**Demonstrate how elements are located using Selenium WebDriver.**

**Different types of Locators in Selenium WebDriver**

Below is the list of these locators of Selenium WebDriver:

ID

Name

ClassName

LinkText

Partial LinkText

TagName

CssSelector

XPath

As mentioned earlier, [Selenium WebDriver](https://www.lambdatest.com/blog/selenium-webdriver-tutorial-with-examples/) provides different web locators for locating WebElements on the page. Here are the different locators in Selenium WebDriver that I will be covering in-depth in the latter part of the blog:

|  |  |  |
| --- | --- | --- |
| Locator | Description | Syntax (in Java) |
| ID | Identify the WebElement using the ID attribute | driver.findElement(By.id(“IdValue”)); |
| Name | Identify the WebElement using the Name attribute | driver.findElement(By.name(“nameValue”)); |
| ClassName | Use the Class attribute for identifying the object | driver.findElement(By.className(“classValue”)); |
| LinkText | Use the text in hyperlinks to locate the WebElement | driver.findElement(By.linkText(“textofLink”)); |
| Partial LinkText | Use a part of the text in hyperlinks to locate the desired WebElement | driver.findElement(By.partialLinkText(“PartialTextofLink”)); |
| CssSelector | CSS used to create style rules in web page is leveraged to locate the desired WebElement | driver.findElement(By.cssSelector(“cssValue”)); |
| XPath | Use XPath to locate the WebElement | driver.findElement(By.xpath(“xpathValue”)); |

Using locators in Selenium 4 is a treat due to the introduction of [relative locators in Selenium 4](https://www.lambdatest.com/blog/selenium-4-relative-locator/). The introduction of locators like above(), below(), toLeftOf(), toRightOf(), and near() makes it easy to locate WebElements in relation to a particular WebElement.

This Selenium [WebDriver Tutorial](https://www.lambdatest.com/learning-hub/webdriver) for beginners and professionals will help you learn what’s new in Selenium 4 (Features and Improvements).

How to identify elements using

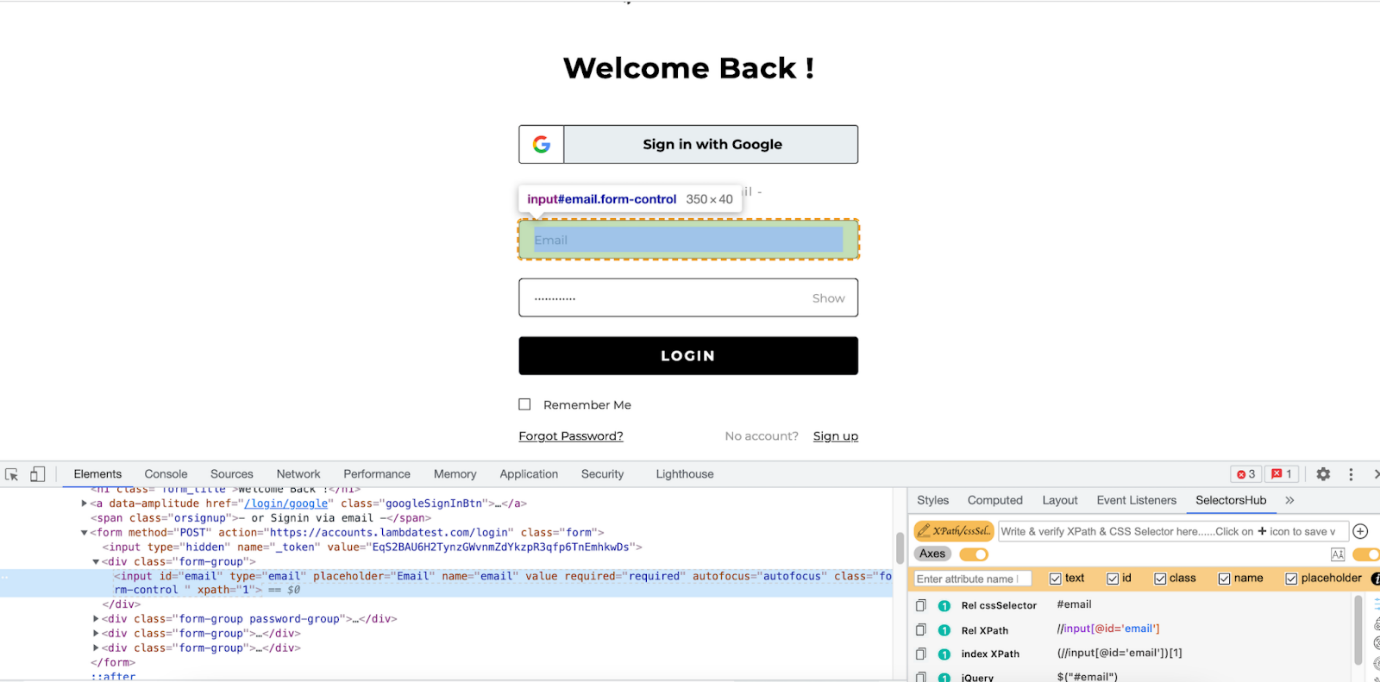
[ID locator in Selenium](https://www.lambdatest.com/blog/making-the-move-with-id-locator-in-selenium-webdriver/) is the most preferred and fastest way to locate desired WebElements on the page. ID Selenium locators are unique for each element in the DOM.

