Hackr.io's XPath Cheat Sheet

This XPath cheat sheet comes in handy when you need to have a quick look at XPath syntax and various other aspects of XPath locators.

First, let's explore exactly what the XPath web location in Selenium means.

What is XPath Web Locator in Selenium?

XPath uses the path-like syntax that helps us identify and navigate the nodes found in XML and HTML documents. In other words, it uses non-XML syntax that makes it flexible to address various parts of an XML document. Moreover, XPath follows the XSLT (eXtensible Stylesheet Language Transformations) standard, which is commonly used for navigating the WebElements and attributes.

DOM is essential to navigate through HTML documents. It works as a map containing all WebElements and the one you are looking for. You can find the desired WebElements from DOM using the appropriate web locator.

Different Types of XPath Locators

There are two ways to locate the desired WebElement in the DOM. One is through the absolute path, and the other is through the relative path. In this section of our XPath cheat sheet, we shall look at different ways of using an XPath locator to find the desired WebElement in the DOM.

Absolute XPath

Absolute path specifies the complete path from the root element to the element you want to use. But, you might face issues using the absolute path. If there is any change made within the path of the element, it results in XPath failure.

Whenever you use the absolute path, you must begin the XPath using the single forward-slash (/). This indicates you're selecting the element from the document's root node.

Syntax:

```
/html/body/div[x]/div[y]/
```

Example:

```
/html//div/div/div/div[1]/div/a/img
```

To locate an element, you can right-click on the web element and click on Inspect. You will then see the Elements tab, where you can write the locator. In this case, we started from the HTML tag and traversed one by one to the div, containing the tag up to the final img tag.

Even though it seems simple, the common disadvantage is that even a small change in the DOM structure will lead to several automation failures.

Relative XPath

Unlike the absolute path, the relative XPath refers to an element that we want to locate in a *specific* location. In this case, the element is positioned relative to its actual position.

To use the relative path, you must specify the double forward-slash (//) at the beginning. Mostly, the relative XPath is preferred for Selenium automation testing. If there is any change to the page design or the DOM hierarchy, the existing XPath selector won't be impacted.

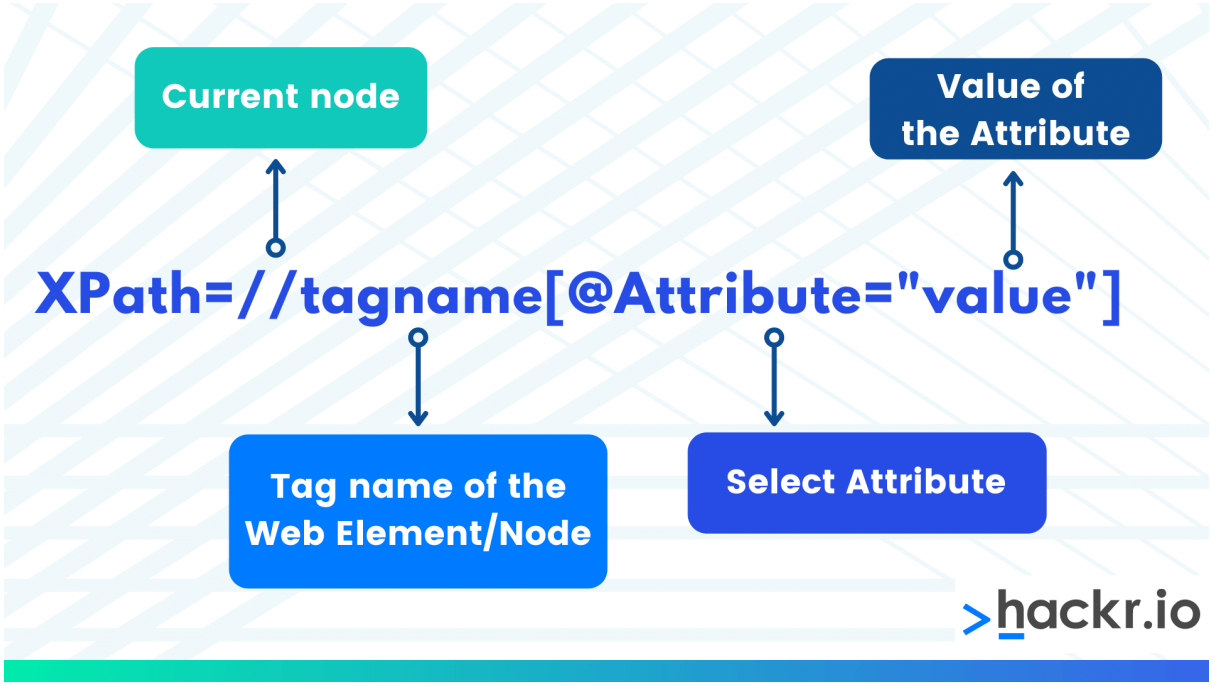
Syntax:

```
//tagname[@attribute='value']
```

