**SELENIUM LOCATORS**

**HTML Tags & Attributes**

**HTML tags** are like keywords which defines that how web browser will format and display the content. With the help of tags, a web browser can distinguish between an HTML content and a simple content. HTML tags contain three main parts: opening tag, content and closing tag.

<!DOCTYPE html>

<html>

<head>

<title>Paragraph Example</title>

</head>

<body>

<p>Here is a first paragraph of text.</p>

<p>Here is a second paragraph of text.</p>

<p>Here is a third paragraph of text.</p>

</body>

</html>

**HTML attributes** provide additional information about HTML elements.

<a href="https://www.w3schools.com">Visit W3Schools</a>

<img src="img\_girl.jpg">

<img src="img\_girl.jpg" width="500" height="600"> etc.

**What is locator?**

Locators provide a way to access the HTML elements from a web page. In Selenium, we can use locators to perform actions on the text boxes, links, checkboxes and other web elements. They are the basic building blocks of a web page. A web developer must use a proper and consistent locator scheme for a website.

How to locate webElement?

1. Right Click on the webpage and click Inspect.

2. By default, it will open the “Elements” tab, which represents the complete ***DOM*** structure of the web page.

3. Click on mouse arrow & select the WebElement, It will highlight the Web Element.

**Locators in Selenium?**

**ID, Name , XPath, CSS Selector**

To access all these locators, Selenium provides the “[***By***](https://www.selenium.dev/selenium/docs/api/java/org/openqa/selenium/By.html#:~:text=Find%20elements%20based%20on%20the%20value%20of%20the%20%22class%22%20attribute,and%20%22two%22%20will%20match.)” class, which helps in locating elements within the DOM.  It offers several different methods (some of which are in the image below) like ***className, cssSelector, id, linkText, name, partialLinkText, taName,*** and ***xpath, etc.,***which can identify the web elements based on their corresponding locator strategies.

Demo URL: https://demoqa.com/automation-practice-form

Graphical user interface, text, application

Description automatically generated

Locators Identification:

By.id(“SpecifyId”)

By.name(“SpecifyName”)

By.className(“Specify\_ClassName”)

By.tagName(“Specify\_TagName”)

By.linkText(“Specify\_LinkText”)

By.partialLinkText(“Specify\_partialLinkText”)

XPath Syntax:

//tagName[@attribute=’attribute\_value’]

//input[@id=’firstName’]

CSS Selector:

tagName[attribute=’attribute\_value’]

**Difference between Absolute Xpath and Relative Xpath?**

**Absolute Xpath**: It contains the complete path from the Root Element to the desire element.

**Relative Xpath**: This is more like starting simply by referencing the element you want and go from the particular location [Refer to above Example]

**WEBELEMENT COMMANDS**

All interesting operations to do with interacting with a page will be performed through this ***WebElement Interface.***

Before going through each and every action of WebElement, let’s just understand that how we get a WebElement object/element. As in the previous chapters, we learned that every method of the **WebDriver** either returns something or return void(means return nothing). The same way ***findElement*** command of ***WebDriver*** returns ***WebElement***.  


So, to get the WebElement object write the below statement:

***WebElement element = driver.findElement(By.id(“UserName“));***

And now if you type ***element dot***, Eclipse’s intellisence will populate the complete list of actions like below

Chart

Description automatically generated with low confidence

One more thing to notice that WebElement can be of any type, like it can be a ***Text, Link, Radio Button, Drop Down, WebTable*** or any HTML element. But all the actions will always populate against any element irrespective of whether the action is valid on the WebElement or not. For e.g. ***clear() command***, even if you have a link element still you get the option to choose clear() command on it, which if you choose may result in some error or may not does anything.

**Element inspector in Browsers**

**Chrome**As discussed above, We can inspect the element with Inspect option  
We also have extensions which help in inspecting xpath automatically like ChroPath but those should be avoided during learning phase.

**Mozilla**

FireBug – It is a plugin which needs to be installed in Browser which will help in inspecting the WebElement.

**IE**

Find F12 Developer tools and tools section in Browser and you can easily inspect the WebElement.