Since IDs are unique for each element on the page, it is considered the fastest and safest method to locate elements. But unfortunately, developers may or may not follow this rule as browsers do allow bypassing this rule.

Specifically, in the case of a table or list, the IDs may populate incrementally or dynamically depending upon the data in the table. In such cases, testers use other locators in Selenium WebDriver to locate the desired element on the page.

One of the [Selenium best practices](https://www.lambdatest.com/blog/selenium-best-practices-for-web-testing/) is to leverage the capabilities offered by the ID locator in Selenium since it is the fastest locator of the entire lot. Therefore, choosing the ID locator over other locators in Selenium WebDriverwill go a long way to [speed up Selenium test case](https://www.lambdatest.com/blog/speed-up-selenium-test-cases-execution/) execution.

Below is an example of the LambdaTest Login page showcasing how the ‘login’ field is being located via ID:



I have used the [SelectorsHub tool](https://www.lambdatest.com/blog/selectorshub-the-next-gen-xpath-css-selectors-tool/) to locate the desired WebElement using the ID locator. Below is the DOM structure of the element:

|  |  |
| --- | --- |
| 1  2 | <input id="email" type="email" placeholder="Email" name="email" value="" required="required"  autofocus="autofocus" class="form-control " xpath="1"> |

The below method is used for locating the desired element using the ID locator:

|  |  |
| --- | --- |
| 1 | driver.findElement(By.id("email")) |

If no element in the DOM matches with the required ID, NoSuchElementException is thrown. Therefore, it is important to have a good know-how of the common [exceptions in Selenium](https://www.lambdatest.com/blog/49-common-selenium-exceptions-automation-testing/) to build a

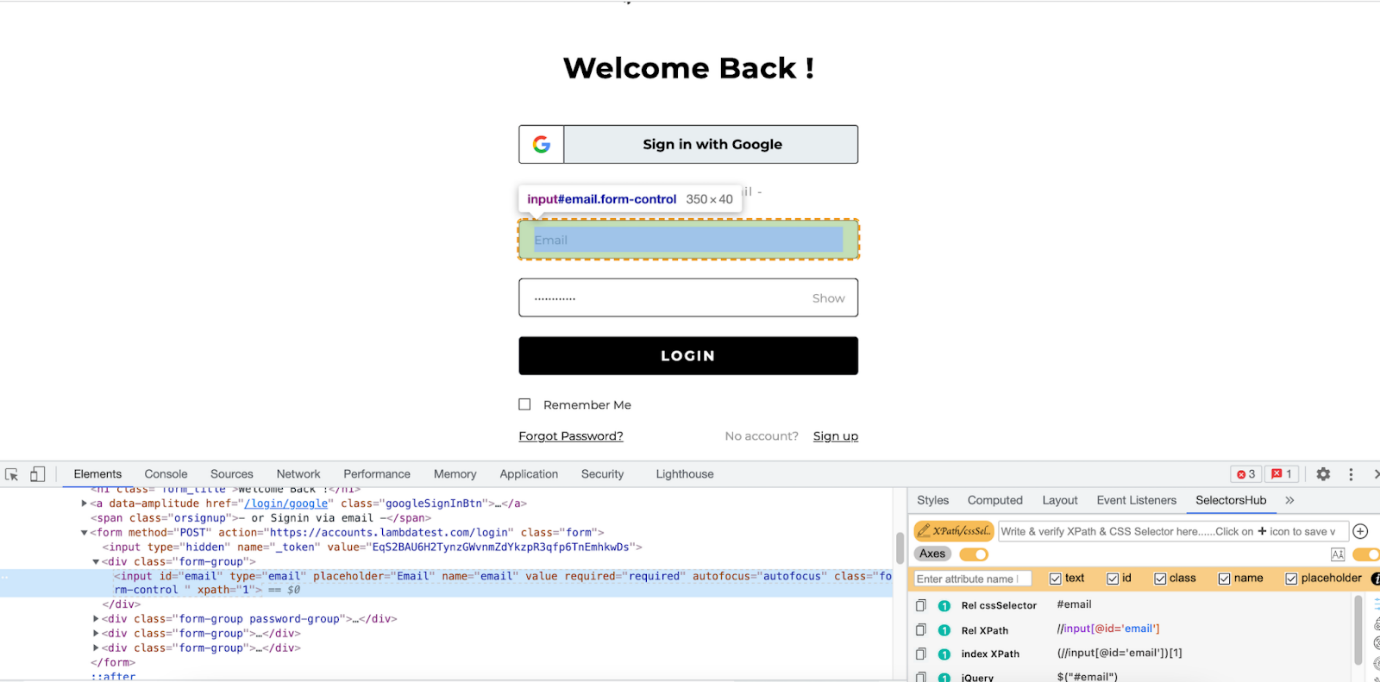
This certification is for anyone who wants to stay ahead among professionals who are growing their career in Selenium automation testing.

An element can be defined via multiple attributes, one such attribute is Name. Name locator in Selenium WebDriver can also be used to locate elements like an ID locator.

Unlike ID locators, which are unique for a page, the Name locator may or may not have a unique value. If there are WebElements with the same name, the locator selects the first element with that Name on the page.

In case no such name matches with the defined attribute value, NoSuchElementException is raised.