XPath Syntax

In order to select the nodes in an HTML document, XPath uses this path expression:

```
XPath=//tagname[@Attribute="Value"]
```



Here are some popular Path expressions for selecting a node in an XML document:

Expression / XPath	Description
node	Select all elements with the name
/	Select from the root node
//	Select elements in the document from the current element that matches the selection (no matter where they are)
.	Select the current node
..	Select the parent of the current node element.
@	Select the attributes

XPath Expressions

XPath expressions are patterns used to select a set of nodes and carry out transformations. XPath uses the path expression for selecting nodes from XML and HTML documents.

Syntax:

```
//tagname[@attribute='value']
```

Example:

```
<?xml version="1.0" standalone="yes"?>
<empinfo>
  <employee id="1">
    <name>Opal Kole</name>
    <designation discipline="web" experience="3 year">Senior
Engineer</designation>
    <email>OpalKole@myemail.com</email>
  </employee>
  <employee id="2">
    <name from="CA">Max Miller</name>
    <designation discipline="DBA" experience="2 year">DBA
Engineer</designation>
    <email>maxmiller@email.com</email>
  </employee>
  <employee id="3">
    <name>Beccaa Moss</name>
    <designation discipline="appdev">Application Developer</designation>
    <email>beccaamoss@email.com</email>
```

```
  </employee>
</empinfo>
```

You'll find the axis and corresponding steps below.

Axis	//	/
Step	ul	a[@id='link']

Prefixes Expression

We can use prefixes in XPath at the beginning of the XPath expression:

Expression	Example	Description
//	//p[@class='footer']	Select paragraph tag from anywhere in the DOM
/	/a	Find all anchor (a) tags from the given node
/	/html/body/div	Find all div, starting from root element

Here are some examples of various steps available in XPath:

XPath	Description
//div	Select all div tags
//[@id='btn']	Select all elements with ID 'btn'
//div[@name='post']	Selecting all div elements with the name 'post'
//a/text()	Gives the text of all the anchor tag(s)
//a/@href	Gives href value of anchor tag

Selecting Nodes

The following table includes XPath expressions for selecting nodes or WebElements.

XPath	Description
div	Selects all div elements
/div	Selects div element from the root element
div/tr	Select all tr elements that are children of div
//tr	Select all tr elements anywhere in the document

div//tr	Selecting all tr elements that are children of div anywhere in the document
//@class	Select all class attributes

Predicates for Finding Nodes

You can use predicates in XPath to locate a specific node containing a designated value. They're enclosed in square brackets, as shown in the table below.

These predicate identifiers return the boolean values (true or false). You can even use the relational and boolean operators with them.

XPath	Description
/div/a[1]	Selects the first anchor element within the div element
/div/a[last()]	Selects the last anchor element within the div element
/div/a[last()-1]	Selects the second-last anchor element within a div element
/div/a[position()<3]	Select the first two anchor elements within a div element
//a[@class]	Select all anchor elements with class attribute
//a[@class='btn']	Select all anchor elements with class attribute and value 'btn'
/div/h1[1]/a	Select all anchor elements within h1 that are children of the div element
/div/h1/a[1]	Select all anchor elements that are children of h1 within a div element

For example:

Expression	Description
element_name[N]	Opening and closing square bracket referring to a specific element sequence number (array sequence).
empinfo/employee[2]	Second Child of child element; Selects the second employee of empinfo element.
empinfo/employee[2]/name	Sequence of child element; Selects name element of second employee element of empinfo element.
element_name[@attribute_name] empinfo/employee[@id]	Selects element with attribute Selects all employee elements with a given id attribute.

element_name[@attribute_name=value] empinfo/employee[@id=2]	Selects element with specified attribute value; Finding employee elements with given attributes and values. Matching only those elements whose attribute and value are specified in the XPath expression.
empinfo/employee[@id=2][2]	Selects all employee elements of given attribute and value. At last, select only the second employee element.
empinfo/employee[1][@id=1]	Selects the first employee element with the given attribute and value.
//designation[@discipline and @experience]	Selects all the designation elements with given first and second both attributes.
//designation[@discipline or @experience] //designation[@discipline | @experience]	Selects all the designation elements with either first or second, both attributes.

Chaining Order

The meaning of XPath changes with the change in order.

For example:

a[1][@href='/'] and
a[@href='/'][1] are different.

Indexing

You have to use [] for indexing in XPath, where [] contains a number that specifies the node you want to select. You can also use functions for indexing in XPath, such as last(), position(), and others, to specify the index of the elements.

