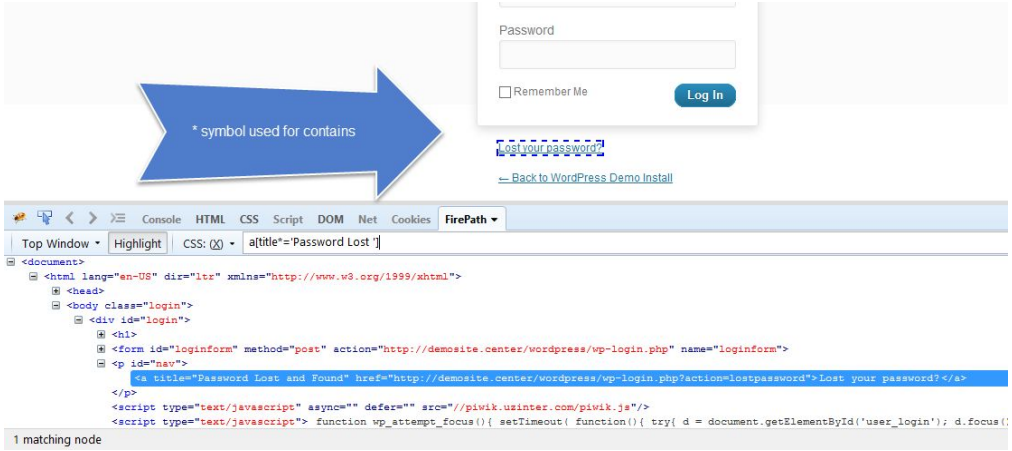
- **Syntax using ID -** *tagname#id*
- **Syntax using Class name -** *tagname.classname*
- Class name generally will not be unique so always try class name with some other attributes.

– By Tushar Sir

Dynamic CSS

Find CSS using contains :

- In CSS we will use * symbol to check particular attribute contains that value or not.
- **Syntax-** *tagname[attribute*='value']*
- In below example it will partially check if link title contains “Password lost ” if yes then it will match.
- I have taken only link in this example but you can take any tag and any attribue.

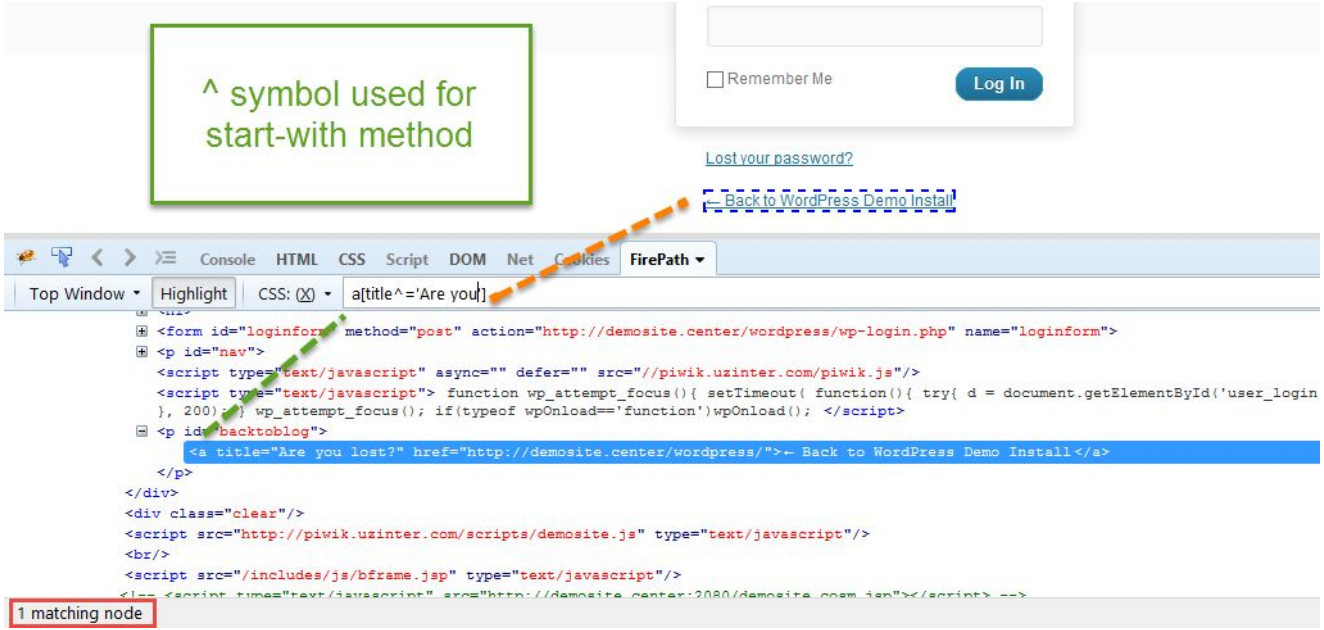


– By Tushar Sir

Dynamic CSS

Find CSS using Start-with :