To demonstrate the usage of the Name locator in Selenium WebDriver, we identify the same WebELement that was earlier located using the ID locator.



|  |  |
| --- | --- |
| 1 | <input id="email" type="email" placeholder="Email" name="email" value="" required="required" autofocus="autofocus" class="form-control " xpath="1"> |

Here is how the desired WebElement was located using the Name locator in Selenium:

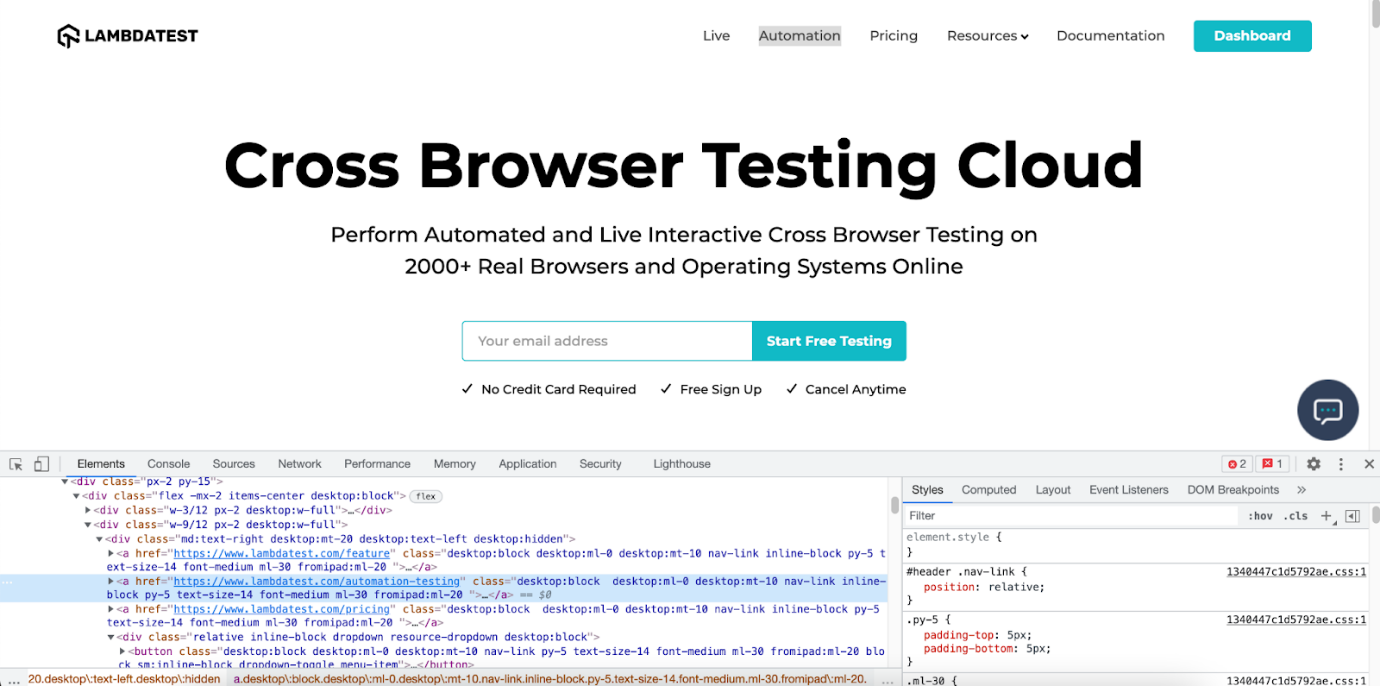
|  |  |
| --- | --- |
| 1 | driver.findElement(By.name("email")); |

‘Link Text’ Locator In Selenium

Elements can be located via link text that is present in the hyperlinks. For example, the first link would be selected in a scenario where there are multiple links of the same text.

However, this Identifier strategy can only be used for elements that have an anchor(a) tag.

Below is an example of Lambdatest homepage showcasing the selection of the ‘Automation’ link that is available on the header. The DOM below shows the highlighted element:



Below is the DOM structure of the same:

|  |  |
| --- | --- |
| 1 | <a href="https://www.lambdatest.com/automation-testing" class="desktop:block  desktop:ml-0 desktop:mt-10 nav-link inline-block py-5 text-size-14 font-medium ml-30 fromipad:ml-20 ">Automation</a> |

Here is how the desired WebElement was located using the linkText locator in Selenium:

|  |  |
| --- | --- |
| 1 | driver.findElement(By.linkText("Automation")); |

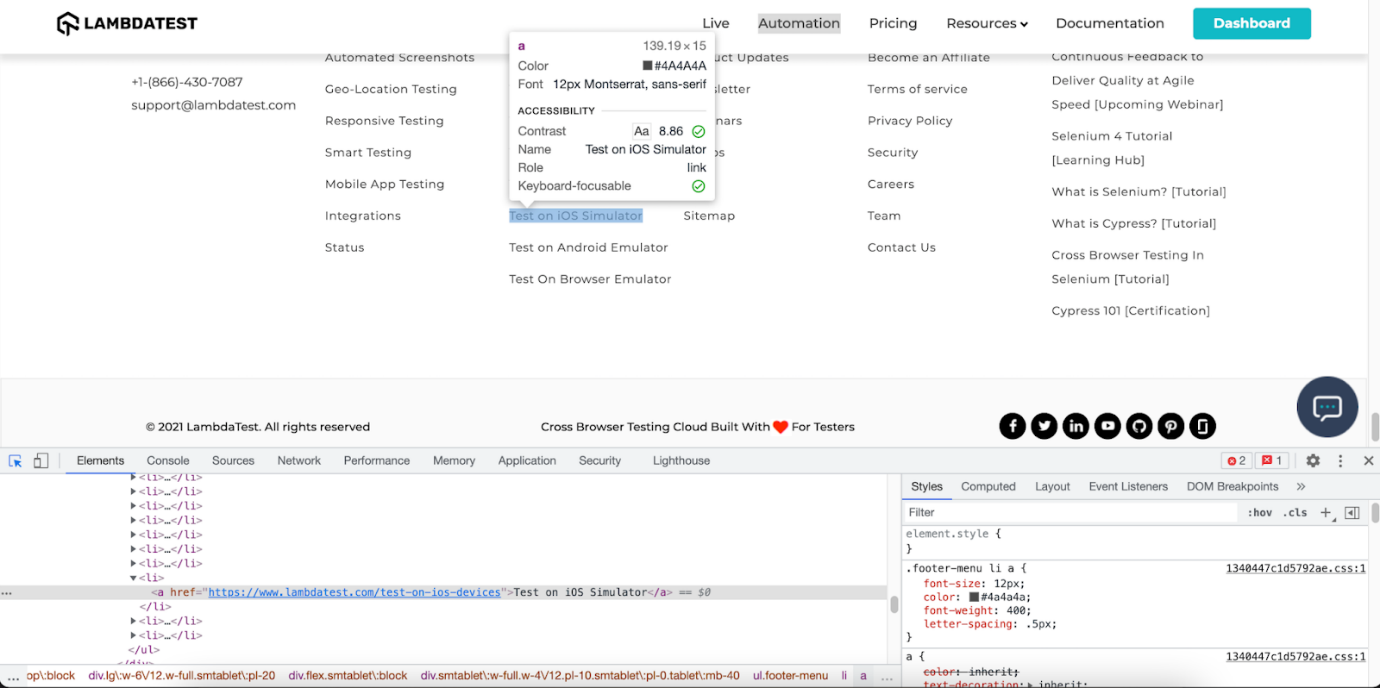
Partial Link Text Locator In Selenium

There is a provision to locate a WebELement using Partial Link Text akin to the normal Link Text locator in Selenium. Locating WebElements using partial link text is preferred when the link text is too long.

Here, the partial text helps identify a unique element and use it to perform further actions on it. Sometimes, using this can also be to locate multiple links on a page with a common partial text.

Read – [How to find HTML elements in Cypress](https://www.lambdatest.com/blog/finding-html-elements-using-cypress-locators/)

Below is a snapshot of the LambdaTest DOM highlighting the element with the link name as ‘Start testing.’ Instead of using the complete link text, I use the partial link text locator to locate the element using the ‘testing’ link text.



Here is the DOM structure of the element:

|  |  |
| --- | --- |
| 1 | <a href="https://www.lambdatest.com/test-on-ios-devices">Test on iOS Simulator</a> |

Here is how the desired WebElement was located using the partial link text locator in Selenium:

The syntax for locating element by partial link text is:

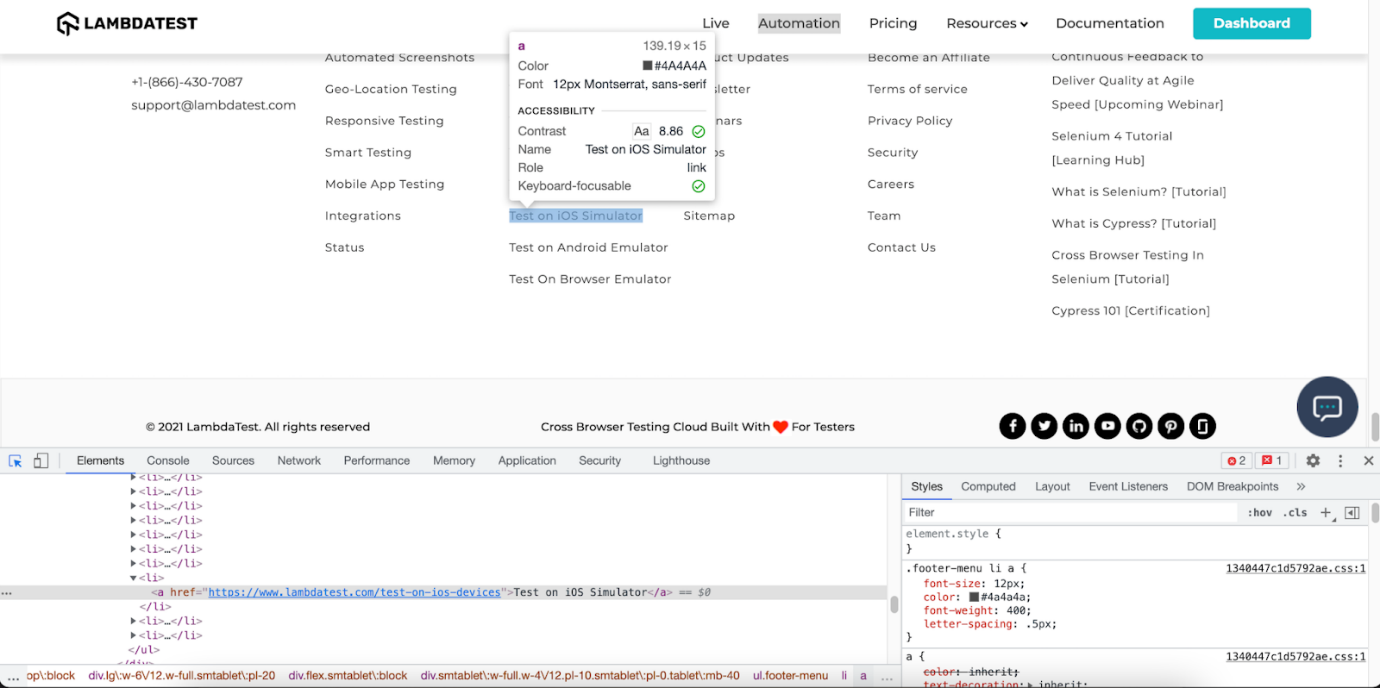
|  |  |
| --- | --- |
| 1 | driver.findElement(By.partialLinkText ("Simulator")); |