For example:

//a[1]	Select the first anchor tag
//a[last()]	Select the last anchor tag
//ul/li[2]	Select the second li tag (a child of ul tag)
//ul/li[position()=2]	Select second li tag which is a child of ul tag

//ul/li[position()>1]	Select li tag which is not a first child of ul tag

Expressions for Selecting Unknown Nodes (Wildcards)

You can use the wildcard with the XPath locator to find the unknown HTML document elements.

*	Matches any HTML element
@*	Matches any attribute of an element
node()	Matches any kind of element
/div/*	Select all children of a div element
//*	Select all elements in the HTML document
//a[@*]	Select all anchor elements with any attribute
.	Matches any HTML element
..	Refers to a parent context node
/	Refers to a root node
//	Refers to a node anywhere within the document
*	Refers to all elements that are child of the context node
| or	Refers to an OR condition combine expression either first expression or second expression
and	Refers to a condition and combining expression
text()	Selects all text node children of the current element.

Expressions for Selecting Several Paths

You can use the '|' operator in the XPath expression to select several paths. Here's how you use the '|' operator:

//div | //a	Select all div and anchor elements in the HTML document
//div/h1 | //div/a	Select all h1 and anchor elements within a div element

XPath Axes

There are 13 axes available in XPath specifications. These axes represent the relationship between the context node or referred node. 13 axes are defined in XPath, enabling you to search different node parts in an XML document from the current context node or the root node. XPath Axes select the nodes from the context node within the document.

Axes example and Node test

AxesName ::= 'self'
|'child'
| 'descendant'
| 'descendant-or-self'
| 'parent'
| 'ancestor'
| 'ancestor-or-self'
| 'attribute'
| 'following'
| 'following-sibling'
| 'preceding'
| 'preceding-sibling'
| 'namespace'

- self: Axes select the context node.
- child: Axes select the child of the context node.
- descendant: Axes select all descendants of the context node, a child in any level depth.
- descendant-or-self: Axes select all descendants of the context node, a child in any level depth also selects the context node for itself.
- parent: Axes select the parent node of the context node.
- ancestor: Axes select all parent nodes of the context node until the root node.
- ancestor-or-self: Axes select all parent nodes of the context node until the root node. also, select the context node to itself.
- attribute: Axes select attributes of the context node.
- following: Axes select all nodes after the context node, excluding attributes node or namespaces node.
- following-sibling: Axes select all following sibling of the context node. It selects none, If context node is attributes node or namespace node following sibling empty.
- preceding: Axes select all nodes before the context node, excluding attributes node or namespace node.
- preceding-sibling: Axes select attributes of the context node.
- namespace: Axes select all namespace node of the context node.

Syntax:

```
AxesName::node[predicate]
```

Where:

- Predicate specifies the sequence of the nodes enclosed to a [].
- Axes name and node are separated by ::.

Example:

```xml
<?xml version="1.0" standalone="yes"?>
<empinfo>
    <employee id="1">
        <name>Opal Kole</name>
        <designation discipline="web" experience="3 year">Senior
Engineer</designation>
        <email>OpalKole@myemail.com</email>
    </employee>
    <employee id="2">
        <name from="CA">Max Miller</name>
        <designation discipline="DBA" experience="2 year">DBA
Engineer</designation>
        <email>maxmiller@email.com</email>
    </employee>
    <employee id="3">
        <name>Beccaa Moss</name>
        <designation discipline="appdev">Application
Developer</designation>
        <email>beccaamoss@email.com</email>
    </employee>
</empinfo>
```

Node Test

Node test is a part of XPath expression for finding nodes in XML documents.

Select Axis	Description
//name/self::*	Selects the name context node
child::*	Selects all child nodes of the context node.
child::node()	Selects all child nodes of the context node
child::empinfo	Selects all child elements of empinfo node
//employee/descendant::*	Selects all descendants of the employee node
//descendant::employee	Selects all descendants of the employee node in context node.
//employee/descendant-or-self::*	Selects all descendants of the employee nodes and context node itself.
//descendant-or-self::employee	This axis selects all descendant of employee node with context node itself.

Ancestors

//employee/ancestor::*	Select all ancestor nodes of the employee node
//ancestor::name	Select all ancestors of the name node in context node
//employee/ancestor-or-self::*	Select all ancestors of the employee nodes and the context node itself.
//name/ancestor-or-self::employee	Select all ancestors of the name node with the context node itself
//name/parent::*	Select the parent node of the name context node
//name/parent::employee	Return result node if employee node is parent node of the context node; otherwise no node found.
//attribute::id	Select all nodes with id attribute
//attribute::*	Select all nodes with any attribute
//employee[@id=1]/following::*	Select all nodes (with child nodes) after the context node
//employee[@id=1]/following-sibling::*	Select all sibling nodes after the context node
//employee[@id=3]/preceding::*	Select all nodes (with child nodes) before the context node
//employee[@id=3]/preceding-sibling::*	Select all sibling nodes before the context node