- **Syntax-** *tagname[attribute^='value']*

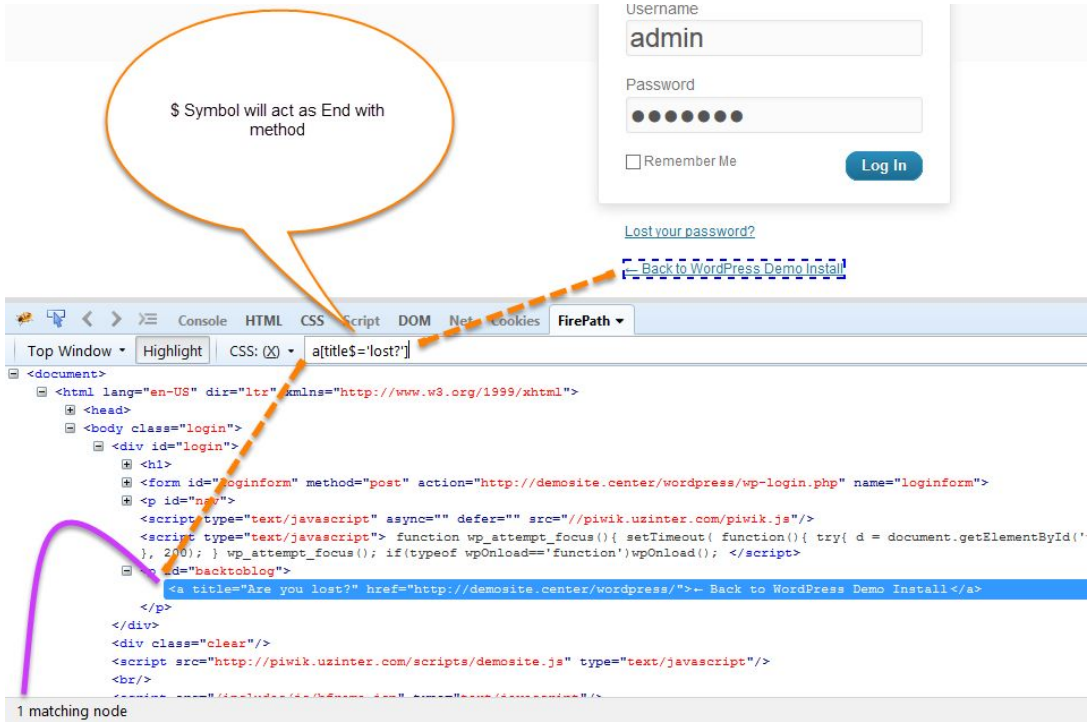


– By Tushar Sir

Dynamic CSS

Find CSS using ends-with :

- **Syntax-** *tagname[attribute$='value']*



– By Tushar Sir

Handling dropdowns

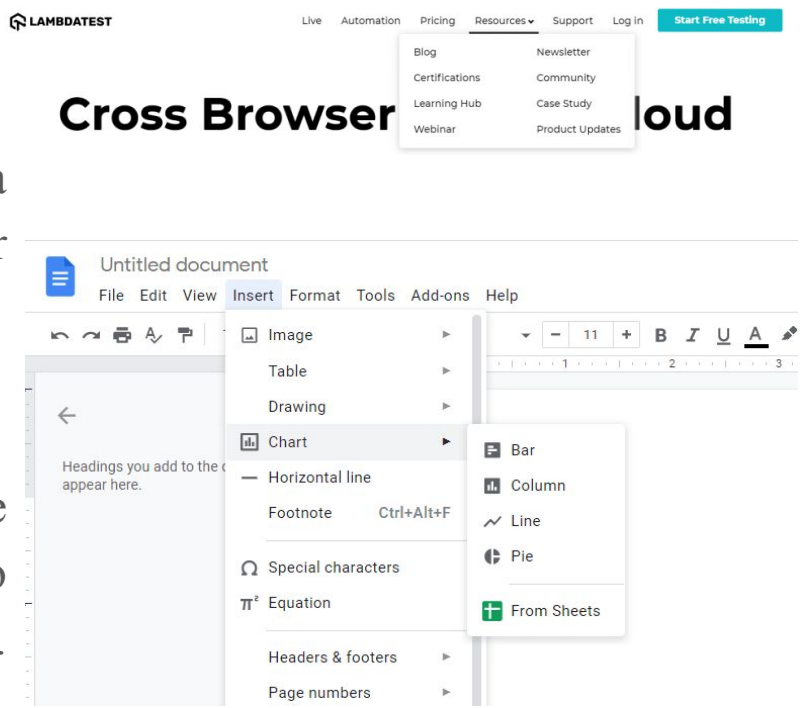
- Dropdowns are an integral part of modern websites. And like any other HTML element, these need to be tested as well, especially when you are performing automated browser testing.
- UI/UX designers love dropdown elements, but it’s the automation testing engineers who get to play with it.
- Therefore, it becomes important to know how to handle dropdowns in Selenium WebDriver when you are handling access forms or testing websites.
- Designers prefer using aesthetically appealing dropdown menus or boxes. Mostly because dropdowns tend to utilize the available screen space frugally.
- It is useful when you want only specific inputs from users and not some crap data. However, dropdown design bloopers could be a serious turn-off for users.

– By Tushar Sir

Different Types of Dropdowns

In HTML, we encounter four types of dropdown implementations:

1. **Dropdown Navigation Options :**
   - These are usually encountered in NavBar of a website with dropdown links to other webpages.
2. **Dropdown Command Options :**
   - Like navigation dropdown options, these are found at the top, but these are meant to perform some action on the active page itself. Example – Google Docs Menu Bar.



– By Tushar Sir

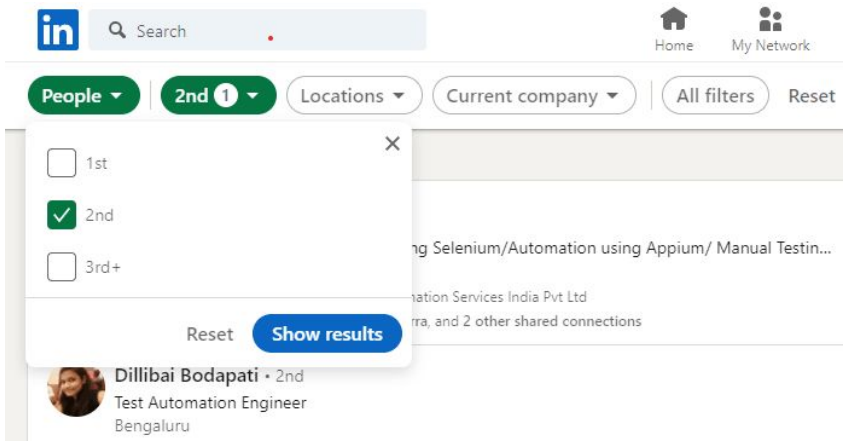
Different Types of Dropdowns

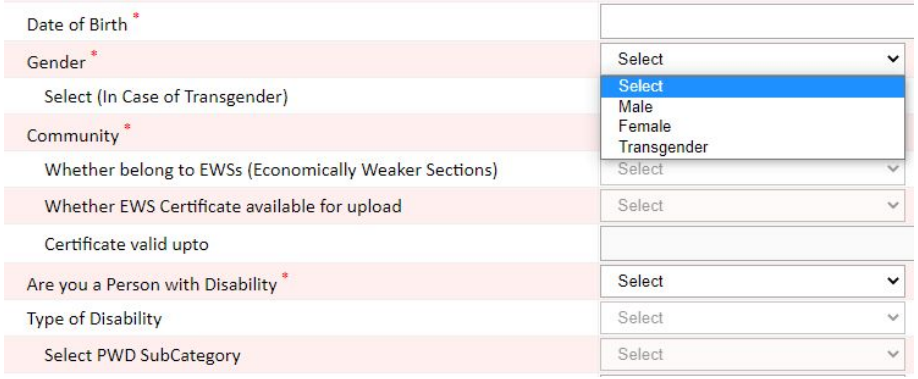
3. Attribute Selection Dropdown Options :

- These are commonly used to implement search filtering features and customization options like changing the color-template or language of a website.