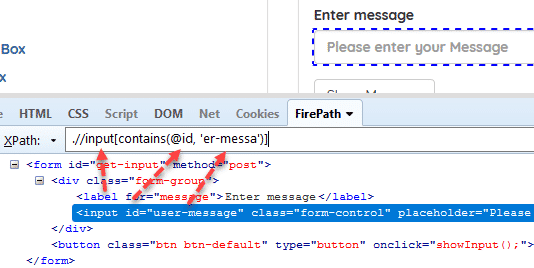
**Effective/Different ways to write X-path**

**Contains**

It is very handy XPath Seleniumlocator and sometimes it saves the life of a test automation engineer. When an attribute of an element is dynamic, then you can use contains() for the constant part of the web element but also you can use contains() in any condition when you need.

**Syntax: //tag[contains(@attribute, ‘value‘)]**

**Example:***//input[contains(@id, ‘user-message)]*

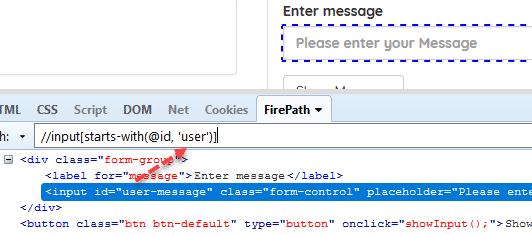
[](https://548225-1759080-raikfcquaxqncofqfm.stackpathdns.com/wp-content/uploads/2017/09/xpath-contains.png)

**Starts-with**

This method checks the starting text of an attribute. It is very handy to use when the attribute value changes dynamically but also you can use this method for non-changing attribute values.

**Syntax: //tag[starts-with(@attribute, ‘value‘)]**

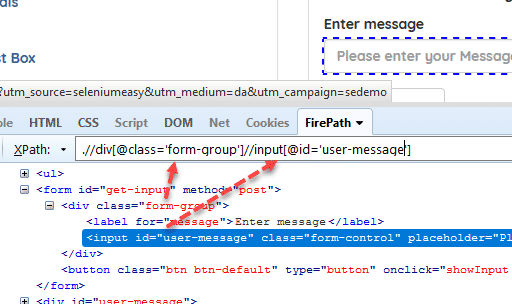
**Example:***//input[starts-with(@id, ‘user’)]*

[](https://548225-1759080-raikfcquaxqncofqfm.stackpathdns.com/wp-content/uploads/2017/09/xpath-starts-with.png)

**Chained Declarations**

We can chain multiple relative XPath declarations with **“//” double slash** to find an element location as shown below.

**Example:***//div[@class=’form-group’]//input[@id=’user-message’]*

[](https://548225-1759080-raikfcquaxqncofqfm.stackpathdns.com/wp-content/uploads/2017/09/chained-xpath.png)

**Operator “or”**

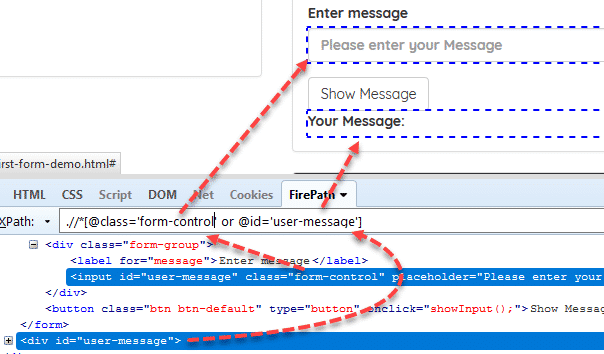
In this method, we use two interrogation conditions such as A and B and return a result-set as shown below:

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **Result** |
| False | False | No Element |
| True | False | Returns A |
| False | True | Returns B |
| True | True | Returns Both |

**“or” is case-sensitive**, you should not use capital “OR”.

**Syntax: //tag[XPath Statement-1 or XPath Statement-2]**

**Example:***//\*[@id=’user-message’ or @class=’form-control’]*

[](https://548225-1759080-raikfcquaxqncofqfm.stackpathdns.com/wp-content/uploads/2017/09/xpath-or-statement.png)