TagName Link Text Locator In Selenium

As the name specifies, this [CSS locator in Selenium](https://www.lambdatest.com/blog/how-pro-testers-use-css-selectors-in-selenium-automation-scripts/) WebDriver is used to identify elements using Tag names like div, table, h1, etc.

The TagName locator is commonly used to identify all the links on a page and identify [broken links in Selenium](https://www.lambdatest.com/blog/broken-links-testing-using-selenium/).

|  |  |
| --- | --- |
| 1 | driver.findElement(By.name("email")); |



Here is the syntax of locating all the links on the LambdaTest homepage:

|  |  |
| --- | --- |
| 1 | driver.findElements(By.tagName("a")); |

Class Name locator is used for locating WebElements that are defined using the class attribute. Shown below is the DOM snapshot of the LambdaTest login page.

For locating the ‘login’ element via the ClassName locator in Selenium, we use the class attribute in the following DOM structure:

|  |  |
| --- | --- |
| 1 | <button id="login-button" data-amplitude="" type="submit" class="btn btn-dark submit-btn" css="1">Login</button> |

Here is how the desired WebElement was located using the ClassName locator in Selenium:

|  |  |
| --- | --- |
| 1 | driver.findElement(By.className("submit-btn")); |

XPath Locator In Selenium

[XPath locator in Selenium](https://www.lambdatest.com/blog/complete-guide-for-using-xpath-in-selenium-with-examples/) helps in locating elements on the web page using XML expressions. The basic syntax used for using XPath as a CSS locator in Selenium WebDriver is shown below:

|  |  |
| --- | --- |
| 1 | XPath: //tagname[@attribute = 'value'] |

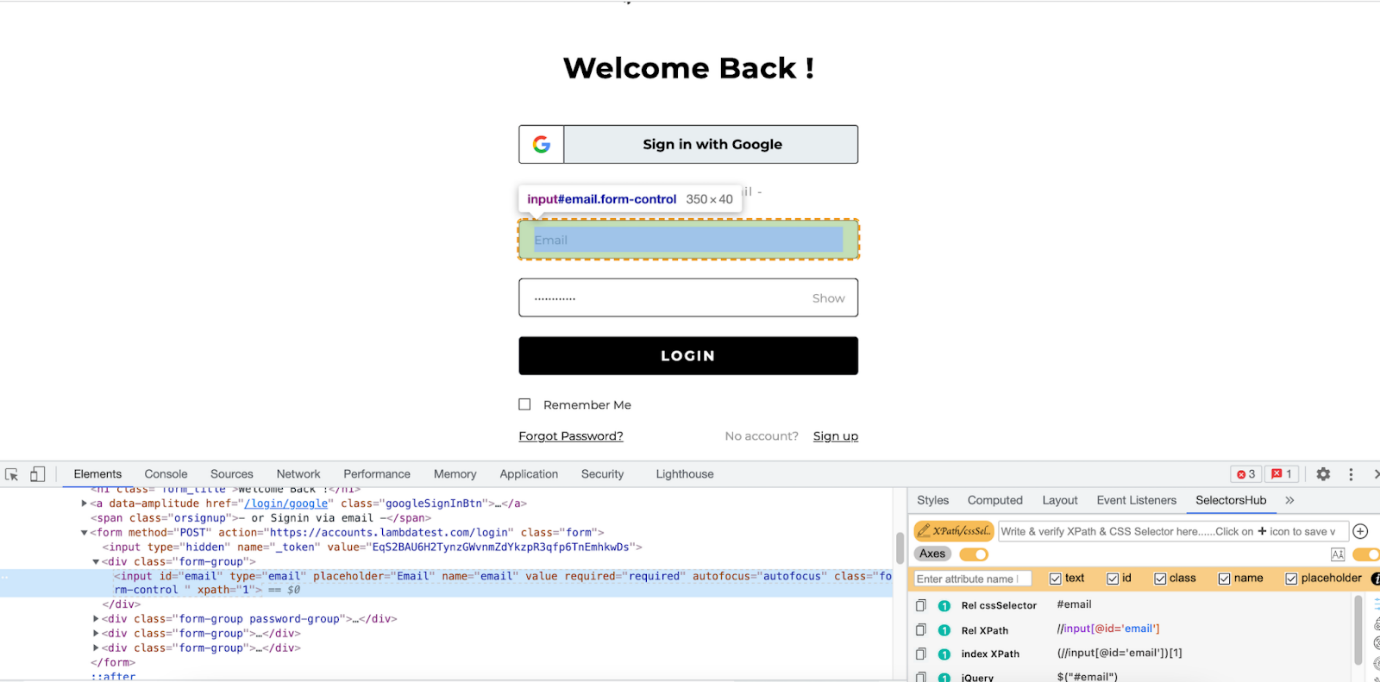
Here, [TagName in Selenium](https://www.lambdatest.com/blog/locating-elements-by-tagname-in-selenium/) signifies the tag in the DOM structure that you are targeting to locate the desired WebElement. TagName can be input tag, anchor tag, etc.

