XPATH by attribute

- In XPATH by attribute we use attributes to identify the element uniquely.
- In order to identify the elements using attribute we use a special symbol “@”.
- “@” -> this symbol is used to select the attributes.

Syntax:

```
//tagName[@attribute = attribute's value]
```

// - points to the any node in the webpage

TagName - tag name is nothing but the name which is present after the < (angular bracket)

@: It is used to select attribute.

attribute - whatever is present inside < and > bracket except tag name is attribute, any number of attributes can present in html code

attribute's value - it is corresponding value to the attribute.

Example:

```
<input type="text" name="username" id="username" class="textField" placeholder="Username">
```

```
//input[@name='username']
```

Example program to login to actiTIME using XPATH attribute expression.

```
from selenium import webdriver

driver = webdriver.Firefox()
driver.maximize_window()
driver.get('http://demo.actitime.com/')
driver.implicitly_wait(30)

userName = driver.find_element_by_xpath("//input[@id='username']")
userName.send_keys('admin')
password = driver.find_element_by_xpath("//input[@class='textField pwdfield']")
password.send_keys('manager')
loginButton = driver.find_element_by_xpath("//a[@id='loginButton']")
loginButton.click()

driver.close()
```

Note:

- We can use more than one attribute or property name to identify the web element.

Example:

```
<input type="text" name="username" id="username" class="textField" placeholder="Username">

//input[@name='username'][@class='textField']
```

Logical operations in XPATH expression

Anding:

- Consider a scenario, where we need to identify the elements only if multiple attributes are present for the element, then we go for performing the anding operation in the XPATH expression to identify the elements by multiple attribute values.
- Below expression will identify the element only if both the property values are present, if one property is present and another property value is not present, it will through the no such element exception.

Syntax:

```
//TagName[@AttributeName='AttributeValue' and @AttributeName='AttributeValue']
```

Example:

```
<input type="text" data-validate="isEmail" id="email_create" name="email_create" value="">

//input[@id='email_create' and @data-validate='isEmail']
```

```
from selenium import webdriver

driver = webdriver.Firefox()
driver.maximize_window()
driver.get('http://demo.actitime.com/')
driver.implicitly_wait(30)

userName = driver.find_element_by_xpath("//input[@type='text' and @id='username']")
userName.send_keys('admin')
password = driver.find_element_by_xpath("//input[@type='password' and @name='pwd']")
password.send_keys('manager')
loginButton = driver.find_element_by_xpath("//a[@id='loginButton']")
loginButton.click()

driver.close()
```

ORING:

- Consider a scenario, where we need to identify the elements if any one of the property value is present in that element. In those cases we develop the XPATH expression using ORING.
- Oring XPATH expression will identify the element, if any one property is present in the DOM.

Syntax:

```
//TagName[@AttributeName='AttributeValue' or @AttributeName='AttributeValue']
```

Example:

```
<input type="text" data-validate="isEmail" id="email_create" name="email_create" value="">
```

```
//input[@id='email_create' or @data-validate='isEmail']
```

```
from selenium import webdriver

driver = webdriver.Firefox()
driver.maximize_window()
driver.get('http://demo.actitime.com/')
driver.implicitly_wait(30)

userName = driver.find_element_by_xpath("//input[@type='text' or @id='username']")
userName.send_keys('admin')
password = driver.find_element_by_xpath("//input[@type='password' or @name='pwd']")
password.send_keys('manager')
loginButton = driver.find_element_by_xpath("//a[@id='loginButton']")
loginButton.click()

driver.close()
```

Not in XPATH

- Not function in XPATH, identifies the element if the specified attributes are not available.

Syntax:

```
//TagName [not(source='value')]
```

Here, Source is either, text of an element or attribute.

Example:

```
//a[not(text()='xpath')]
```

- The above expression identifies all the element, which doesn't have a text value "xpath"

```
//a[not(@id)]
```

- The above expression identifies all the element, which doesn't have a "@id".