**Operator “and”**

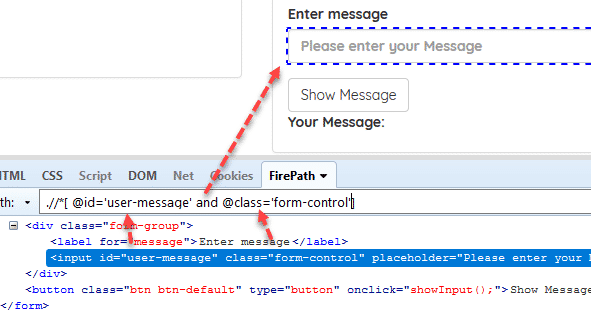
In this method, we use two interrogation conditions such as A and B and return a result-set as shown below:

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **Result** |
| False | False | No Element |
| True | False | No Element |
| False | True | No Element |
| True | True | Returns Both |

**“and” is case-sensitive**, you should not use capital “AND”.

**Syntax: //tag[XPath Statement-1 and XPath Statement-2]**

**Example:***//\*[@id=’user-message’ and @class=’form-control’]*

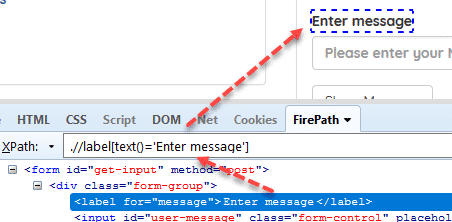
[](https://548225-1759080-raikfcquaxqncofqfm.stackpathdns.com/wp-content/uploads/2017/09/xpath-and-statement.png)

**Text**

We can find an element with its exact text.

**Syntax: //tag[text()=’text value‘]**

**Example:***.//label[text()=’Enter message’]*

[](https://548225-1759080-raikfcquaxqncofqfm.stackpathdns.com/wp-content/uploads/2017/09/xpath-text.png)

**XPATH AXES**

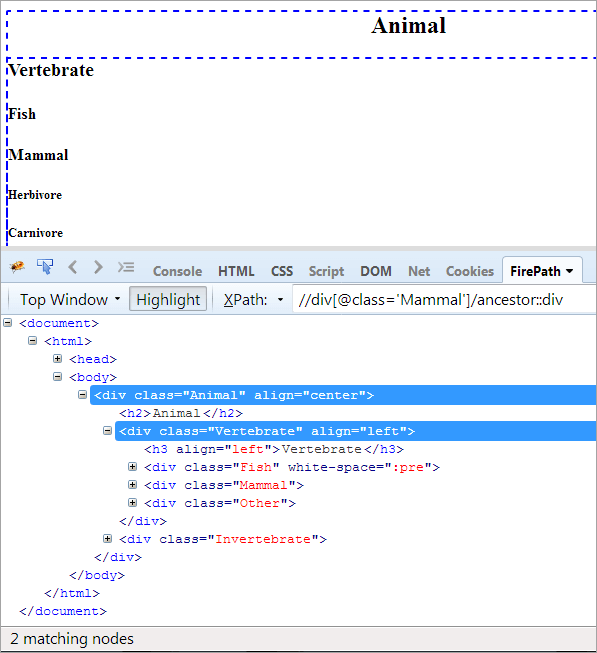
**Diagram

Description automatically generated**

**#1) Ancestor**

**Agenda:** To identify the ancestor element from the context node.

**XPath#1:** //div[@class=’Mammal’]/ancestor::div

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2020/03/Image-17.png)

**The XPath “//div[@class=’Mammal’]/ancestor::div” throws two matching nodes:**

* Vertebrate, as it is the parent of “Mammal”, hence it is considered the ancestor too.
* Animal as it the parent of the parent of “Mammal”, hence it is considered an ancestor.