For example:

```
<?xml version="1.0" standalone="yes"?>
<empinfo>
  <employee id="1">
    <name>Opal Kole</name>
    <designation discipline="web" experience="3 year">Senior Engineer</designation>
    <email>OpalKole@myemail.com</email>
  </employee>
  <employee id="2">
    <name from="CA">Max Miller</name>
    <designation discipline="DBA" experience="2 year">DBA Engineer</designation>
    <email>maxmiller@email.com</email>
  </employee>
  <employee id="3">
    <name>Beccaa Moss</name>
    <designation discipline="appdev">Application Developer</designation>
```

16. <email>beccaamoss@email.com</email>
17. </employee>
18. </empinfo>

Where:

- //name/self::* (name node value- 4,9,14 line number)
- child::* (3 to 7, 8 to 12, 13 to 17 lines)
- child::node() name node value (4, 9, 14 lines).
- child::empinfo 2 to 18 lines.
- /employee/descendant::*4, 5, 6, 9, 10, 11, 14, 15, 16 lines
- //descendant::employee (3 to 7, 8 to 12, 13 to 17 lines.)
- //employee/descendant-or-self::* (4, 5, 6, 8, 9, 10, 11, 13, 14, 15, 16 lines.)
- //descendant-or-self::employee (3 to 7, 8 to 12, 13 to 17 lines.)
- //employee/ancestor::* (2 to 18 lines.)
- //ancestor::name (name node value (4, 9, 14 lines)).
- //employee/ancestor-or-self::* (2, 3, 7, 8, 12, 13, 17, 18 lines (4 node select).)
- //name/ancestor-or-self::employee (3 to 7, 8 to 12, 13 to 17 lines (3 node select).)
- //name/parent::* (3 to 7, 8 to 12, 13 to 17 lines (3 node select).)
- //name/parent::employee (3 to 7, 8 to 12, 13 to 17 lines (3 node select).)
- //attribute::id (id attribute value (3, 8, 13 lines).)
- //attribute::* (3,5,8,9,10,13,15)
- //employee[@id=1]/following::* (8 to 12, 9, 10, 11, 13 to 17, 14, 15, 16 lines (8 node select).)
- //employee[@id=1]/following-sibling::* (8 to 12, 13 to 17 lines (2 node select).)
- //employee[@id=3]/preceding::* (3 to 7, 4, 5, 6, 8 to 12, 9, 10, 11 lines (8 node select).)
- //employee[@id=3]/preceding-sibling::* (3 to 7, 8 to 12 lines (2 node select).)

XPath Operators

An XPath expression can return a number, boolean (true or false), node-set(div,a,li), or string. You can use various operators to manipulate the values returned by an XPath expression.

Here are the various operators used in the XPath expression:

Operator	Description
|	Computes two node-sets.
+	Addition Operator
-	Subtraction Operator
*	Multiplication Operator
div	Division Operator
=	Equal Operator

!=	Not Equal Operator
<	Less than Operator
<=	Less than or Equal to Operator
>	Greater than Operator
>=	Greater than or Equal to Operator
or	Or Operator
and	And Operator
mod	Modulus (Division Remainder)

There are five different types of XPath Operators, listed below:

- Comparison operators
- Boolean operators
- Number operators or functions
- String functions
- Node operators or functions

Comparison Operators

Comparison operators compare two different values. Here are examples of various comparison operators used in an XPath expression:

Operator	Description
=	Specifies equals to
!=	Specifies not equals to
<	Specifies less than
>	Specifies greater than
<=	Specifies less than or equals to
>=	Specifies greater than or equals to

In the below example, we create a table of elements containing attribute id and child <firstname>,<lastname>, <nickname>, and <salary> by iterating over each employee. It checks the salary to be greater than (>) 25000 and then prints the details.

Employee.xml

```
<?xml version = "1.0"?>
<?xml-stylesheet type = "text/xsl" href = "employee.xsl"?>
<class>
```

```
   <employee id = "001">
      <firstname>Abhiram</firstname>
      <lastname>Kushwaha</lastname>
      <nickname>Manoj</nickname>
      <salary>15000</salary>
   </employee>
   <employee id = "002">
      <firstname>Akash</firstname>
      <lastname>Singh</lastname>
      <nickname>Bunty</nickname>
      <salary>25000</salary>
   </employee>
    <employee id = "003">
      <firstname>Brijesh</firstname>
      <lastname>Kaushik</lastname>
      <nickname>Ballu</nickname>
      <salary>20000</salary>
   </employee>
    <employee id = "004">
      <firstname>Zoya</firstname>
      <lastname>Mansoori</lastname>
      <nickname>Sonam</nickname>
      <salary>30000</salary>
   </employee>
</class>
```