4. Form Filling Dropdowns Options :

- These sort of dropdown options are used for registration forms or product/service booking forms.





– By Tushar Sir

The Need for a Dropdown

- The main advantage of using a dropdown is that it saves your apps/web pages from unnecessary space consumption by containing all the options within it.
- It eliminates the problems like mistyping, misspelling, or wrong input from the users.
- It helps find particular content on a web page without scrolling up and down.
- Save users' time on a large website. You can access your targeted content without wasting any time.
- It sets the limit of options to choose from and helps users easily navigate.
- Preserve screen space from unwanted content.
- Also helps to navigate to another page.

– By Tushar Sir

How to Handle Dropdowns in Selenium Webdriver

- Using CSS and/or XPATH selectors in Selenium Python can test a variety of dropdown components.
- However, Selenium WebDriver includes the SELECT class as an extra functionality.
- It can be used to automate interactions with dropdown menus created with the html tag “select”.

What is ‘Select’ Class in Selenium WebDriver?

- Selenium provides the Select class to implement the HTML Select elements.
- The Select class in Selenium is an ordinary Java class that you can create as a new object using the keyword New and specifying a web element’s location.

– By Tushar Sir

How to Handle Dropdowns in Selenium Webdriver

Now let's see in **Selenium WebDriver**, different **SELECT** class functions : -

class selenium.webdriver.support.select.Select(webelement)

- Under the hood, the Selenium select class checks if the HTML web element we are using it with is a <select> tag or not. If not, Selenium WebDriver throws

  UnexpectedTagNameException

- **Note:** Selenium 'Select' is only meant to be used with html <select> tag and none else."

– By Tushar Sir

How to Handle Dropdowns in Selenium Webdriver

The selenium SELECT class comprises of following sub-functions:

options(self) :

- This function finds all <options> tags within the target <select> tag.

all_selected_options(self)

- This function loops over all the options and checks if selected using is_selected() method and returns a list of selected “options.”

first_selected_option(self)

- This is similar to the above function, loops over all available options under the <select> tag but returns as soon as it finds the first is_selected() <option>

– By Tushar Sir

How to Handle Dropdowns in Selenium Webdriver

select_by_value(self, value) :

- This function makes use of CSS selector to evaluate value attributes of web elements. It returns all the <options> with matching values.

select_by_index(self, index) :

- This SELECT class function of Selenium evaluates the index of <option> tags by using get_attribute("index") and returns the matching option.

select_by_visible_text(self, text) :

- This function is implemented using XPATH and multiple if-else. Selects the option elements which contain string equal to text attribute between opening and closing option tags i.e, <option>dummy_text_attribute</option>.

– By Tushar Sir

How to Handle Dropdowns in Selenium Webdriver

Similarly, there are functions to deselect the selected options too –

- *deselect_by_index(self, index)*
- *deselect_by_value(self, value)*
- *deselect_all(self)*
- *deselect_by_visible_text(self, text)*

Different methods of Select class:

- Selection an option from the dropdown menu by **INDEX.**
- Selection an option from the dropdown by **VISIBLE TEXT.**
- Selection an option from the dropdown menu by **VALUE.**

– By Tushar Sir

How to Handle Dropdowns in Selenium Webdriver

- The two functions which are internally used by Selenium WebDriver's SELECT class to implement above functions for "selecting or deselecting" options are :

```
def _setSelected(self, option):
    if not option.is_selected():
        option.click()

def _setSelected(self, option):
    if not option.is_selected():
        option.click()
```

– By Tushar Sir

Selecting a Dropdown option in Selenium WebDriver with matching text

- To automate testing of **<option>** selection, based on the criteria whether it matches a particular text string or not, we use the ***select_by_visible_text(self, text)*** method of a **SELECT** class in Selenium.
- **Example :**

```
dropdown1 = Select(driver.find_element_by_id('lang1'))
dropdown1.select_by_visible_text('Java')


dropdown = Select(driver.find_element_by_id('lang'))
dropdown.select_by_value('python')
```

– By Tushar Sir

Selecting a Dropdown option in Selenium WebDriver using index

- Javascript provides a DOM property to select <options> using index –

  document.getElementById("myCourses").selectedIndex = "3";

- Similarly, Selenium Python provides a method i.e., select_by_index(self, index) to automate selection of option(s). In “Demo for individual select” section on our test webpage, we can select C# by passing an index of ‘3’.

```
dropdown = Select(driver.find_element_by_id('lang1'))
dropdown.select_by_index(3)
```

– By Tushar Sir

Using Selenium WebDriver for handling Dropdowns with multiple select options enabled

- Multiple option selection is enabled by adding an attribute named “multiple” to <select> elements.
- To check if a <select> dropdown allows multiple selection, we can use xpath or get attribute or both.
- Internally, Selenium in its __init__() constructor use following approach to evaluate if drop-down list allow multiple option selection –

```
1 self._el = webelement
2 multi = self._el.get_attribute("multiple")
3 self.is_multiple = multi and multi != "false"
```

– By Tushar Sir

Using Selenium WebDriver for handling Dropdowns with multiple select options enabled

- We first identify a <select> webElement using xpath or CSS selectors and then evaluate if it contains a “multiple” attribute using the get_attribute() method.

```
dropdown = driver.find_element_by_tag_name('select')
if dropdown.get_attribute("multiple"):
    print("multiple select options can be chosen")
else:
    print("only one select option can be selected")
```

– By Tushar Sir

Selecting multiple Dropdown options in Selenium WebDriver

- Once you know that multiple options can be selected in a dropdown, we can iterate over options and choose them using Selenium WebDriver's "select" class methods.
- Alternatively, we can also use the "actionchains" class of Selenium WebDriver to select multiple choices.
- The strategy is first to get the <select> web element and then perform chained click actions on it with "ctrl" key pressed.
- For our demo test page, we demonstrate both approaches to select multiple options in Selenium Python.
- For the "action chains" demo, we first evaluate if the <select> element with id "lang2" is having a "multiple" attribute.