XPATH by text function

- In order to identify the element by visible text we use a function "text ()" in XPATH to locate the element.

Syntax:

```
//TagName[text()='Visible text/text of an element']
```

Example 1:

```
from selenium import webdriver

driver = webdriver.Firefox()
driver.maximize_window()
driver.get('http://demo.actitime.com/')
driver.implicitly_wait(30)

keepMeLoggedIn = driver.find_element_by_xpath("//label[text()='Keep me logged in']")
keepMeLoggedIn.click()

driver.close()
```

Example 2:

```
from selenium import webdriver

driver = webdriver.Firefox()
driver.maximize_window()
driver.get('https://accounts.google.com/')
driver.implicitly_wait(30)

createAccount = driver.find_element_by_xpath("//span[text()='Create account']")
createAccount.click()

driver.close()
```

- Text function is used to navigate to entire HTML document and check for the expected value in UI, this return true if complete string matches with UI or else it returns no such element exception.
- Text function is used to identify any web element using the visible text in the UI.

- Text function is exact string matching function, which can't identify object part of a string.

Drawback back of Text function:

- Text function cannot identify the element, if that text contains the spaces before or after the text.
- It can't identify web element using the part of a string. We need to give the complete string to identify the element.
- Text function can't be used with back end attributes.

Note: In order to overcome from the drawback of text function. We use text function with another function.

Below are the two function, with which we use text function to identify the element using the visible text.

XPATH by normalize-space

The normalize-space function is used to normalize a string i.e. to remove any leading or trailing spaces from the string passed as parameter to the XPATH function.

Syntax:

```
//htmlTag[normalize-space(source)='value']

here,
    source may be a text() function or @attributeName
```

Example using text function:

```
//label[normalize-space(text())='Keep me logged in']
```

Example using the attribute:

```
//input[normalize-space(@name)='username']
```

Will look into few examples to see how the normalize space function works.

Example:

```
<html>
    <head>
        <title>Qspiders</title>
    </head>
    <body>
        <form>
            <input type="text" id="username">
            <input type="text" id="password">
            <br><br>
            <button type="button" name="pwd" value="Login">   Login   </button>
            <br><br>
            <a href="http://qspiders.com/">  Forgot Password?  </a>
        </form>
    </body>
</html>
```

- In the above HTML source, if we try to identify the login and forgot password element using the text function. We won't be able to identify the element.
- Coz, both the element has a spaces before and after the text.
- So to identify the element, even if the string contains the trailing spaces, we use normalize-space function.

Example program for text function in normalize-space:

```
from selenium import webdriver

driver = webdriver.Firefox()
driver.maximize_window()
driver.get('file:///C:/Users/PriyaPramod/Desktop/HTML%20Pages/XPath/ID.html')
driver.implicitly_wait(30)

loginButton = driver.find_element_by_xpath("//button[normalize-space(text())='Login']")
loginButton.click()

driver.close()
```

Example program for attribute in normalize-space:

```
from selenium import webdriver

driver = webdriver.Firefox()
driver.maximize_window()
driver.get('file:///C:/Users/PriyaPramod/Desktop/HTML%20Pages/XPath/ID.html')
driver.implicitly_wait(30)

loginButton = driver.find_element_by_xpath("//input[normalize-space(@id)='username']")
loginButton.click()

driver.close()
```

XPATH by Contains

By using 'contains' function in XPath, we can extract all the elements which matches a particular text value and attribute value.

