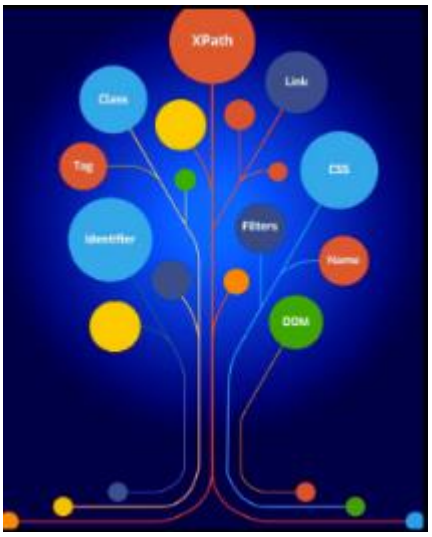
# All about Selenium Locators




**By ID**

Locate elements using the 'id' attribute of the HTML element. This is usually unique and preferred.

**By Name**

Locate elements using the 'name' attribute. Not always unique, but useful for forms.

**By Link-Text**

Locate anchor tags (`<a>`) using the exact link text. Use when you know the full link text.

. **By Partial Link Text**

Locate anchor tags using a partial match of the link text. Good for dynamic text.

**By Tag Name**

. **By Class Name**

Locate elements using the class attribute. Use when elements share common styles.

**By XPath –** Relative & absolute x path – Most preferrable used

**By CSS Selector**

**Locate elements using CSS selectors (e.g., `.class Name`, `#id`). Preferred for performance.**

# Locators in Selenium

Locators are used in selenium WebDriver to find an element on a DOM. Locating elements in Selenium WebDriver is performed with the help of ***findElement()*** and ***findElements()*** methods provided by WebDriver and WebElement class.

- ***findElement()*** returns a WebElement object based on a specified search criteria or ends up throwing an exception if it does not find any element matching the search criteria.
- ***findElements()*** returns a list of WebElements matching the search criteria. If no elements are found, it returns an empty list.

There are 8 types of Locators in Selenium are as follows –

| Sr. | Method | Syntax | Locate By Using |
|---|---|---|---|
| 1 | By ID | driver.findElement(By.***id***(<element id >)) | ID Attribute |
| 2 | By Name | driver.findElement(By.***name***(<element Name>)) | Name Attribute |
| 3 | By LinkText | driver.findElement(By.***linkText***(<linkText >)) | Link Attribute |
| 4 | By PartialLinkTest | driver.findElement(By.***partialLinkTest***(<linkText >)) | Partial Link Attribute |
| 5 | By Tag Name | driver.findElement(By.***tagName***(<element HTMLTagName >)) | Tag Name Attribute |
| 6 | By Class Name | driver.findElement(By.***className***(<element class>)) | Class Name Attribute |
| 7 | By xPath | driver.findElement(By.***xPath***(<xpath >)) | Css selector |
| 8 | By Css Selector | driver.findElement(By.***cssSelector***(<css Selector>)) | xPath query |

**Example**

**Login Username :** **Password :** Login

```
<form name="loginForm">Login
 Username: <input id="username" type="text" name="login" />
 Password: < input id="password" type="password" name="password" />
 < input type="submit" name="signin" value="SignIn" /></ form >
```

## 1. Using id –

Each ***id*** is supposed to be unique couldn't be duplicated. Which makes ids a very faster and reliable way to locate elements. With ***id*** attribute value matching the location will be returned. If no element has a matching idattribute, a "***NoSuchElementException***" will be raised.

```
WebElement elementUsername = driver.findElement(By.id("username"));

WebElement elementPassword = driver.findElement(By.id("password"));
```

All objects on a page are not having id attribute, it's not realistic. In some cases developers make it havingnon-unique ids on a page or auto-generate the ids, in both cases it should be avoided.