– By Tushar Sir

Selecting multiple Dropdown options in Selenium WebDriver

- If yes, then we select webelements with “PHP” and “C#” options respectively –

```
myOption = driver.find_element_by_xpath("//select[@multiple]/option[contains(text(), 'C#')]")
myOption1 = driver.find_element_by_xpath("//select[@multiple]/option[contains(text(), 'PHP')]")
```

- and then we can execute chained actions on it using the following code –

```
ActionChains(driver).key_down(Keys.CONTROL).click(myOption).key_up(Keys.CONTROL).perform()
ActionChains(driver).key_down(Keys.CONTROL).click(myOption1).key_up(Keys.CONTROL).perform()
```

- For the Selenium SELECT class demo, we first find the <select> element with id “lang2” having a “multiple” attribute. If yes, then we select options with “Java,” “PHP,” and “Python” using select_by_index, select_by_value, select_by_visible_text, respectively.

```
dropdown = Select(driver.find_element_by_id('lang2'))
dropdown.select_by_index(3)
dropdown.select_by_value('php')
dropdown.select_by_visible_text('Python')
```

– By Tushar Sir

Selecting all the Dropdown options in Selenium WebDriver

- To choose all options, we can loop over available options and select using Selenium WebDrivers SELECT APIs. For our example demo, to select all options of webelement with id “lang2” –

```
dropdown = Select(driver.find_element_by_id('lang2'))
for opt in dropdown.options:
    dropdown.select_by_visible_text(opt.get_attribute("innerText"))
```

– By Tushar Sir

Deselecting or clearing already selected Dropdown options in Selenium WebDriver

To deselect an option we can use any of :

- ***deselect_by_index(self, index)***
- ***deselect_by_value(self, value)***
- ***deselect_by_visible_text(self, text)***
- And to deselect all, we have the method ***deselect_all(self)***

We can deselect python from multiple selected options using –

```
dropdown = Select(driver.find_element_by_id('lang2'))
dropdown.deselect_by_visible_text('Python')
```

– By Tushar Sir

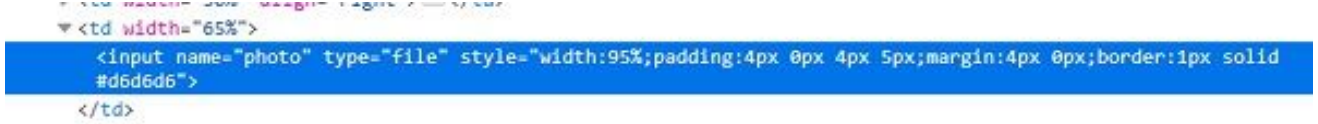
Handling Dropdowns

```
# importing the modules
from selenium import webdriver
from selenium.webdriver.support.ui import Select
import time

# using chrome driver
driver=webdriver.Chrome()

# web page url
driver.get("https://fs2.formsite.com/meherpavan/form2/index.html?1537702596407")

# find id of option
x = driver.find_element_by_id('RESULT_RadioButton-9')
drop=Select(x)

# select by visible text
drop.select_by_visible_text("Afternoon")
time.sleep(4)
driver.close()
```

– By Tushar Sir

Handling file uploads

- 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.
- This feature is only applied to elements having the type attribute set to file. Also, the tagname of the element should be input.
- Let us investigate the html code of an element having the above properties.



– By Tushar Sir

Handling file uploads

```
from selenium import webdriver
driver = webdriver.Chrome(executable_path="C:\chromedriver.exe")
driver.implicitly_wait(0.5)
driver.maximize_window()

driver.get("https://www.tutorialspoint.com/selenium/selenium_automation_practice.htm")
#to identify element
s = driver.find_element_by_xpath("//input[@type='file']")
#file path specified with send_keys
s.send_keys("C:\Users\Pictures\Logo.jpg")
```

– By Tushar Sir

Handling file uploads

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

driver = webdriver.Chrome(executable_path="C:\chromedriver.exe")
driver.implicitly_wait(0.5)
driver.maximize_window()

driver.implicitly_wait(10)
driver.get("https://the-internet.herokuapp.com/upload")
driver.find_element(By.ID,"file-upload").send_keys("selenium-snapshot.jpg")
driver.find_element(By.ID,"file-submit").submit()
if(driver.page_source.find("File Uploaded!")):
    print("file upload success")
else:
    print("file upload not successful")
driver.quit()
```

– By Tushar Sir

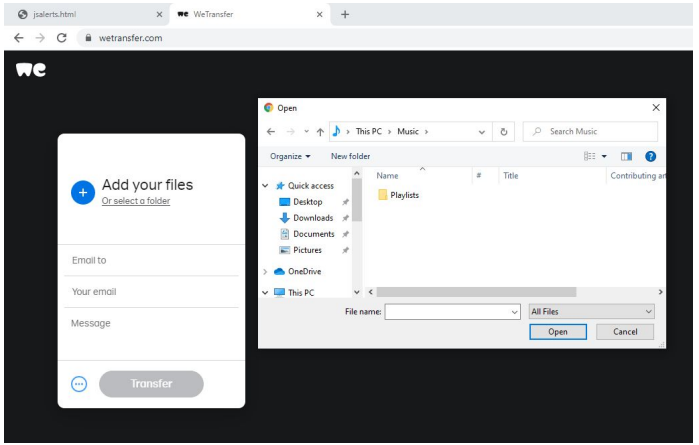
Handling file uploads