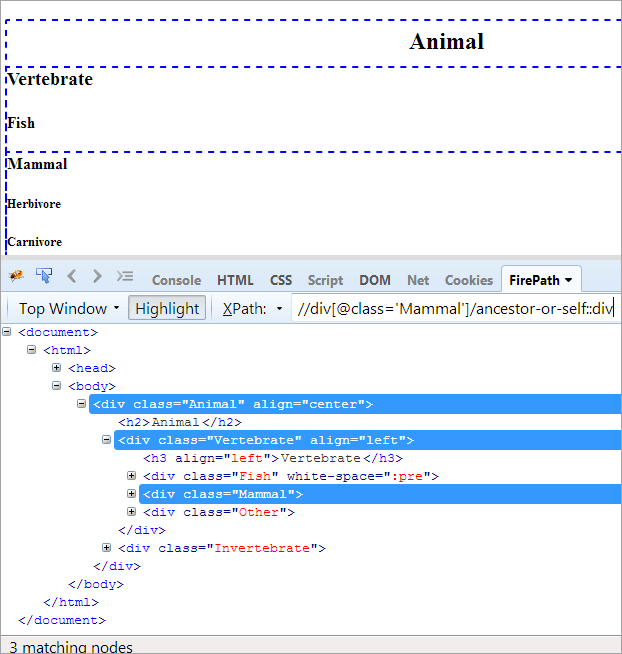
Now, we only need to identify one element that is “Animal” class. We can use the XPath as mentioned below.

**XPath#2:** //div[@class='Mammal']/ancestor::div[@class='Animal']

**#2) Ancestor-or-self**

**Agenda:** To identify the context node and the ancestor element from the context node.

**XPath#1:**//div[@class=’Mammal’]/ancestor-or-self::div

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2020/03/Image-20.png)

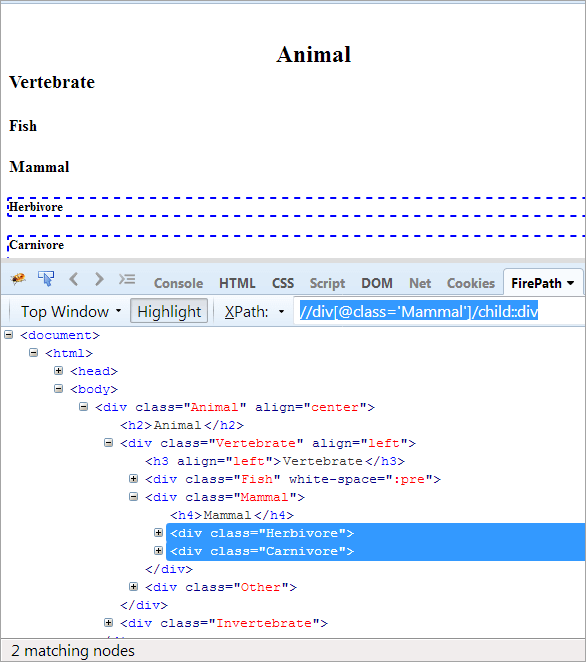
**The above XPath#1 throws three matching nodes:**

* Animal(Ancestor)
* Vertebrate
* Mammal(Self)

**#3) Child**

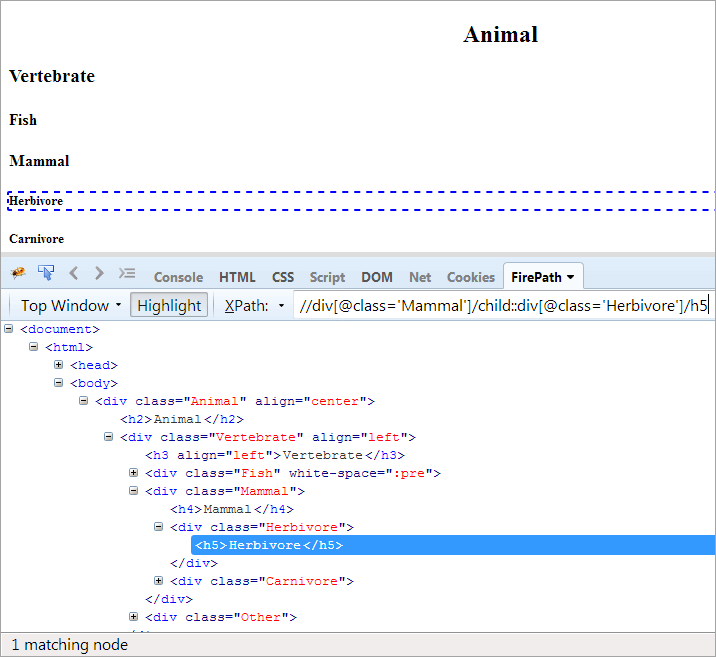
**Agenda:** To identify the child of context node “Mammal”.