Attributes are defined via the prefix ‘@’ and their corresponding value. Thus, attributes like Name, ID, Class, etc., can be used along with TagName.

XPath in Selenium can be used in multiple ways, as shown below:

Standard XPath

As the name indicates, this is the most basic (or standard) way of writing an XPath. To demonstrate the usage of a standard XPath locator in Selenium, let’s locate the email element on the LambdaTest homepage.



Below is the DOM structure of the element:

|  |  |
| --- | --- |
| 1 | <input type="email" name="email" value="" placeholder="Email" required="required" autofocus="autofocus" class="form-control mt-3 form-control-lg"> |

The standard XPath of the desired WebElement is //input[@name= ’email’]. Here is how the XPath is used with the findElement() method to locate the element.

|  |  |
| --- | --- |
| 1 | driver.findElement(By.xpath("//input[@name= ’email’]")); |

XPath Contains

XPath similarly contains works like CSS selector ‘contains.’ It is extensively used on WebElements whose value is changing dynamically.

Consider an example where the value of the login changes after appending the login text. Here, XPath contains will be super-helpful in locating the desired WebElement.

Syntax:

|  |  |
| --- | --- |
| 1 | //tagname[contains(@attribute, ‘partial value of attribute’)] |

Below is the DOM structure of the element:

|  |  |
| --- | --- |
| 1 | <input type="text" placeholder="Full Name\*" name="name" value="" class="form-control sign-up-input-2 "> |

Here is how the desired WebElement was located using the ‘XPath contains’ locator in Selenium:

|  |  |
| --- | --- |
| 1 | driver.findElement(By.xpath("//input[contains(@class, ‘form-control’)]")) |

XPath using ‘AND’ & ‘OR’

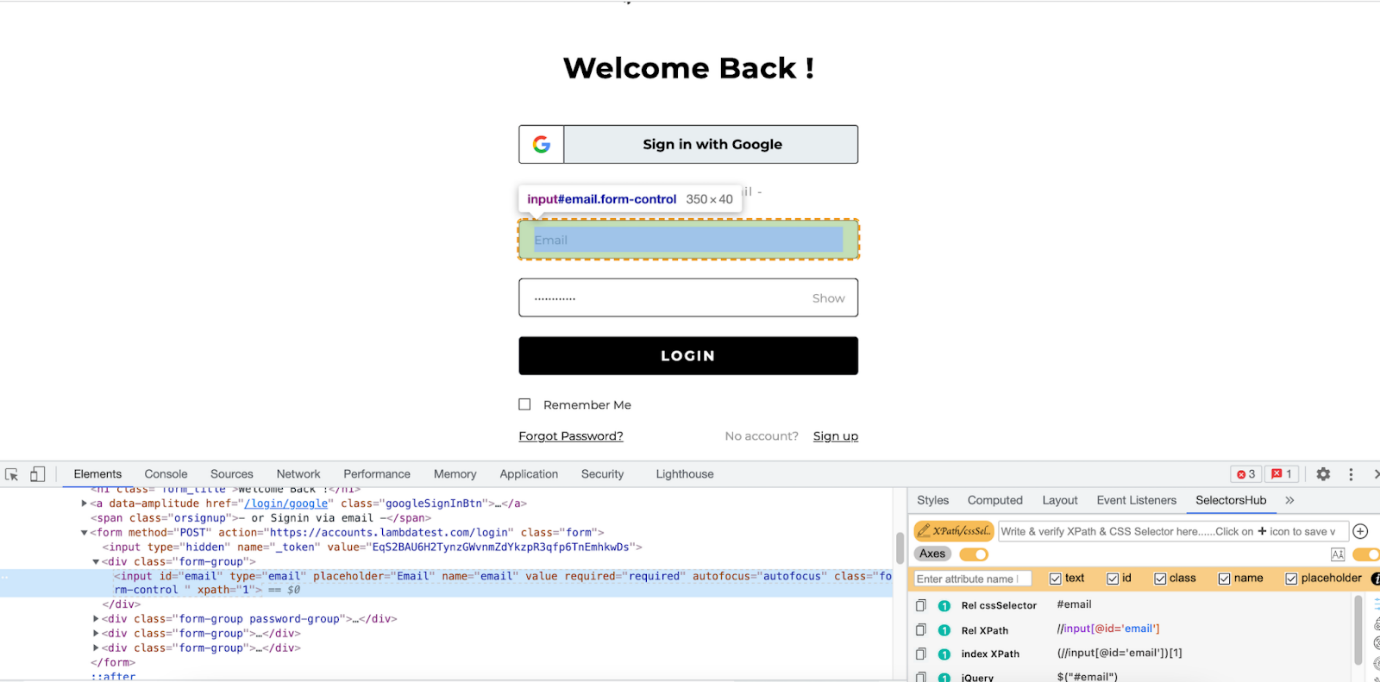
The ‘AND’ & ‘OR’ operators in the XPath selector in Selenium are used when locating a WebElement based on certain condition sets. In the case of ‘AND,’ both the conditions should be True. On the other hand, either of the two conditions can be true for ‘OR’ in operator XPath.

