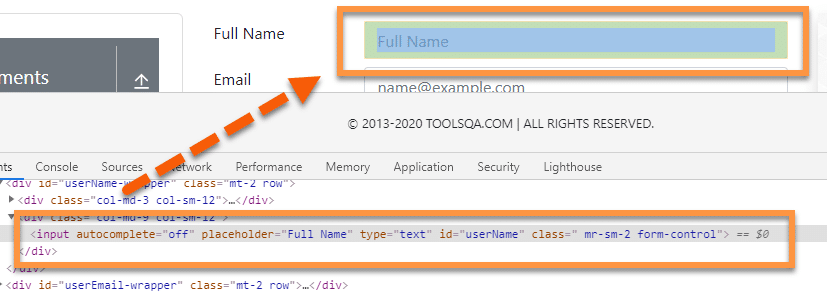
### ***Xpath Contains() function***

In this example, the contains() function matches any <button> element with an id attribute containing the text 'dynamicPart'. This approach is useful when only a portion of the attribute value is constant.

//tag\_name[contains(@attribute,'value\_of\_attribute')]

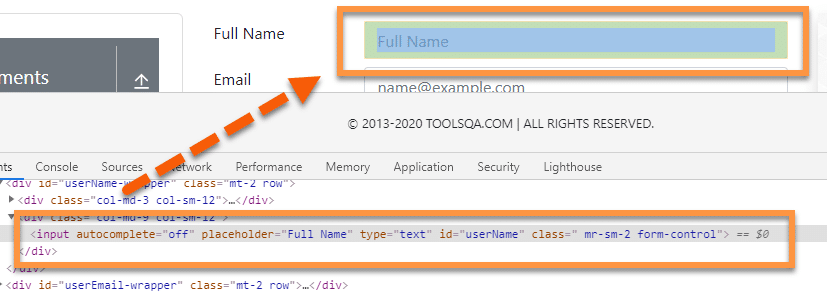


//input[contains(@id, "userN")]

### ***XPath Starts-with() function***

The XPath ***starts-with()*** function, as the name suggests, finds the element which has attribute value starting with the specific given character or character sequence. This function is quite useful while dealing with dynamic web pages. Imagine an element that has an attribute value that keeps on changing with every page load or page operation. Usually, these dynamic elements have few common starting characters, followed by random dynamic texts. Apart from the dynamic attribute, this can also identify static elements.

//tag\_name[starts-with(@attribute,'Part\_of\_Attribute\_value')]

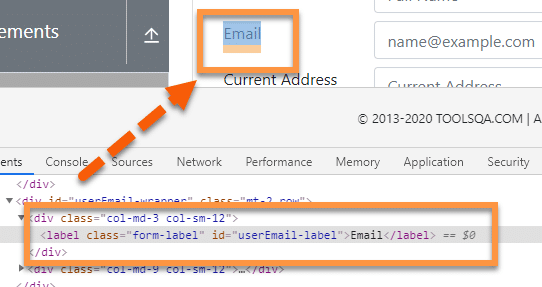


//input[starts-with(@placeholder,"Fu")]

### ***XPath Text() function***

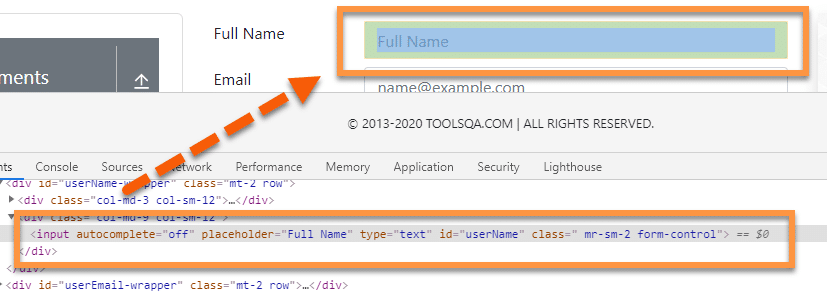
This function uses the ***text of the web element*** for locating the element on the webpage. This function is quite useful if your element contains the text, for example, labels, which always contain a static text.

//tag\_name[text()='Text of the element']



//label[text()=”Email”]

### ***AND & OR operators***

The ***"and "*** operator is used to ***combining two different conditions or attributes*** to identify any element from a webpage using XPath efficiently. For example, if we have two attributes, a and b, we can combine both to uniquely identify an element on the webpage using the ***"and "*** operator.