Employee.xsl

```
<?xml version = "1.0" encoding = "UTF-8"?>
<xsl:stylesheet version = "1.0"
   xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">
   <xsl:template match = "/">
      <html>
         <body>
            <h2>Employee</h2>
            <table border = "1">
               <tr bgcolor = "pink">
                  <th>ID</th>
                  <th>First Name</th>
                  <th>Last Name</th>
                  <th>Nick Name</th>
                  <th>Salary</th>
               </tr>
```

```
            <xsl:for-each select = "class/employee">
               <xsl:if test = "salary > 25000">
                  <tr>
                     <td><xsl:value-of select = "@id"/></td>
                     <td><xsl:value-of select = "firstname"/></td>
                     <td><xsl:value-of select = "lastname"/></td>
                     <td><xsl:value-of select = "nickname"/></td>
                     <td><xsl:value-of select = "salary"/></td>
                  </tr>
               </xsl:if>
            </xsl:for-each>
         </table>
      </body>
   </html>
</xsl:template>
</xsl:stylesheet>
```

Boolean Operators

This type of operators return true or false as result. The following are different boolean operators in XPath:

Operator	Description
and	Specifies both conditions must be satisfied
or	Specifies any one of the conditions must be satisfied
not()	Specifies a function to check conditions not to be satisfied

In this example, we create a table of elements with its attribute id and child <firstname>,<lastname>, <nickname>, and <salary>.The example checks id to be either 001 or 003, then prints the details.

Employee.xml

```
<?xml version = "1.0"?>
<?xml-stylesheet type = "text/xsl" href = "employee.xsl"?>
<class>
   <employee id = "001">
      <firstname>Abhiram</firstname>
      <lastname>Kushwaha</lastname>
      <nickname>Manoj</nickname>
      <salary>15000</salary>
   </employee>
   <employee id = "002">
```

```
      <firstname>Akash</firstname>
      <lastname>Singh</lastname>
      <nickname>Bunty</nickname>
      <salary>25000</salary>
   </employee>
    <employee id = "003">
      <firstname>Brijesh</firstname>
      <lastname>Kaushik</lastname>
      <nickname>Ballu</nickname>
      <salary>20000</salary>
   </employee>
    <employee id = "004">
      <firstname>Zoya</firstname>
      <lastname>Mansoori</lastname>
      <nickname>Sonam</nickname>
      <salary>30000</salary>
   </employee>
</class>
```

Employee.xsl

```
<?xml version = "1.0" encoding = "UTF-8"?>
<xsl:stylesheet version = "1.0"
   xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">
   <xsl:template match = "/">
      <html>
         <body>
            <h2>Employee</h2>
            <table border = "1">
               <tr bgcolor = "pink">
                  <th>ID</th>
                  <th>First Name</th>
                  <th>Last Name</th>
                  <th>Nick Name</th>
                  <th>Salary</th>
               </tr>
               <xsl:for-each select = "class/employee[(@id = 001) or ((@id  =
003))]">
                  <tr>
                     <td><xsl:value-of select = "@id"/></td>
                     <td><xsl:value-of select = "firstname"/></td>
                     <td><xsl:value-of select = "lastname"/></td>
                     <td><xsl:value-of select = "nickname"/></td>
                     <td><xsl:value-of select = "salary"/></td>
```

```
                    </tr>
                </xsl:for-each>
            </table>
        </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Number Operators and Functions

The following describes various number operators you can use an in Xpath expression:

Operator	Description
+	Addition operation
-	Subtraction operation
*	Multiplication operation
div	Division operation
mod	Modulo operation

Here are some functions you can perform on XPath expressions:

Function	Description
ceiling()	Returns the smallest integer larger than the value provided
floor()	Returns the largest integer smaller than the value provided
round()	Returns the rounded value to the nearest integer
sum()	Returns the sum of two numbers

In this example, we create a table of <employee> element with its attribute id and child <firstname>,<lastname>, <nickname>, and <salary>. It calculates the salary of the employees, then displays the result.

Employee.xml

```
<?xml version = "1.0"?>
<?xml-stylesheet type = "text/xsl" href = "employee.xsl"?>
<class>
   <employee id = "001">
      <firstname>Abhiram</firstname>
      <lastname>Kushwaha</lastname>
      <nickname>Manoj</nickname>
      <salary>15000</salary>
```