Syntax of OR operator in XPath:

|  |  |
| --- | --- |
| 1 | //input[@id='login\_1' OR @name='login’] |

Syntax of AND operator in XPath:

Let’s locate the email login element on the LambdaTest homepage using the ‘AND’ & ‘OR’ operators.



Below is the DOM structure of the element:

|  |  |
| --- | --- |
| 1 | <input type="email" name="email" value="" placeholder="Email" required="required" autofocus="autofocus" class="form-control mt-3 form-control-lg"> |

Here is how we used the OR operator with XPath locator in Selenium:

|  |  |
| --- | --- |
| 1 | driver.findElement(By.xpath("//input[@type='email' OR @name='email']")); |

Here is how we used the AND operator with XPath locator in Selenium:

|  |  |
| --- | --- |
| 1 | driver.findElement(By.xpath("//input[@type='email' AND @name='email']" |

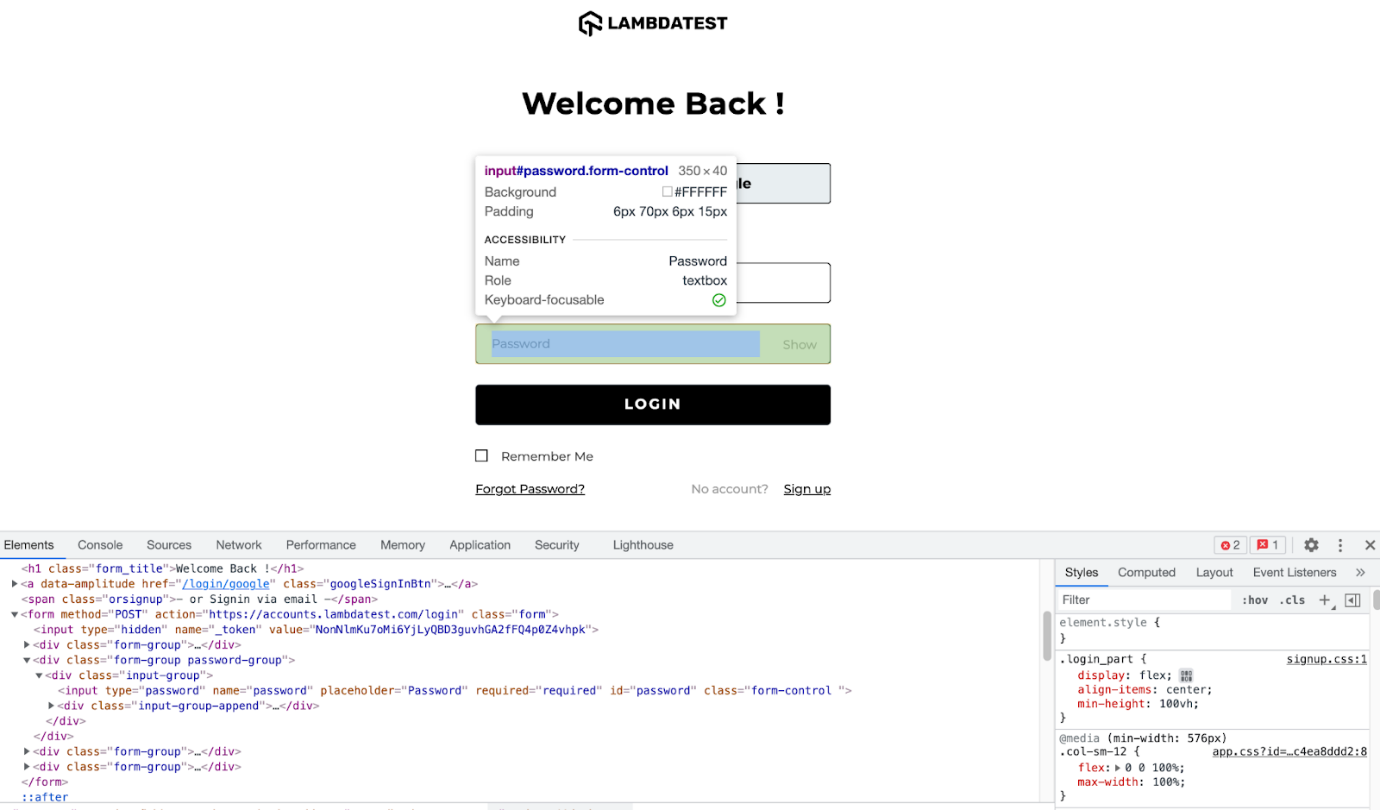
starts-with() method in XPath

The starts-with() method in XPath offers functionalities that are similar to the CSS Selector in Selenium. It helps in locating elements that start with a specified attribute value. The starts-with() method in XPath is majorly used for locating WebElements whose value changes on the refresh of a page.