//input[@placeholder ='Full Name' and @type = 'text']

//input[@placeholder ='Full Name' or @type = 'text']

### ***Ends-With***

ends-with(@attribute-name, 'substring')

<ul>

<li>document.pdf</li>

<li>image.png</li>

<li>file.txt</li>

<li>presentation.pptx</li>

</ul>

//li[ends-with(text(), '.pdf')]

**Explain the difference between absolute and relative XPath. When should you use each?**

Absolute XPath specifies the complete path from the root of the HTML document to the target element. It's less flexible and more likely to break with changes.

Relative XPath specifies the path to an element relative to another element. It's more flexible and recommended for dynamic web elements.

**Explain the use of wildcards in dynamic XPath. What are the wildcard characters, and how do you use them?**

Wildcard characters like \* and @ are used in dynamic XPath to match any element or attribute. For example, //\*[contains(@id, 'partialId')] matches elements with partial IDs.

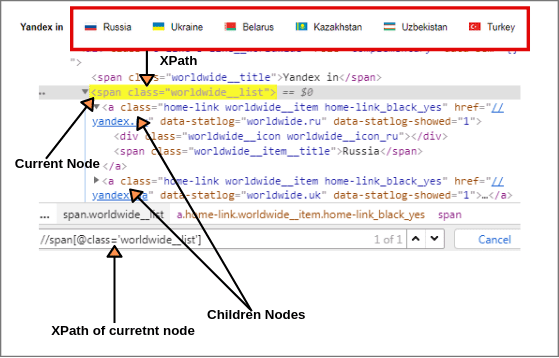
**XPath Axes Methods in Selenium**

A list of 13 XPath axes methods in Selenium WebDriver is as follows:

1. Child Axis
2. Parent Axis
3. Ancestor Axis
4. Ancestor-or-self Axis
5. Descendant Axis
6. Descendant-or-self Axis
7. Following Axis
8. Following-sibling Axis
9. Preceding Axis
10. Preceding-sibling Axis

**Child Axis:**

//child::tagName



Now we will find out XPath of children elements of current node using child axis as shown in above figure.

XPath of all children elements: //span[@class = 'worldwide\_\_list']//child::a (1 of 6 matched)

XPath(Russia): //span[@class = 'worldwide\_\_list']//child::a[1]  (1 of 1 matched)

XPath(Ukraine): //span[@class = 'worldwide\_\_list']//child::a[2] (1 of 1 matched)

XPath(Belarus): //span[@class = 'worldwide\_\_list']//child::a[3] and so on.

**Parent Axis:**

The parent axis selects the parent of the current node. The parent node may be either root node or element node. The root node has no parent.

Therefore, when the current node is root node, the parent axis is empty. For all other element nodes, the parent axis contains a maximum of one node.  
The syntax of parent axis is given below:

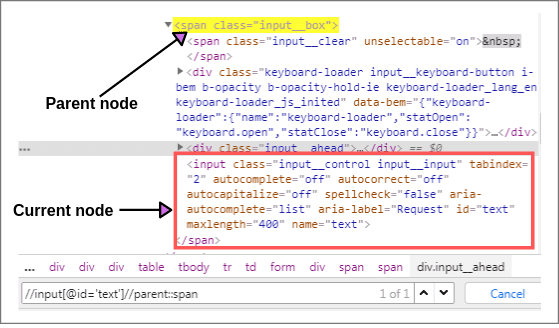
**Syntax://parent::tagName**

Let’s take a scenario for example.

Open the website yandex.com and right-click on the search box. We will find the parent of the current node. Choose a search input box as a current node and find XPath of the current node (Search box).

XPath(Current node): //input[@id = 'text'] (1 of 1 matched).

Now we will find the XPath of the parent element node of current node using parent syntax as shown below screenshot:

[](https://www.scientecheasy.com/2019/08/xpath-axes.html/)

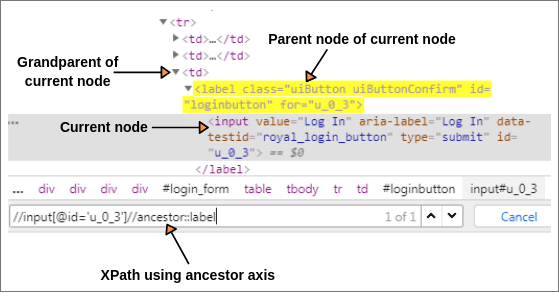
XPath(Parent node): //input[@id = 'text']//parent::span (1 of 1 matched)

It will select the parent node of the input tag of Id = ‘text’.