### 2. Using Name –

By using name attribute we can find element on DOM, name attributes are not unique in a page all time. With the ***Name*** attribute value matching the location will be returned. If no element has a matching nameattribute, a "***NoSuchElementException***" will be raised.

```
WebElement elementUsername = driver.findElement(By.name("username"));

WebElement elementPassword = driver.findElement(By.name("password"));
```

### 3. Using Link –

With this you can find elements of "**a**" tags(**Link**) with the link names. Use this when you know link text used within an anchor tag.

```
<a href="link.html">Name of the Link </a>
```

Link = "Name of the Link"

```
WebElement element = driver.findElement(By.linkText("Name of the Link"));
```

### 4. Using xPath –

While DOM is the recognized standard for navigation through an HTML element tree, ***XPath is the standard navigation tool for XML and an HTML document is also an XML document (xHTML).*** XPath is used everywhere where there is XML. Xpath has a fixed structure (syntax). See below –

***// tag[@ attribute = 'value']***

Some possible syntax are as follows –

- // tag[@attribute1 = 'value' ***and*** @attribute2 = 'value']
- // tag[@attribute1 = 'value' ***or*** @attribute2 = 'value']
- // tag[@attribute1 = 'value', ***contains***(text(),'-xxxxx-')]
- // tagP[@attribute = 'value'] // innerTagOfP[@attribute1 = 'value' and @attribute2 = 'value']

By using following ways we can select username for above example :

Xpath = //*[@id='username']

Xpath = //input[@id='username']

Xpath = //form[@name='loginForm']/input[1]

Xpath = //*[@name='loginForm']/input[1]

- **Difference Between Absolute xPath and Relative xPath –**

| Sr. No. | Absolute xPath | Relative xPath |
|---|---|---|
| 1 | It uses Complete path from the Root (html) Element to the desire element. | Its not complete path from root to Element. |
| 2 | If any change is made in html code thenthis absolute xpath will get disturbed. | If any change is made in html code then thisrelative xPath will not get disturbed. |
| 3 | It is not customized xpath | It is customized type of xPath |
| 4 | It starts with / . | It starts with // . |
| 5 | It is not safe | It is safe |
| 6 | It identifies element very fast | It will take little more time in identifying the element |

- **We can use Inner Text for relative xpath –**

Use ***text(),"xxxx"***, ***contains(text(),"xxxx"), starts-with("xxxx")*** to customize the xpath.

// tag [text( ),"xxxx"]

// tag [contains(text( ),"xxxx")]

// tag [starts-with(@id, "msg") ]

**How to find xpath Dynamic Element ?**

Dynamic elements are those elements who changes is attribute on every runtime. Xpath Axes are used tofind the xpath of the such dynamic elements.

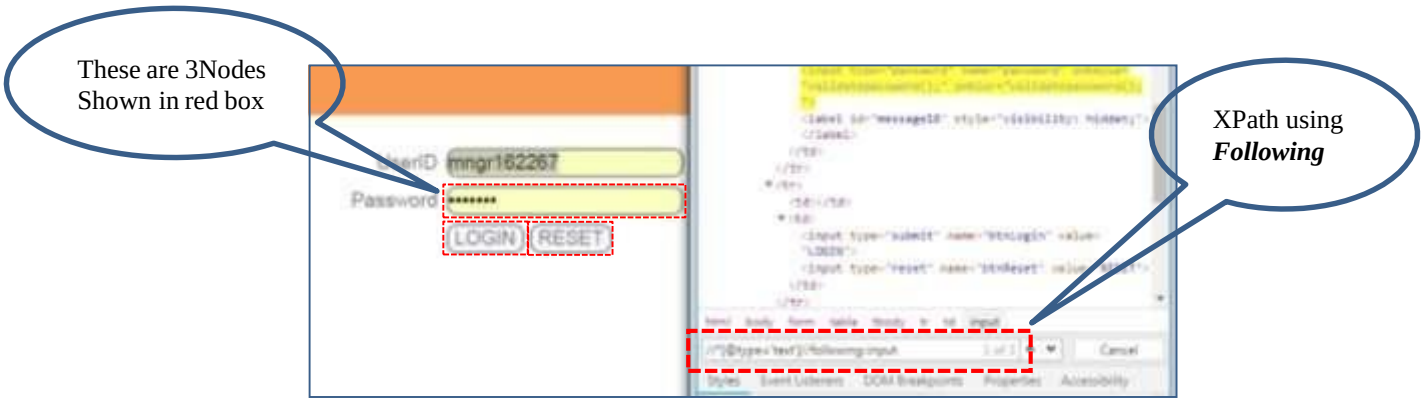
- **Xpath Axes –**

***XPath Axes are the methods used to find dynamic elements***. XPath axes search different nodes in XML document from current context node. XPath expression select nodes or list of nodes on the basis of attributes likeID , Name, Classname, etc. from the XML document .

**a) Following:**

Selects all elements in the document of the current node( ) in following image, ***input box*** is thecurrent node.