Syntax:

|  |  |
| --- | --- |
| 1 | //tagname[starts-with(@attribute,'starting name of the attribute value')] |

Shown below is the DOM structure for locating the Password field on the LambdaTest signup page:



Below is the DOM structure of the element:

|  |  |
| --- | --- |
| 1 | <input type="password" placeholder="Desired Password\*" name="password" class="form-control sign-up-input-2 " aria-autocomplete="list"> |

Here is how we locate the Password element using the starts-with() method with XPath in Selenium:

|  |  |
| --- | --- |
| 1 | driver.findElement(By.xpath("//input[starts-with(@name,'pass')]")); |

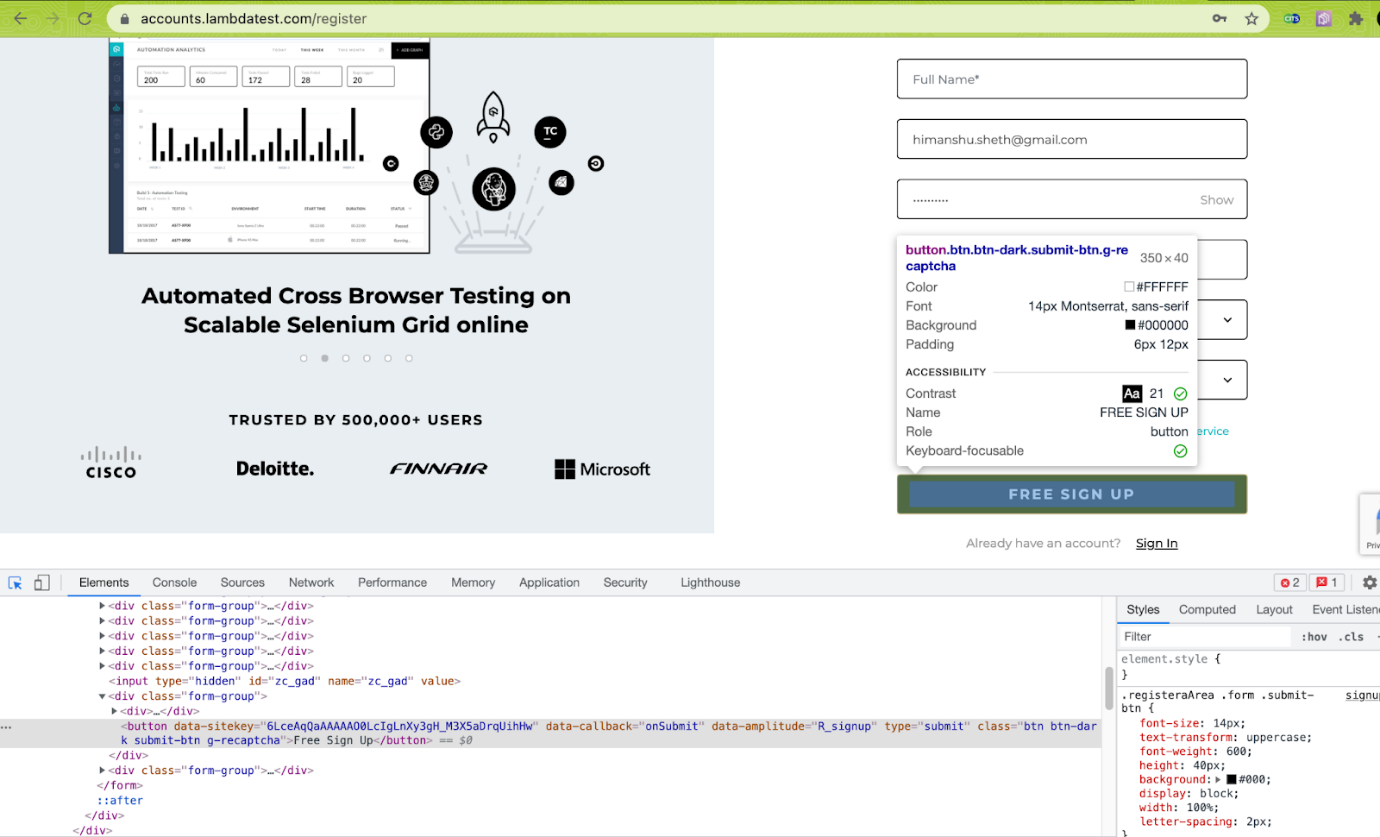
XPath Text

Text in the XPath locator in Selenium helps in locating WebElements via XPath using exact text match. It can be used when elements have to be located by looking into the tags containing certain text.

Syntax:

|  |  |
| --- | --- |
| 1 | //div[text()='Logged In'] |

To demonstrate XPath text usage, we locate the ‘FREE SIGN UP’ button on the [LambdaTest registration page](https://accounts.lambdatest.com/register" \t "_blank).



Here is the DOM structure of the required WebElement:

|  |  |
| --- | --- |
| 1 | <button class="btn btn-dark submit-btn g-recaptcha">Free Sign Up</button> |

Here is how we locate the ‘FREE SIGN UP’ button element using the XPath text:

|  |  |
| --- | --- |
| 1 | driver.findElement(By.xpath("//button[text()='Free Sign Up']")); |

Both CSS Selector and XPath are useful when running complex Selenium test automation scenarios. Though I use XPath extensively, the choice between XPath and CSS Selector purely depends on the scenario complexity and your convenience in locating WebElements using the corresponding locator.

When choosing, always look into the maintainability aspects of the locators, as that can make your