```
   </employee>
   <employee id = "002">
      <firstname>Akash</firstname>
      <lastname>Singh</lastname>
      <nickname>Bunty</nickname>
      <salary>25000</salary>
   </employee>
    <employee id = "003">
      <firstname>Brijesh</firstname>
      <lastname>Kaushik</lastname>
      <nickname>Ballu</nickname>
      <salary>20000</salary>
   </employee>
    <employee id = "004">
      <firstname>Zoya</firstname>
      <lastname>Mansoori</lastname>
      <nickname>Sonam</nickname>
      <salary>30000</salary>
   </employee>
</class>
```

Employee.xsl

```
<?xml version = "1.0" encoding = "UTF-8"?>
<xsl:stylesheet version = "1.0"
   xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">
   <xsl:template match = "/">
      <html>
         <body>
            <h2>Employee</h2>
            <table border = "1">
               <tr bgcolor = "pink">
                  <th>ID</th>
                  <th>First Name</th>
                  <th>Last Name</th>
                  <th>Nick Name</th>
                  <th>Salary</th>
                  <th>Grade</th>
               </tr>
               <xsl:for-each select = "class/employee">
                  <tr>
                     <td><xsl:value-of select = "@id"/></td>
                     <td><xsl:value-of select = "firstname"/></td>
```

```
                  <td><xsl:value-of select = "lastname"/></td>
                  <td><xsl:value-of select = "nickname"/></td>
                  <td><xsl:value-of select = "salary"/></td>
                  <td>
                     <xsl:choose>
                        <xsl:when test = "salary div 25000 > 1">
                           High
                        </xsl:when>
                        <xsl:when test = "salary div 20000 > 1">
                           Medium
                        </xsl:when>
                        <xsl:otherwise>
                           Low
                        </xsl:otherwise>
                     </xsl:choose>
                  </td>
               </tr>
            </xsl:for-each>
         </table>
      </body>
   </html>
</xsl:template>
</xsl:stylesheet>
```

String Functions

Here are various string functions that help you carry out XPath expression tasks:

- **starts-with(string1, string2):** Returns true when the first string starts with the second string
- **contains(string1, string2):** Returns true when the first string contains the second string
- **substring(string, offset, length?):** You get a section of the string as a result. The section starts at offset up to the length provided.
- **substring-before(string1, string2):** This function returns the part of string1 up before the first occurrence of string2.
- **substring-after(string1, string2):** It returns the part of string1 after the first occurrence of string2.
- **string-length(string):** This function returns the length of string in terms of characters.
- **normalize-space(string):** You can trim the leading and trailing space from string with this function.
- **translate(string1, string2, string3):** It returns string1 after any matching characters in string2 have been replaced by the characters in string3.
- **concat(string1, string2, ...):** You can combine all the strings using this function.

- **format-number(number1, string1, string2):** It returns a formatted version of number1 after applying string1 as a format string. String2 is an optional locale string.

In this example, we create a table of <employee> elements with their names and length of names, by iterating over each employee. It calculates the length of the employee name after concatenating firstname and lastname, then displays the employee details.

Example.xml

```
<?xml version = "1.0"?>
<?xml-stylesheet type = "text/xsl" href = "employee.xsl"?>
<class>
   <employee id = "001">
      <firstname>Abhiram</firstname>
      <lastname>Kushwaha</lastname>
      <nickname>Manoj</nickname>
      <salary>15000</salary>
   </employee>
   <employee id = "002">
      <firstname>Akash</firstname>
      <lastname>Singh</lastname>
      <nickname>Bunty</nickname>
      <salary>25000</salary>
   </employee>
    <employee id = "003">
      <firstname>Brijesh</firstname>
      <lastname>Kaushik</lastname>
      <nickname>Ballu</nickname>
      <salary>20000</salary>
   </employee>
    <employee id = "004">
      <firstname>Zoya</firstname>
      <lastname>Mansoori</lastname>
      <nickname>Sonam</nickname>
      <salary>30000</salary>
   </employee>
</class>
```

Example.xsl

```
<?xml version = "1.0" encoding = "UTF-8"?>
<xsl:stylesheet version = "1.0"
   xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">
   <xsl:template match = "/">
```

```
      <html>
         <body>
            <h2>Employee</h2>
            <table border = "1">
               <tr bgcolor = "pink">
                  <th>Name</th>
                  <th>Length of Name</th>
               </tr>
               <xsl:for-each select = "class/employee">
                  <tr>
                     <td><xsl:value-of select = "concat(firstname,'
',lastname)"/></td>
                     <td><xsl:value-of select =
"string-length(concat(firstname,' ',lastname))"/></td>
                  </tr>
               </xsl:for-each>
            </table>
         </body>
      </html>
   </xsl:template>
</xsl:stylesheet>
```

Node Functions

The following table highlights various node operators, along with their descriptions:

Operator	Description
/	Select node under a specific node.
//	Select node from root node.
[...]	Check node value.
|	Unifies two node sets.

The following is the list of the node functions used in XPath expressions.

Function	Description
node()	Selects all kinds of nodes
processing-instruction()	Selects nodes processing instructions
text()	Selects a text node
name()	Provides node name

position()	Provides node position
last()	Selects the last node relative to current node
comment()	Selects nodes that are comments

In this example, we create a table of <employee> elements with their details by iterating over each employee. It calculates the position of the student node, then displays employee details with serial numbers.

Employee.xml:

```
<?xml version = "1.0"?>
<?xml-stylesheet type = "text/xsl" href = "employee.xsl"?>
<class>
   <employee id = "001">
      <firstname>Abhiram</firstname>
      <lastname>Kushwaha</lastname>
      <nickname>Manoj</nickname>
      <salary>15000</salary>
   </employee>
   <employee id = "002">
      <firstname>Akash</firstname>
      <lastname>Singh</lastname>
      <nickname>Bunty</nickname>
      <salary>25000</salary>
   </employee>
    <employee id = "003">
      <firstname>Brijesh</firstname>
      <lastname>Kaushik</lastname>
      <nickname>Ballu</nickname>
      <salary>20000</salary>
   </employee>
    <employee id = "004">
      <firstname>Zoya</firstname>
      <lastname>Mansoori</lastname>
      <nickname>Sonam</nickname>
      <salary>30000</salary>
   </employee>
</class>
```

Employee.xsl:

```
<?xml version = "1.0" encoding = "UTF-8"?>
```

```
<xsl:stylesheet version = "1.0"
   xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">
   <xsl:template match = "/">
      <html>
         <body>
            <h2>Employee</h2>
            <table border = "1">
               <tr bgcolor = "pink">
                  <th>Serial No</th>
                  <th>ID</th>
                  <th>First Name</th>
                  <th>Last Name</th>
                  <th>Nick Name</th>
                  <th>Salary</th>
               </tr>
               <xsl:for-each select = "class/employee">
                  <tr>
                     <td><xsl:value-of select = "position()"/></td>
                     <td><xsl:value-of select = "@id"/></td>
                     <td><xsl:value-of select = "firstname"/></td>
                     <td><xsl:value-of select = "lastname"/></td>
                     <td><xsl:value-of select = "nickname"/></td>
                     <td><xsl:value-of select = "salary"/></td>
                  </tr>
               </xsl:for-each>
            </table>
         </body>
      </html>
   </xsl:template>
</xsl:stylesheet>
```

XPath Selectors

With the help of the XPath selectors, you can select only a specific part of your HTML document specified by the XPath elements. XPath has many different types of selectos.

Descendent Selectors

Descendant selectors represent all the current node's children, and all children of each child, etc. These selectors do not include attribute and namespace nodes. The parent of an attribute node is its element node, whereas attribute nodes are not the offspring of their parents.

div	//div	Select all div elements

div h1	//div//h1	Select all h1 within a div element
ul > li	//ul/li	Select all li elements which are children of ul
div > p > a	/div/p/a	Select all anchor tags within paragraph tag of div element
div > *	//div/*	Select all elements in div tag
:root	/	Select root element of the DOM
:root > body	/body	Select body tag

Attribute Selectors

The XPath attribute selector matches elements based on the presence or value of a given attribute.

Element	Xpath	Description
#id	//*[@id=”id”]	Select all elements with matching ID attribute
.class	//*[@class=”class”]	Select all elements with matching class attribute
a[rel]	//a[@rel]	Select all anchor tag(s) with rel attribute.
a[href^=’/’]	//a[starts-with(@href, ‘/’)]	Select all anchor tag(s) with href starting with ‘/’
a[href$=’txt’]	//a[ends-with(@href, ‘.txt’)]	Select all anchor tag(s) with href ending with ‘.txt’
a[rel~=’details’]	//a[contains(@rel, ‘details’)]	Select all anchor tag(s) with rel value ‘details’
input[type=”password”]	//input[@type=”password”]	Select all input tag(s) of type password
a#btn[for=”XYZ”]	//a[@id=”btn”][@for=”XYZ”]	Select all anchor tag(s) with ‘btn’ ID linked with XYZ

Order Selectors

You can use order selectors in XPath to retrieve list elements.

Element	XPath	Description
ul > li:first-of-type	//ul/li[1]	Select first li tag that is a child of ul
ul > li:nth-of-type(2)	//ul/li[2]	Select second li tag that is a child of ul
li#id:first-of-type	//li[1][@id=”id”]	Select first li with id value “id”

ul > li:last-of-type	//ul/li[last()]	Select last li that is a child of ul
a:first-child	//*[1][name()=”a”]	Select first child of anchor tag
a:last-child	//*[last()][name()=”a”]	Select last child of anchor tag

Siblings

In Selenium WebDriver, you can retrieve a WebElement that is a sibling to a parent element. Here’s how to fetch elements using siblings in Selenium WebDriver:

Element	XPath	Description
h1 ~ ul	//h1/following-sibling::ul	Select all ul tags following sibling of h1 tag
h1 ~ #id	//h1/following-sibling::[@id=”id”]	Select all elements with ID value “id” that are siblings of h1