**XPath#1:**//div[@class=’Mammal’]/child::div

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2020/03/Image-21.png)

**XPath#1** helps to identify all the children of context node “Mammal”. If you want to get the specific child element, please use XPath#2.

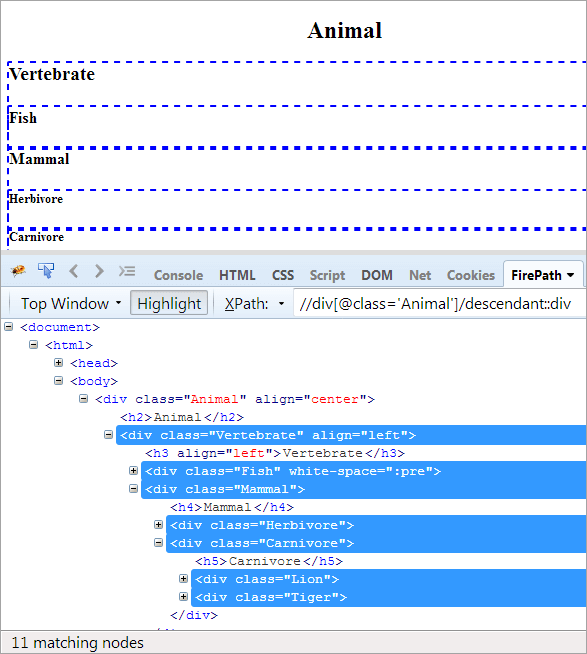
**XPath#2:**//div[@class=’Mammal’]/child::div[@class=’Herbivore’]/h5

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2020/03/Image-22.png)

**#4) Descendent**

**Agenda:** To identify the children and grandchildren of the context node (for instance: ‘Animal’).

**XPath#1:**//div[@class=’Animal’]/descendant::div

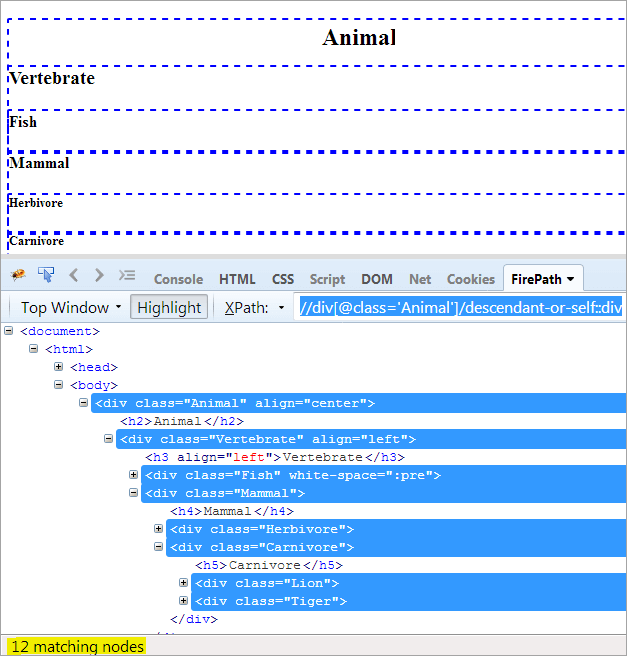
[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2020/03/Image-23.png)

As Animal is the top member in the hierarchy, all the child and descendant elements are getting highlighted. We can also change the context node for our reference and use any element we want as the node.