Xpath = //*[@type='text']// following :: input




There are 3 "input" nodes matching by using "following" axis- password, login and reset button. Ifyou want to focus on any particular element then you can use the below XPath method:

Xpath = //*[@type='text']// following :: input[1] → Password TextBox

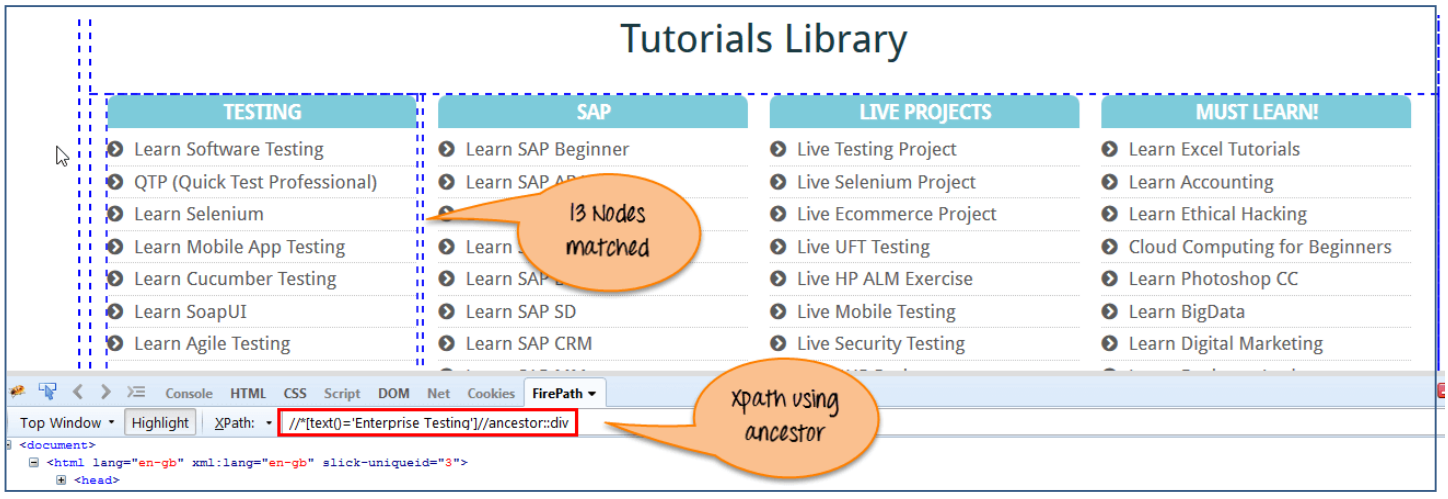
Xpath = //*[@type='text']// following :: input[2] → Login Button

Xpath = //*[@type='text']// following :: input[3] → Reset Button

**b) Ancestor:** प ज

The ancestor axis selects all ancestors element (***parent, grandparent***,...etc.) of the current node as shown in the below screen. In the below expression, we are finding ancestors element of the current node("ENTERPRISE TESTING" node).