```
from selenium import webdriver
from webdriver_manager.chrome import ChromeDriverManager
from selenium.webdriver.common.by import By

driver = webdriver.Chrome()
driver.maximize_window()

driver.get('http://autopract.com/selenium/upload1/')

file_input = driver.find_element(By.XPATH,"//input[@name='files[]']")
file_path = "D:\iVagus\SelPy\Luffy.png"
file_input.send_keys(file_path)

driver.find_element(By.XPATH,"//button[@type='submit']").click()

print(driver.find_element(By.XPATH,"//p[@class='name']").text)
```

– By Tushar Sir

Handling Alerts, Popups and Multi-windows

- **Windows-based alerts** and **browser-based alerts** are two main categories of alerts or popups in JavaScript.
- Windows (or OS-based alerts) are alerts that are triggered by calling native operating system APIs. These types of alerts cannot be interacted directly using Selenium WebDriver, as Selenium is primarily used for automating browser interactions and not OS interfaces.
- An example of windows based alert pop-up is **'file uploading window'** on websites. For instance, when we visit wetransfer to upload files, we get the below screen.



– By Tushar Sir

Handling Alerts, Popups and Multi-windows

- As the name indicates, browser-specific alerts (or popups) are generally encountered when interacting with websites (or web products). They are broadly classified in three categories:
  - **Simple alerts**
  - **Prompt alerts**
  - **Confirm alerts**
  - **Authentication alerts**

– By Tushar Sir

Handling Alerts, Popups and Multi-windows

Simple JavaScript Alerts :

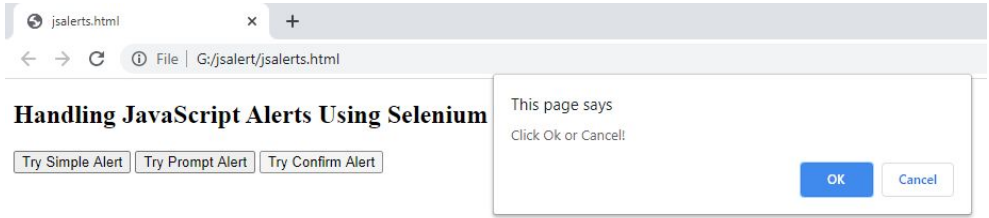
- Simple Alert is used to display some message (or error) in the browser pop-up. An example is informing the user about mandatory fields (e.g. “email is a mandatory field”).

– By Tushar Sir

Handling Alerts, Popups and Multi-windows

Confirm JavaScript Alerts :

- Confirm alert is used to take confirmation from the user regarding a specified action that will be performed by the user. For example, a confirmation alert saying “You have unsaved changes. Do you want to reload anyways?” comes up when you have unsaved work in Google docs, Canva, or other similar websites.



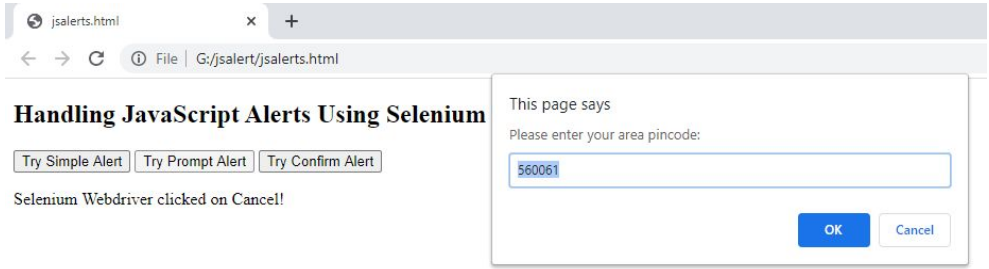
- When you perform a mouse operation on Okay or Cancel, the activity is recorded and the corresponding code block is executed.

– By Tushar Sir

Handling Alerts, Popups and Multi-windows

Prompt JavaScript Alerts :

- Prompt alert is used to send a message to the user as well as collect a message from the user.



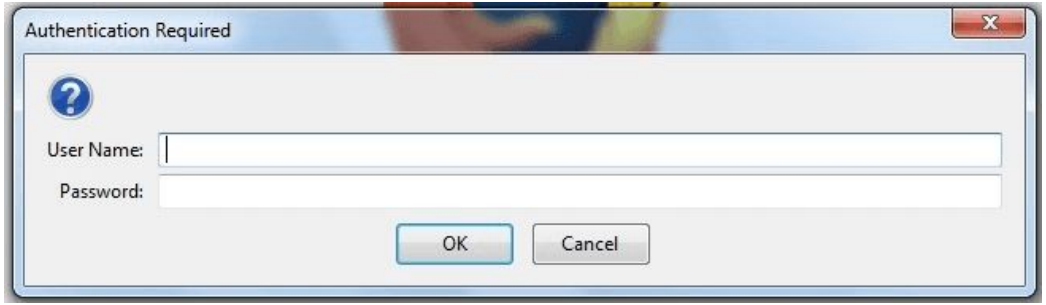
- We input pin code in the input window and an alert message appears when the Submit button is clicked.

– By Tushar Sir

Handling Alerts, Popups and Multi-windows

Authentication JavaScript Alerts :

- Authentication alert is an advanced form of a prompt alert. To exhibit an authentication alert, we need to set up a server and then trigger the pop-up. The user enters a URL to be fetched in the browser and the server responds with a 401 response.
- Additionally, it triggers an authentication pop-up. When the user enters the credentials and submits the pop-up, the same URL is requested yet again but this time with credentials in the request header. If authenticated, the website loads normally.



– By Tushar Sir

Handling Alerts, Popups and Multi-windows

- When an alert gets triggered, the control still remains with the webpage and an alert interface is required to shift the control to the pop-up window. The switch_to method in Python is used for switching to the desired alert window.

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

- The Java equivalent of the same is below:

```
Alert alert = driver.switchTo().alert();
```

– By Tushar Sir

Handling Alerts, Popups and Multi-windows

- Here are the major operations that can be performed on JavaScript alert in Selenium WebDriver:
- **accept() :** This is akin to pressing the “OK” button in the alert pop-up window.
- **sendKeys() :** This is for prompt alert which requires input from the user. Using this method you can send keys/text as input to the alert box.
- **dismiss() :** Cancels the alert, closes the window, and control is handed over to the webpage.
- **text() :** Enter text in the alert box using the sendKeys() method in Selenium.

– By Tushar Sir

Handling Alerts, Popups and Multi-windows