**#5) Descendant-or-self**

**Agenda:** To find the element itself, and its descendants.

**XPath1:**//div[@class=’Animal’]/descendant-or-self::div

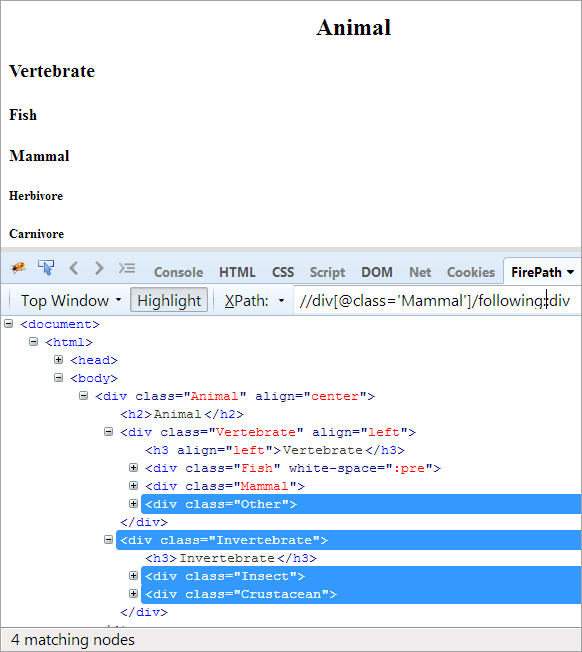
[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2020/03/Image-24.png)

The only difference between descendent and descendent-or-self is that it highlights itself in addition to highlighting the descendants.

**#6) Following**

**Agenda:** To find all the nodes that follow the context node. Here, the context node is the div that contains the Mammal element.

**XPath:**//div[@class=’Mammal’]/following::div

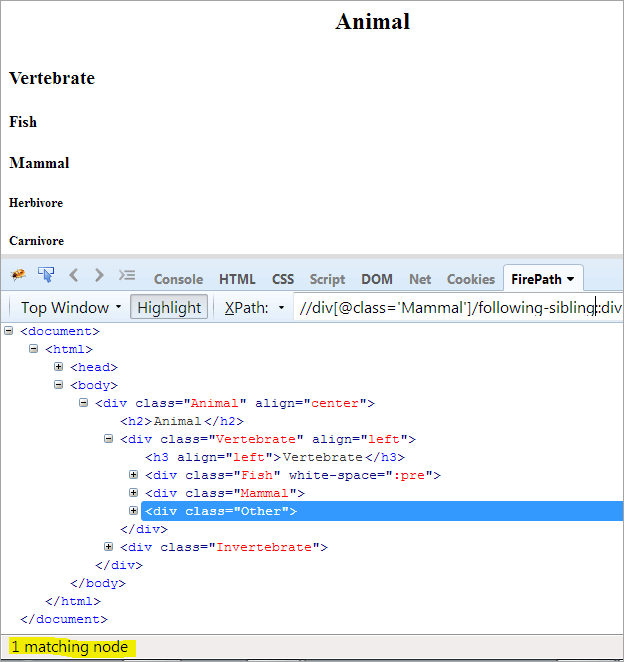
[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2020/03/Image-25.png)

In the following-axes, all the nodes that follow the context node, be it the child or descendant, are getting highlighted.

**#7) Following-sibling**

**Agenda:** To find all the nodes after the context node that share the same parent, and are a sibling to the context node.

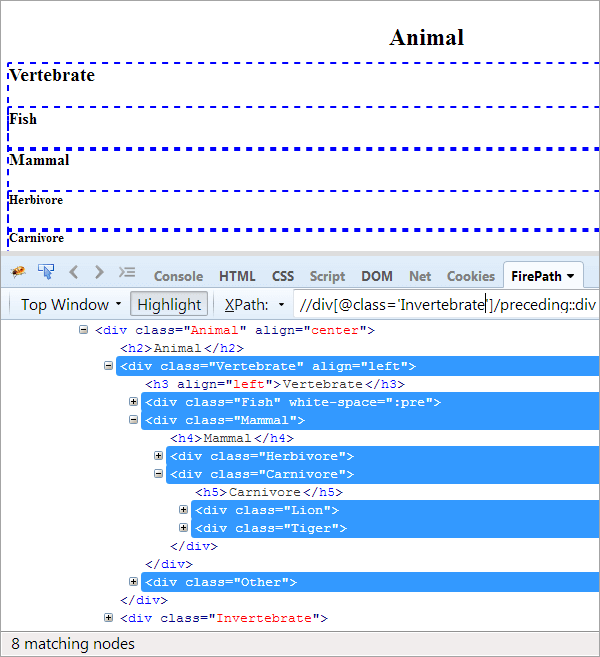
**XPath:**//div[@class=’Mammal’]/following-sibling::div