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



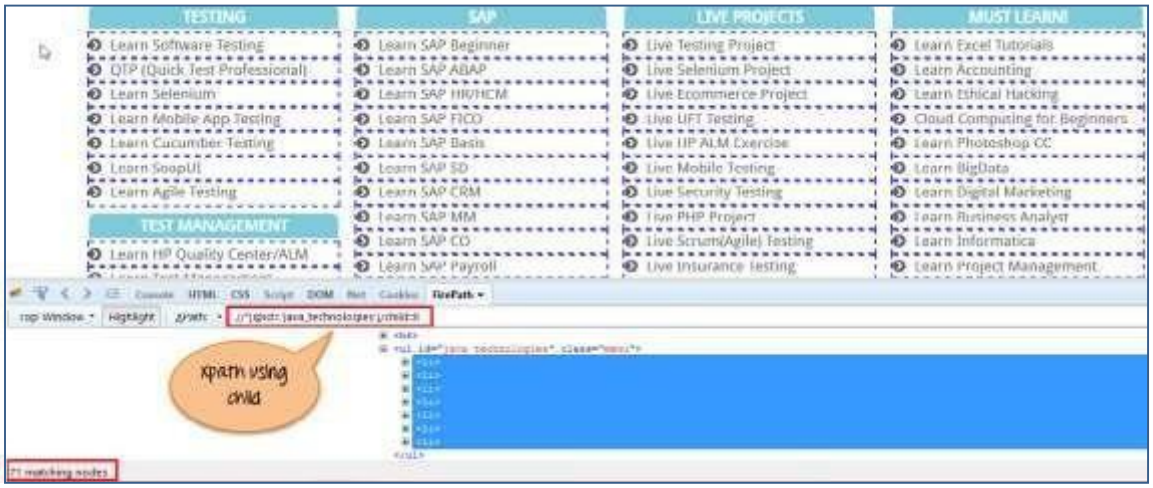

There are 13 "div" nodes matching by using "ancestor" axis. If you want to focus on any particular element then you can use the below XPath, where you change the number 1, 2,3,...13 as per your requirement:

Xpath = //*[text() = 'Enterprise Testing'] // ancestor :: div[**1**]

**c) Child:**

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

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

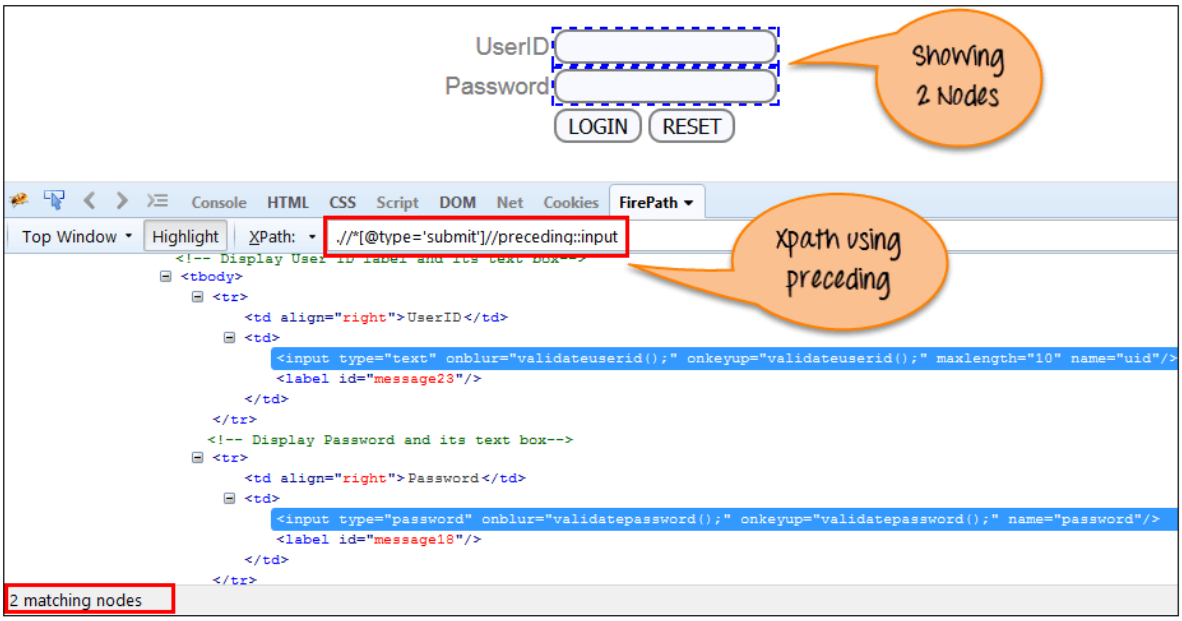



There are 71 "li" nodes matching by using "child" axis. If you want to focus on any particular element then You can change the xpath according to the requirement by putting [1],[2] ..... and so on.

Xpath = //*[@id='java_technologies']/child::li[1]

**d) Preceding:** पूर्वज

Select all nodes that come before the current node as shown in the below screen. In the below expression, itidentifies all the input elements come before "***LOGIN***" button that is **Userid** and **password** input element.