h1 + ul	//h1/following-sibling::ul[1]	Select first ul tags following sibling of h1

Different Operators

There are other operators in XPath to locate elements:

Operators	XPath	Description
Not Operators	//p[not(@id)]	Select all paragraph tag(s) with attributes not matching id
Text match	//button[text()=” Submit”]	Select button with text input “Submit”
Text match (substring)	//button[contains(text(),”pass”)]	Select button that has string ‘pass’ present in it
Arithmetic	//product[@price > 3]	Select price with value > 3
Has children	//ul[*]	Select ul with any number (or type) if it has children
Has children (specific)	//ul[li]	Select ul with any number and li tag present as a child
logic	//a[@name or @href]	Select all anchor tag(s) with the name and href attribute
Union (joins results)	//a | //div	Union of a and div tags

Contextual Selectors

Locators	Description
//img	image element
//img/*[1]	first child of element img
//ul/child::li	first child 'li' of 'ul'
//img[1]	first img child
//img/*[last()]	last child of element img
//img[last()]	last img child
//img[last()-1]	second last img child
//ul[*]	ul' that has children

Attribute Selectors

Locators	Description
//img[@id='myId']	image element with @id= 'myId'
//img[@id!='myId']	image elements with @id not equal to 'myId'
//img[@name]	image elements that have name attribute
//*[contains(@id, 'Id')]	element with @id containing
//*[starts-with(@id, 'Id')]	element with @id starting with
//*[ends-with(@id, 'Id')]	element with @id ending with
//*[matches(@id, 'r')]	element with @id matching regex ‘r’
//*[@name='myName']	image element with @name= 'myName'
//*[@id='X' or @name='X']	element with @id X or a name X
//*[@name="N"][@value="v"]	element with @name N & specified @value ‘v’
//*[@name="N" and @value="v"]	element with @name N & specified @value ‘v’
//*[@name="N" and not(@value="v")]	element with @name N & not specified @value ‘v’
//input[@type="submit"]	input of type submit

//a[@href="url"]	anchor with target link 'url'
//section[//h1[@id='hi']]	returns <section> if it has an <h1> descendant with @id= 'hi'
//*[@id="TestTable"]//tr[3]//td[2]	cell by row and column
//input[@checked]	checkbox (or radio button) that is checked
//a[@disabled]	all 'a' elements that are disabled
//a[@price > 2.50]	a' with price > 2.5

XPath Methods

Here are some of the various XPath methods:

Locator	Explanation
//table[count(tr) > 1]	Return table with more than 1 row
//*[.="t"]	Element containing text 't' exactly
//a[contains(text(), "Log Out")]	Anchor with inner text containing 'Log Out'
//a[not(contains(text(), "Log Out"))]	Anchor with inner text not containing 'Log Out'
//a[not(@disabled)]	All 'a' elements that are not disabled

Math Methods

Locator	Explanation
ceiling(number)	Evaluates a decimal number and returns the **smallest** integer greater than or equal to the decimal number
floor(number)	Evaluates a decimal number and returns the **largest** integer less than or equal to the decimal number
round(decimal)	Returns the nearest integer to the given number
sum(node-set)	Returns the sum of the numeric values of each node in a given node-set

String Methods

Locator	Explanation
contains(space-string, planet-string)	Determines whether the first argument string contains the second argument string and returns boolean true or false
concat(string1, string2 [stringn]*)	Concatenates two or more strings and returns the resulting string
normalize-space(string)	Strips leading and trailing white-space from a string; replaces sequences of whitespace characters by a single space; returns the resulting string
starts-with(spacetrack, space)	Checks whether the first string starts with the second string and returns true or false
string-length([string])	Returns a number equal to the number of characters in a given string
substring(string, start [length])	Returns part of a given string
substring-after(spacetrack, track)	Returns the rest of a given string after a given substring
substring-before(spacetrack, tra)	Returns the rest of a given string before a given substring
translate(string, ghj, GHJ)	Evaluates a string and a set of characters to translate and returns the translated string

Obtaining XPath using JQuery

JQuery supports all basic kinds of XPath expressions. The major ones are listed below in this next section of our XPath cheat sheet!

JQuery XPath	XPath	Description
$(‘ul > li’).parent()	//ul/li/..	Selects all ul elements that are parent of li tag
$(‘li’).closest(‘section’)	//li/ancestor-or-self::section	Selects all anchor tags with href attribute
$(‘p’).text()	//p/text()	Selects text within paragraph tags