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2020/03/Image-26.png)

The major difference between the following and following-sibling is that the following-sibling takes all the sibling nodes after the context but will also share the same parent.

**#8) Preceding**

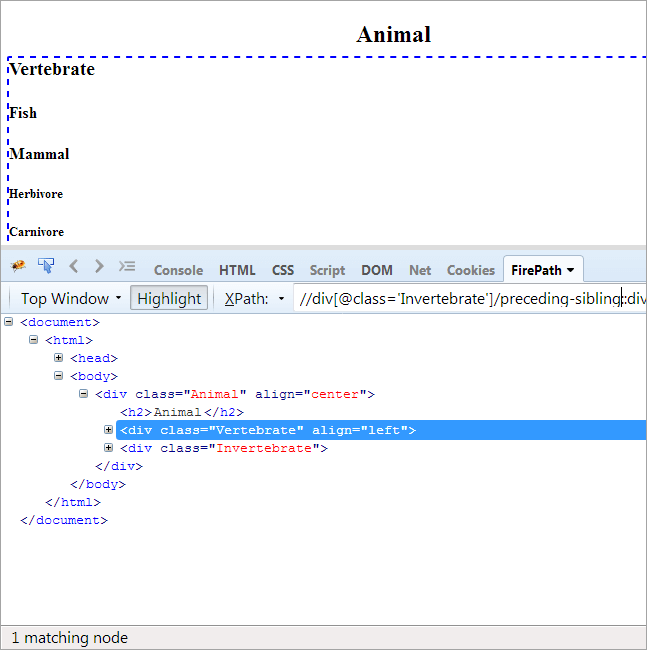
**Agenda:**It takes all the nodes that come before the context node. It may be the parent or the grandparent node.

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2020/03/Image-27.png)

Here the context node is Invertebrate and highlighted lines in the above image are all the nodes that come before the Invertebrate node.

**#9) Preceding-sibling**

**Agenda:**To find the sibling that shares the same parent as the context node, and that comes before the context node.

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2020/03/Image-28.png)

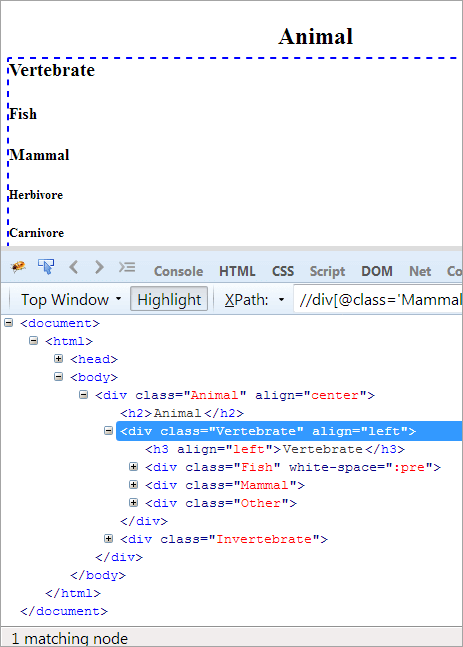
As the context node is the Invertebrate, the only element that is being highlighted is the Vertebrate as these two are siblings and share the same parent ‘Animal’.

**#10) Parent**

**Agenda:**To find the parent element of the context node. If the context node itself is an ancestor, it won’t have a parent node and would fetch no matching nodes.

**Context Node#1: Mammal**

**XPath:**//div[@class=’Mammal’]/parent::div

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2020/03/Image-29.png)

As the context node is Mammal, the element with Vertebrate is getting highlighted as that is the parent of the Mammal.s