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

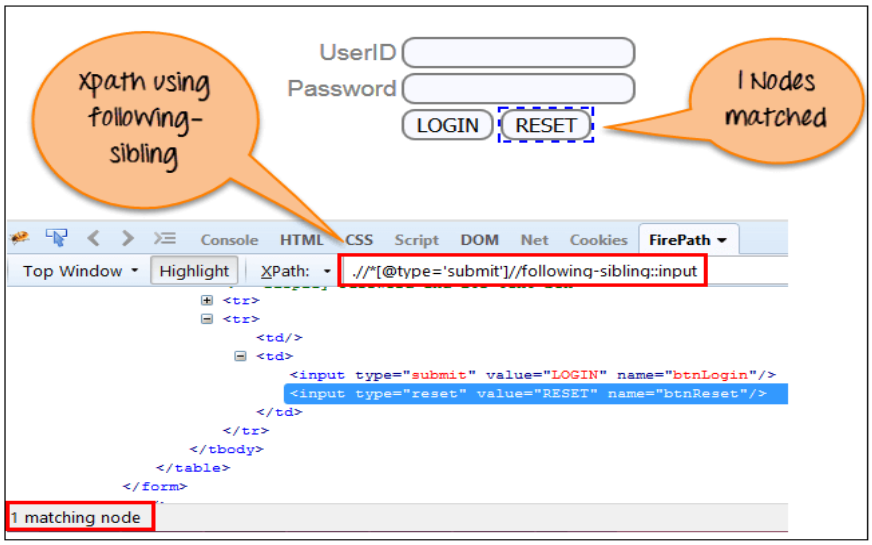
There are 2 "input" nodes matching by using "preceding" axis. If you want to focus on any particularelement then You can change the xpath according to the requirement by putting [1],[2] and so on.

Xpath = //*[@type = 'submit']//preceding::input [1]

**e) Following-sibling:** खाल ल भार्ड

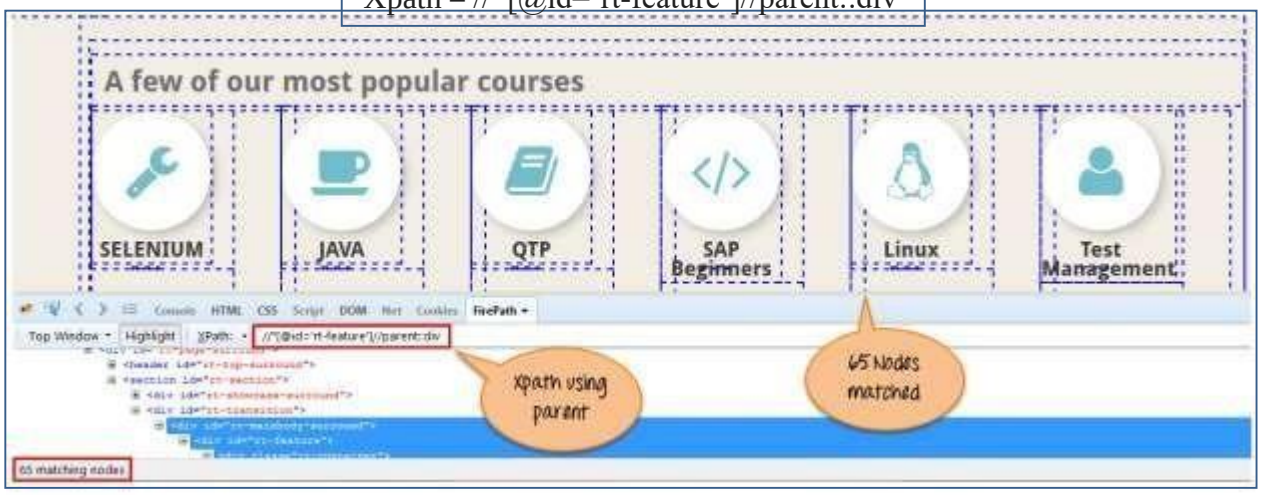
Select the following siblings of the context node. Siblings are at the same level of the current node as shownin the below screen. It will find the element after the current ***Login*** node . One input nodes matching by using "following-sibling" axis