```
from selenium import webdriver
import time
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.common.exceptions import TimeoutException

driver = webdriver.Chrome()
driver.get("file:///G:/jsalert/jsalerts.html")

try:
    driver.find_element_by_id('simple').click()
    WebDriverWait(driver, 5).until(EC.alert_is_present(), 'Timed out waiting for simple alert to appear')
    alert = driver.switch_to.alert
    alert.accept()
    print("alert accepted")
except TimeoutException:
    print("no alert")
time.sleep(2)
print("simple alert test passed")
print("Now running prompt alert test")

try:
    driver.find_element_by_id('prompt').click()
    WebDriverWait(driver, 5).until(EC.alert_is_present(), 'Timed out waiting for prompt alert to appear')
    alert = driver.switch_to.alert
    alert.send_keys('560085')
    print("printing alert text")
    print(alert.text)
    alert.accept()
    print("alert accepted")
```

By Tushar Sir

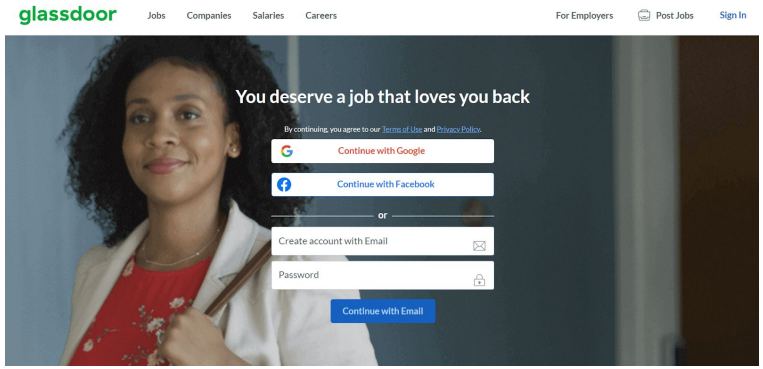
Handling Alerts, Popups and Multi-windows

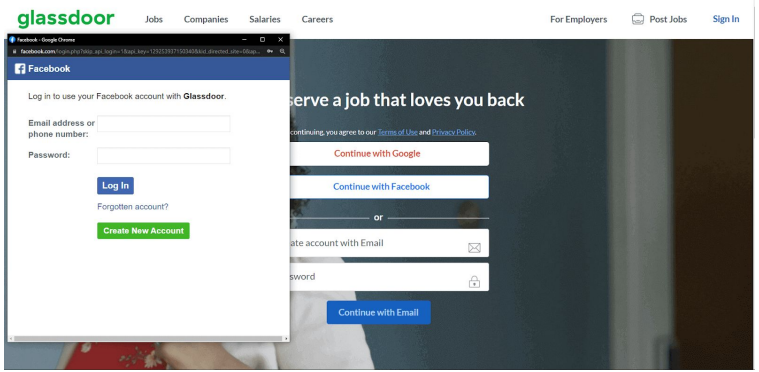
```
except TimeoutException:
    print("no alert")
time.sleep(2)
print("prompt alert test passed")
print("Now running confirm alert test")

try:
    driver.find_element_by_id('confirm').click()
    WebDriverWait(driver, 5).until(EC.alert_is_present(), 'Timed out waiting for prompt alert to appear')
    alert = driver.switch_to.alert
    print("printing alert text from confirmation alert")
    print(alert.text)
    alert.accept()
    print("alert accepted")
except TimeoutException:
    print("no alert")
time.sleep(2)
print("confirmation alert test passed")