XPath(Parent node): //input[@id = 'text']//parent::\* (1 of 1 matched)

## Ancestor Axis:

The ancestor axis selects all ancestor elements (parent, grandparent, great-grandparents, etc.) of the current node. This axis always contains the root node (unless the current node is the root node)



XPath(Current node): //input[@id = 'u\_0\_a']

Now, we will find XPath of parent and grandparent of current node.

XPath(Parent node): //input[@id = 'u\_0\_a']//ancestor::label

XPath(Grandparent node): //input[@id = 'u\_0\_a']//ancestor::td

<!DOCTYPE html>

<html>

<head>

<title>Product Page</title>

</head>

<body>

<div class="category">

<h2>Electronics</h2>

<div class="subcategory">

<h3>Smartphones</h3>

<ul>

<li>

<span class="product-name">iPhone X</span>

<span class="product-price">$999</span>

</li>

<li>

<span class="product-name">Samsung Galaxy S20</span>

<span class="product-price">$799</span>

</li>

</ul>

</div>

<div class="subcategory">

<h3>Laptops</h3>

<ul>

<li>

<span class="product-name">Dell XPS 13</span>

<span class="product-price">$1299</span>

</li>

<li>

<span class="product-name">MacBook Air</span>

<span class="product-price">$1099</span>

</li>

</ul>

</div>

</div>

</body>

</html>

Let's say you want to validate the price of a specific product (e.g., "MacBook Air") and ensure that it belongs to the "Electronics" category and the "Laptops" subcategory.

//span[@class='product-name'][text()='MacBookAir']/ancestor::div[@class='subcategory']/ ancestor::div[@class='category']

## Ancestor-or-self Axis:

This axis selects all ancestor elements (parent, grandparent, great-grandparents, etc.) of the current node and the current node itself.

Let us find XPath of current node (login button) by using the ancestor-or-self axis.

**Scenario:** You are analyzing a webpage that provides information about different countries. Each country's information is organized into sections with meaningful section names. You want to select and extract data about a specific country along with its header and content.

Here's an example of the HTML structure:

<div class="country-section">

<h2>United States</h2>

<p>The United States is a country in North America...</p>

</div>

<div class="country-section">

<h2>France</h2>

<p>France is a European country known for its rich culture...</p>

</div>

<div class="country-section">

<h2>Japan</h2>

<p>Japan is an island nation in East Asia...</p>

</div>

You want to select and extract data about "France," including its header ("France") and content ("France is a European country known for its rich culture..."). Here's how you can achieve this using the **ancestor-or-self** axis:

//h2[text()='France']/ancestor-or-self::div[@class='country-section']

Explanation:

//h2[text()='France']: This part of the expression selects the <h2> element with the text "France," which is the header of the section you want to target.

/ancestor-or-self::div[@class='country-section']: This part uses the ancestor-or-self axis to select the <div> element with the class "country-section" that is either the parent of the <h2> element ("France") or the <h2> element itself.

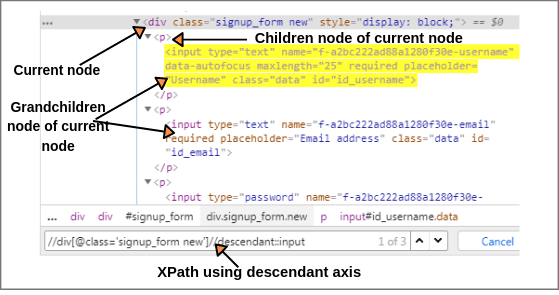
When you evaluate this XPath expression, it will select both the <h2> element with "France" and the <p> element with "France is a European country known for its rich culture." This allows you to contextually select and extract data about the country "France" based on the header's text.

In this example, the section name ("France") is meaningful, and the ancestor-or-self axis helps you select the relevant content within the context of the country section you are interested in.

## Descendant Axis:

The descendant axis selects all descendant elements (children, grandchildren, etc) of the current node. Let’s take an example to understand the concepts of the descendant axis.

As shown in the below screenshot, let’s suppose “signup\_form new” as a current node. You can bring the cursor to this node to see current node.