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

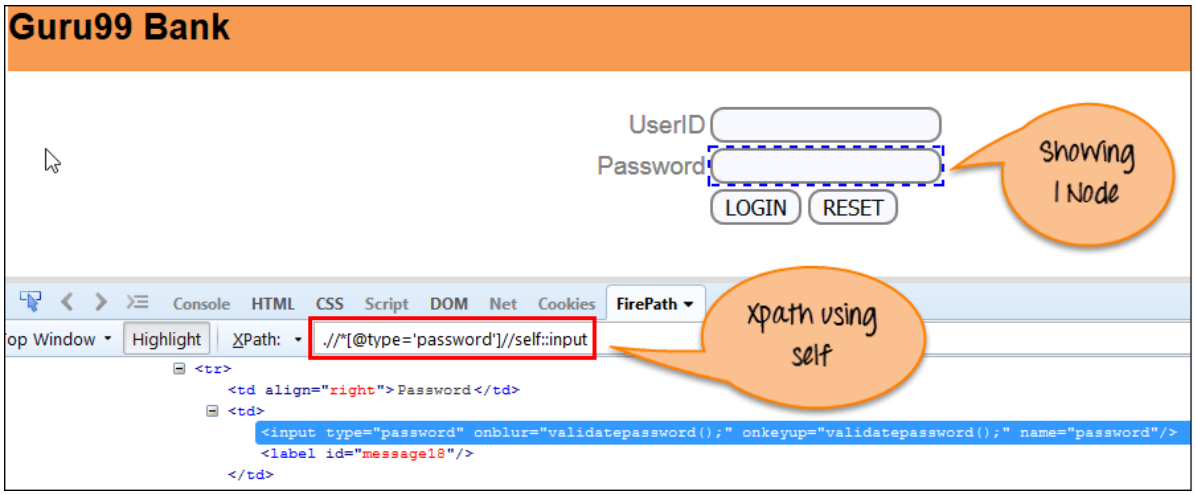



There are 65 "div" nodes matching by using "parent" axis. If you want to focus on any particular element thenYou can change the XPath according to the requirement by putting [1],[2] ..... and so on.

Xpath = //*[@id =’rt-feature’]//parent::div[1]

**g) Self:**

Selects the current node or 'self' means it indicates the node itself as shown in the below screen. One node matching by using "self " axis. It always finds only one node as it represents self-element.




Xpath = //*[@type = ‘password’]//self::input

**h) Descendant:**

Selects the descendants of the current node as shown in the below screen. In the below expression, 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-fearture’]//descendant::a


There are 12 "link" nodes matching by using "descendant" axis. If you want to focus on any particular elementthen You can change the XPath according to the requirement by putting [1],[2]. and so on.

Xpath = //*[@id = ‘rt-fearture’]//descendant::a [1]

## 5. Using CSS Selector -

There is a debate on the performance of CSS Locator and XPath locator. Most of the automation testers believe that using CSS selectors makes the execution of script faster compared to XPath locator. CSS Selector locator isalways the best way to locate elements on the page. CSS is always same irrespective of browsers.

CSS selector structure is - *Tag [ attribute = "value" ]*

In dynamic elements, there is always a part of locator which is fixed. We need to generate the locator using this fixed part

If fixed part is at starting  use (^)  e.g. input [id^='XXXXXX'] If fixed part is at mid  use (*)  e.g. input [id*='XXXXXX'] If fixed part is at end  use ($)  e.g. input [id$='XXXXXX']

Following are some of the mainly used formats of CSS Selectors.

| | |
|---|---|
| • Tag and ID<br>• Tag and Class<br>• Sub-String Matches<br>  ○ Starts With (^)<br>  ○ Ends With ($)<br>  ○ Contains (*) | • Tag and Attribute<br>• Tag, Class, and Attribute<br>• Child Elements<br>  ○ Direct Child<br>  ○ Sub-child<br>  ○ nth-child |

- **Tag and ID :** **Syntax:** css=tag#id


```
<div>
<label class="hidden-label" for="Email"> Enter your email</label>
<input id="Email" type="email" autofocus="" placeholder="Enter your email" name="Email" spellcheck="false" value=""> <input id="Passwd-hidden" class="hidden" type="password" spellcheck="false">
</div>
```

Css = input#Email

- **Tag and Class:**

If multiple elements have the same HTML tag and class, then the first one will be recognized.