driver.quit()
```

– By Tushar Sir

Handling Alerts, Popups and Multi-windows

- The automated test cases you perform using Selenium might lead you to instances where it is necessary to handle multiple windows while working with a web application.
- This situation may arise as a result of clicking a button on the website that opens up a URL to a new tab or a new window.
- For example, the job search platform Glassdoor opens a new window when you click the buttons to sign in using your Google account or Facebook account.



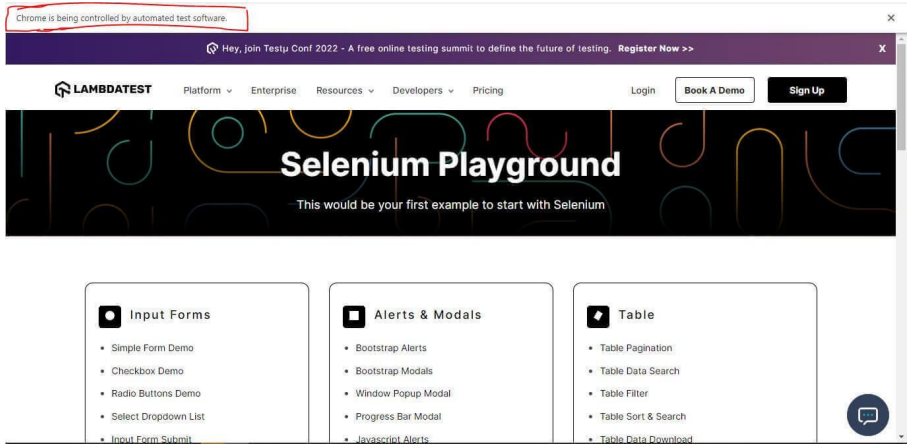


– By Tushar Sir

Handling Alerts, Popups and Multi-windows

What is a Browser Window in Selenium?

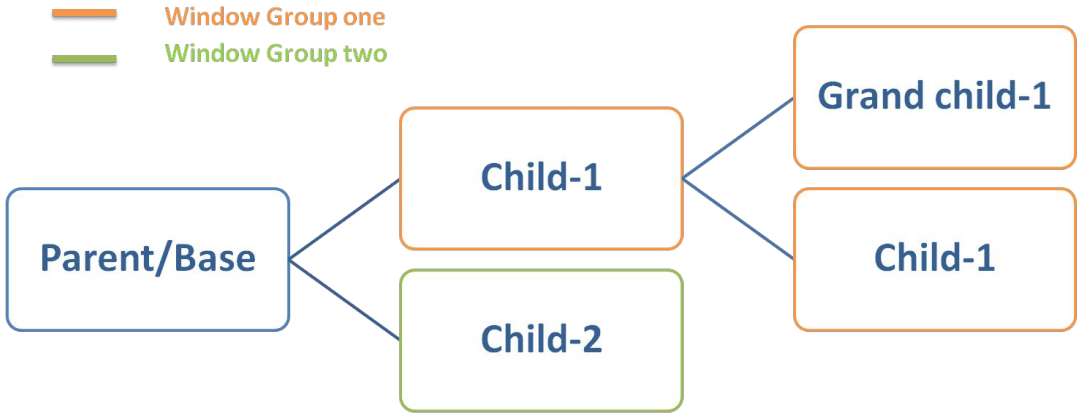
- The browser window, often called the main or parent window, represents the homepage or the currently open web page a user sees when opening a browser.
- When a Selenium automation script runs, it typically starts with the parent window. Initiating a Selenium WebDriver session involves opening a window that is initially controlled by the WebDriver.



– By Tushar Sir

Handling Alerts, Popups and Multi-windows

- Selenium WebDriver always keeps the context of the parent window when navigating to a defined URL that opens a new window.
- The child window will be within the context of the parent window.
- This enables navigating back and forth between the windows once you are done with the operations you were handling in the newly opened child window.

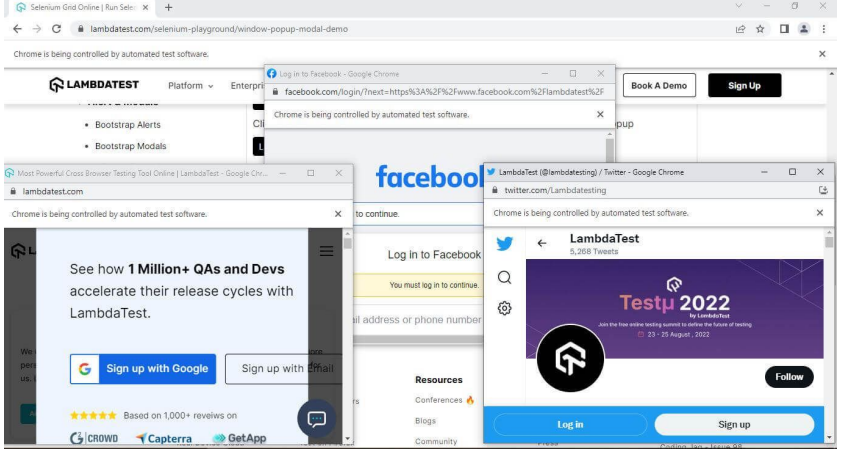


– By Tushar Sir

Handling Alerts, Popups and Multi-windows

What is a Child Window in Selenium?

- When we click on a button or URL link in the parent window, and the action opens another window(s) within the main window, the new window(s) is called a child window. The new window(s) or tab(s) opened are called children of the parent window and have unique window handles.



– By Tushar Sir

Handling Alerts, Popups and Multi-windows

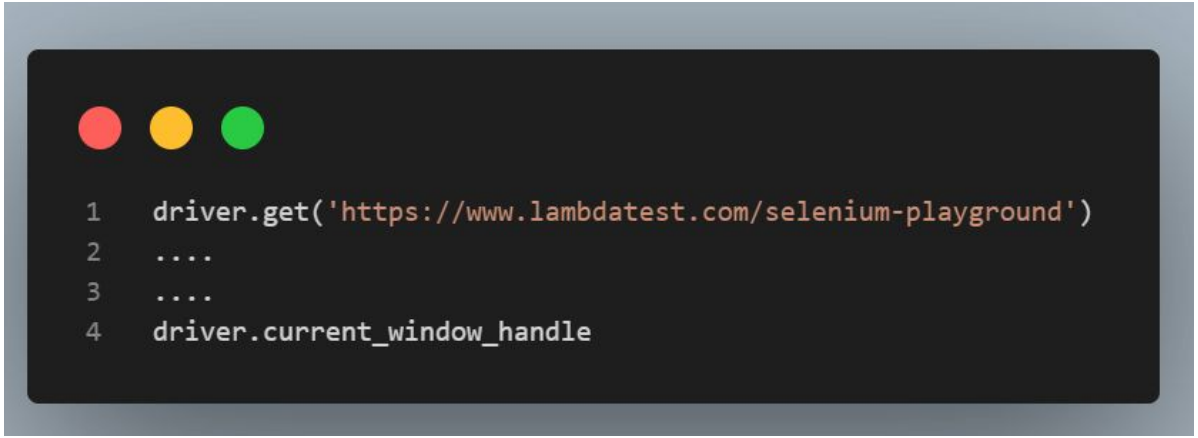
- We can handle the occurrence of child windows or tabs when running test cases using Selenium Python. When a parent window opens a child window, the WebDriver only controls one window at a time in the running session.
- The WebDriver controls the main window, and any actions in a test script will be executed in the currently active window.
- The Selenium Python WebDriver provides the following methods to handle multiple windows.

– By Tushar Sir

Handling Alerts, Popups and Multi-windows

current_window_handle() :

- This method collects the unique window handle ID of the browser window that is currently active.
- **Syntax:** driver.current_window_handle



– By Tushar Sir

Handling Alerts, Popups and Multi-windows

window_handles() :

- This method returns the IDs of all the windows that have been opened by the WebDriver in the current session. It stores the window IDs to be used for switching windows.
- **Syntax:** driver.window_handles

– By Tushar Sir

Handling Alerts, Popups and Multi-windows

switch_to.window() :

- This method switches the WebDriver's focus from the currently open browser window to the intended browser window.
- The targeted window's id is passed as the argument to shift the WebDriver's control to the new window.
- **Syntax:**
- **Option 1:** driver.switch_to.window()
- **Option 2:** driver.switch_to.window()

– By Tushar Sir

Handling Alerts, Popups and Multi-windows