The XPath of current node will be as follow:

XPath(Current node): //div[@class = 'signup\_form new']

Now using the descendant axis with above XPath, we can find easily all children, grandchildren elements, etc of current node.

XPath: //div[@class = 'signup\_form new']//descendant::input (1 of 3)

The above XPath expression identified three elements like username, password, and email address. So, we can write XPath by putting 1, 2, and 3 in the above expression.

XPath(Username): //div[@class='signup\_form new']//descendant::input[1] (1 of 1 matched)

XPath(Email address): //div[@class = 'signup\_form new']//descendant::input[2]

XPath(Password): //div[@class = 'signup\_form new']//descendant::input[3]

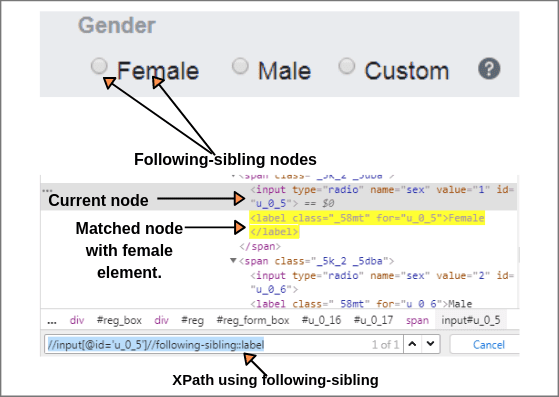
## Following Axis:

The following axis selects all elements (nodes) in the document after closing tag of the current node. Let’s consider “First name” input box as current node in the Facebook webpage.

## Following-sibling Axis:

The following-sibling selects all sibling nodes after the current node at the same level. i.e. It will find the element after the current node.

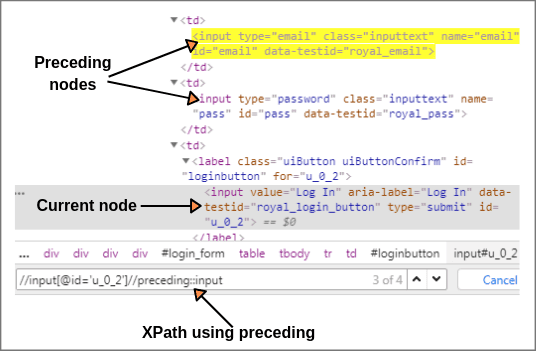
For example, the radio button of female and female text both are siblings on the Facebook home page as shown in the below screenshot.



## Preceding Axis:

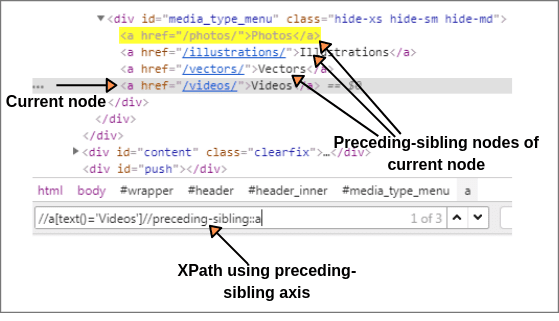
The preceding axis selects all nodes that come before the current node in the document, except ancestor, attribute nodes, and namespace nodes.

Let us consider the login button as current node on the Facebook web page as shown below screenshot.



**Preceding-sibling Axis:**

The preceding-sibling axis selects all siblings before the current node. Let’s take an example to understand the concept of the preceding-sibling axis.



<!DOCTYPE html>

<html>

<head>

<title>Following and Preceding Axes Example</title>

</head>

<body>

<div>

<h2>Heading 1</h2>

<p>Paragraph 1</p>

<p>Paragraph 2</p>

<h2>Heading 2</h2>

<p>Paragraph 3</p>

<p>Paragraph 4</p>

</div>

</body>

</html>

Using the **Following** Axis:

Select All <p> Elements Following the First <h2> Element:

//h2[1]/following::p

This expression selects both **<p>** elements ("Paragraph 1" and "Paragraph 2") that appear after the first **<h2>** element ("Heading 1").

Using the **Preceding** Axis:

Select All <h2> Elements Preceding the Third <p> Element:

//p[1]/Preceding::h2

This expression selects the **<h2>** element ("Heading 2") that appears before the third **<p>** element ("Paragraph 3").