**Syntax:** css=tag.class

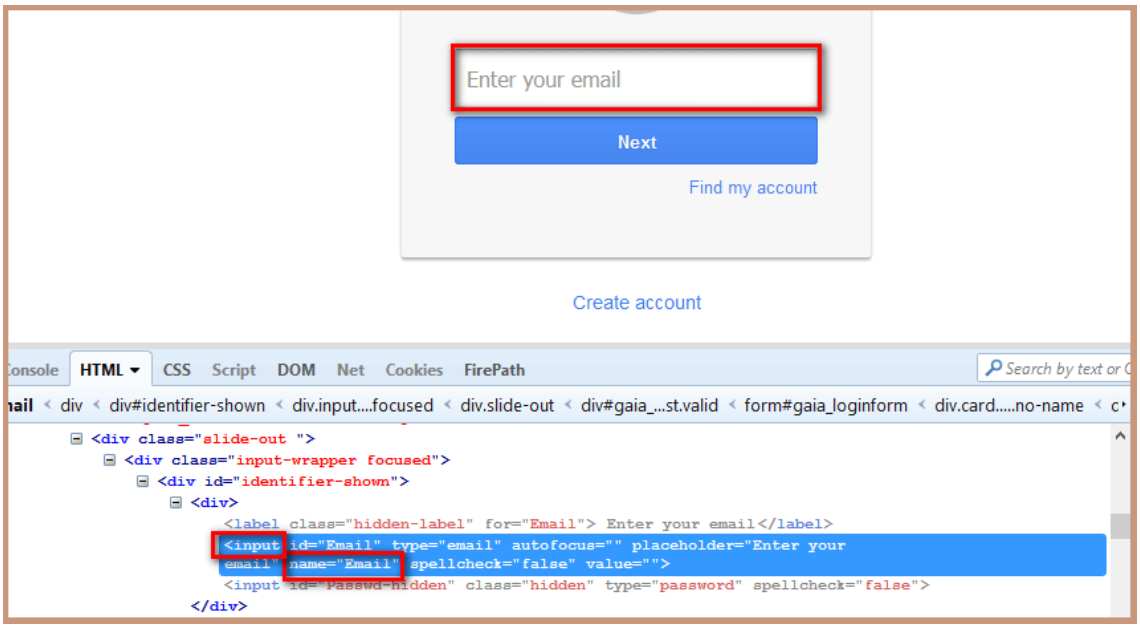

```
<td>
<input id=”email“ class=”inputtext“ type=”email“ tabindex=”1“ value=”” name=”email“>
</td>
```

```
css=input.inputtext
```

- **Tag and Attribute:**

If multiple elements have the same HTML tag and attribute, then the first one will be recognized. It acts in the same way of locating elements using CSS selectors with the same tag and class.

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



```
<div>
<label class=”hidden-label“ for=”Email“> Enter your email</label>
<input id=”Email“ type=”email“ autofocus=”” placeholder=”Enter your email“ name=”Email“
spellcheck=”false“ value=””> <input id=”Passwd-hidden“ class=”hidden“ type=”password“
spellcheck=”false“>
</div>
```

css = input[name=Email]

**Tag, Class And Attribute:**

**Syntax:** css=tag.class[attribute=value]


```
<td>
<input id="email" class="inputtext" type="email" tabindex="1" value="" name="email">
</td>
```

css=input.inputtext[name=email]

**SUB-STRING MATCHES:**

CSS in Selenium has an interesting feature of allowing partial string matches using **^**, **$**, and *****.

**Suppose**

<input="Employee_ID_001">

**Starts with (^):** To select the element, we would use ^ which means ‘*starts with*’

**Syntax: css=<HTML tag><[attribute^=prefix of the string]>**

css=input[id^='Em']

**Ends with ($): To select the element, we would use *$* which means *'ends with'.***

**Syntax:** css = <HTML tag> <[attribute$=suffix of the string]>

```
css=input[id^='001']
```

**Contains (*):** To select the element, we would use ***** which means '*sub-string*'

**Syntax:** css=<HTML tag><[attribute*=sub string]>

```
css=input[id*='id']
```

**Also we can use 'contains()':**

```
Css = "input:contains('id')"
```

**Locating Child Elements(Direct Child):**

```
<div id="buttonDiv" class="small">
<button id="submitButton" type="button"
class="btn">Submit</button>
</div>
```

**Syntax:** parentLocator>childLocator

**CSS Locator:** div#buttonDiv>button

**Explanation:** 'div#buttonDiv>button' will first go to div element with id 'buttonDiv' and then select its child element – 'button'

**Locating elements inside other elements (child or sub-child):**

**Syntax:** parentLocator>locator1 locator2

**CSS Locator:** div#buttonDiv button

**Explanation:** 'div#buttonDiv button' will first go to div element with id 'buttonDiv' and then select 'button' element inside it (which may be its child or sub child)

**Locating nth Child:**

To find nth-child css.

```
  <li>Selenium</li>
  <li>QTP</li>
  <li>Sikuli</li>
</ul>
```

To locate the element with text ‘QTP’, we have to use “nth-of-type”

```
css="ul#automation li:nth-of-type(2)"
```

Similarly, To select the last child element, i.e. ‘Sikuli’, we can use

```
css="ul#automation li:last-child"
```

----------END----------