```python
# import selenium module
from selenium import webdriver

# import select class
from selenium.webdriver.support.ui import Select

# using chrome driver
driver = webdriver.Chrome()

# web page url and open first window page
driver.get("http://demo.automationtesting.in/Windows.html")

# find xpath of button for child window page
# this page no. 2
driver.find_element_by_xpath('//*[@id="Tabbed"]/a/button').click()

# return all handles value of open browser window
handles = driver.window_handles
for i in handles:
    driver.switch_to.window(i)

    # print every open window page title
    print(driver.title)
```

– By Tushar Sir

Handling Mouse events

- The mouse and keyboard movements can be simulated with the help of Selenium WebDriver.
- The actions like a double click, right-click, mouse movement, key press, mouse button connection, hovering and drag and drop and so on are performed with the help of ActionChains class in Selenium.
- While we are using methods of ActionChains object, all the actions are kept in a queue.
- After executing the perform () method, all the actions are triggered one by one in a sequence. The actions can be queued up one after the other or placed in a chained order.

– By Tushar Sir

Handling Mouse events

Let us see some of the mouse actions methods under ActionChains:

- **click (args) –** This method shall click on a web element. The argument args is the web element to be clicked. If the parameter to the method is ignored, then the current mouse position is clicked.
- **click_and_hold (args) –** This method shall click on the current mouse location without releasing it. The argument args is the web element to mouse down. If the parameter to the method is ignored, then the current mouse position is clicked.
- **double_click (args) –** This method shall double click on a web element. The argument args is the web element to be double-clicked. If the parameter to the method is ignored, then the current mouse position is double-clicked.

– By Tushar Sir

Handling Mouse events

- **context_click (args) –** This method shall right click on a web element. The argument args is the web element to be right-clicked. If the parameter to the method is ignored, then the current mouse position is right-clicked.
- **drag_and_drop (s, d) –** This method is used to hold down the mouse from the element at the source, then move to the destination element and finally release the mouse. The argument s refers to the location of the source element to mouse down and the argument d refers to the location of the destination to mouse up.
- **drag_and_drop_by_offset (s, offsetX, offsetY) –** This method is used to hold down the mouse from the element at the source, then move to the destination element offset, and finally release the mouse. The argument s refers to the location of the source element to mouse down. The argument offsetX refers to the X offset to move to. The argument offsetY refers to the Y offset to move to.
- **move_to_element (args) –** This method is used to move to the middle of the web element. The args is the web element to be moved to.

– By Tushar Sir

Handling Mouse events

- **move_to_element (args) –** This method is used to move to the middle of the web element. The args is the web element to be moved to.
- **release (args) –** This method is used to release a held mouse button on a web element. The argument args is the web element to mouse up. If the parameter to the method is ignored, then the current mouse position is released.

– By Tushar Sir

Handling Mouse events

Let us see some of the keyboard methods under ActionChains:

- **send_keys (keys) –** This method is used to send keys to the present active element. The argument keys are keys to send. The modifier keys are available from the Keys class.
- **key_down (v, args) –** This method is used to do a key press without releasing it. This method can be used only with the Control, Alt, and Shift modifier keys. The argument v is a modifier key available from the Keys class. The argument args is the web element to send keys. If that parameter to the method is ignored, it sends keys to the present active element.
- **key_up (v, args) –** This method is used to release a modifier key. The argument v is a modifier key available from the Keys class. The argument args is the web element to send keys. If that parameter to the method is ignored, it sends keys to the present active element.

– By Tushar Sir

Handling Mouse events

```
# Code Implementation for mouse hover action.
from selenium import webdriver
from selenium.webdriver import ActionChains
from selenium.webdriver.common import keys
driver = webdriver.Chrome (executable_path="C:\\chromedriver.exe")

driver.get("https://www.softwaretestingmaterial.com/")

l = driver.find_element_by_link_name ("Blog")
act = ActionChains(driver)
act.move_to_element (l)
act.perform ()
driver.quit ()
```

```
# Code Implementation for clicking action.

from selenium import webdriver
from selenium.webdriver import ActionChains
from selenium.webdriver.common import keys
driver = webdriver.Chrome (executable_path="C:\\chromedriver.exe")

driver.get ("https://www.softwaretestingmaterial.com")

s = driver.find_element_by_link_text ("Blog")

act = ActionChains(driver)
act.click (s)
act.perform ()

driver.quit ()
```

```
# Code Implementation for drag and drop action.

from selenium import webdriver
from selenium.webdriver import ActionChains
from selenium.webdriver.common import keys
driver = webdriver.Chrome (executable_path="C:\\chromedriver.exe")

driver.get ("http://jqueryui.com/droppable/")
driver.switch_to.frame (0)

s = driver.find_element_by_css_selector ("#draggable")
t = driver.find_element_by_css_selector ("#droppable")

act = ActionChains(driver)
act.drag_and_drop (s, t)

act.perform ()
driver.quit ()
```

– By Tushar Sir

Handling Mouse events

```
# Code Implementation for key up and down action.

from selenium import webdriver
from selenium.webdriver import ActionChains
from selenium.webdriver.common.keys import Keys
driver = webdriver.Chrome (executable_path="C:\\chromedriver.exe")

driver.get ("https://www.softwaretestingmaterial.com")

s = driver.find_element_by_css_selector ("#form-field-name")
# Action Chains object creation
act = ActionChains(driver)
act.click (s)
# to press SHIFT key
act.key_down (Keys.SHIFT)
# to send keys on the element
act.send_keys ("Python")
# to release SHIFT key
act.key_up (Keys.SHIFT)
# perform the queued actions
act.perform ()
driver.quit ()
```

– By Tushar Sir

Screenshot handling

- 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.
- **Syntax :** driver.save_screenshot("image.png")

```
# importing webdriver from selenium
from selenium import webdriver
from PIL import Image
# Here Chrome will be used
driver = webdriver.Chrome()

# URL of website
url = "https://www.geeksforgeeks.org/"

# Opening the website
driver.get(url)

driver.save_screenshot("image.png")

# Loading the image
image = Image.open("image.png")

# Showing the image
image.show()
```

– By Tushar Sir

Thank You

- **By Tushar Sir**