Syntax:

```
//HtmlTag[contains(source,'value')]

Here,
    as a source, we can use text() function or @Attribute
```

Example for using the text () function in Contains

```
//label[contains(text(),'Keep me logged in')]
```

Example for using the @Attribute in contains

```
//input[contains(@id,'username')]
```

Example:

```
from selenium import webdriver

driver = webdriver.Firefox()
driver.maximize_window()
driver.get('https://demo.actitime.com/')
driver.implicitly_wait(30)

userName = driver.find_element_by_xpath("//input[contains(@id,'username')]")
userName.send_keys('admin')
password = driver.find_element_by_xpath("//input[contains(@name,'pwd')]")
password.send_keys('manager')
loginButton = driver.find_element_by_xpath("//div[contains(text(),'Login')]")
loginButton.click()

driver.close()
```

Note:

Situations where contains function should be used:

- When the values of the attributes are dynamic i.e. changing.
- When we would like to create a list of web elements contains same partial attribute value.

- Contains function is a sub string matching function, which can verify web element using part of the string.
- Contains function automatically ignore white spaces before and after the string.

XPATH by Starts-with

Starts-with function is used when we know about the initial partial attribute value or initial partial text associated with the web element.

User can also use this function to locate web elements those are consists of both the static (initial) and dynamic (trailing) values.

Syntax:

```
//htmlTag[starts-with(source, 'value')]

Here,
     In source, we can use text() function or @attributes
```

Example to use text function in starts-with:

```
//div[starts-with(text(),'Log')]
```

Example to use attribute in starts-with:

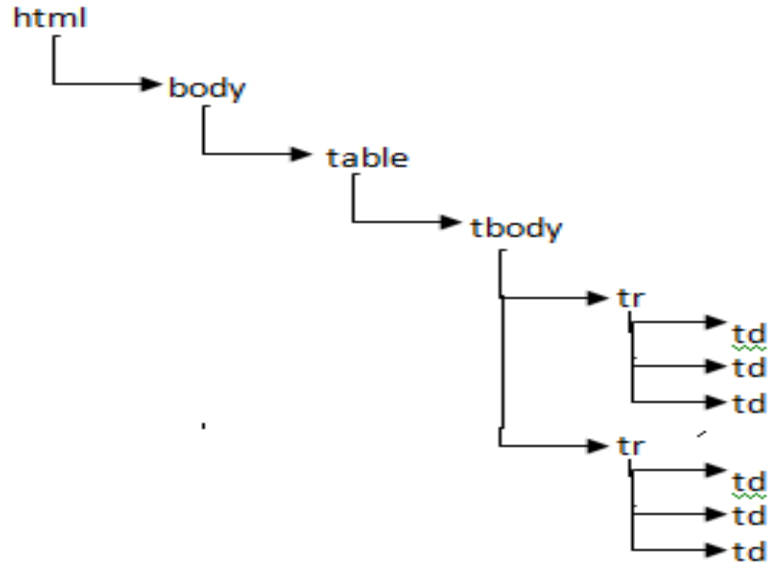
```
//input[starts-with(@id,'user')]
```

```
from selenium import webdriver

driver = webdriver.Firefox()
driver.maximize_window()
driver.get('https://demo.actitime.com/')
driver.implicitly_wait(30)

userName = driver.find_element_by_xpath("//input[starts-with(@id,'username')]")
userName.send_keys('admin')
password = driver.find_element_by_xpath("//input[starts-with(@name,'pwd')]")
password.send_keys('manager')
loginButton = driver.find_element_by_xpath("//div[starts-with(text(),'Login')]")
loginButton.click()

driver.close()
```

XPATH by traversing

We can write 'XPATH' expression which navigates from one element to another element which is called as traversing. In 'XPATH' there are 2 types of traversing.

- Forward Traversing
- Backward Traversing.



Forward Traversing

- Navigating from parent element to any of its child element is called as forward traversing.

Example:

1. Navigating from table node to java cell

```
//table/tbody/tr[1]/td[2]
```

2. Navigating from table to Unix cell

```
//table/tbody/tr[2]/td[2]
```

Backward Traversing

- Navigating from child element to any of its parent element is called as backward traversing.

Example:

There are two ways to travel in a backward direction.

- **Enclosing the child expression with in []**
- **Using the "/.."**

Navigating from java cell to table node

```
//table[tbody[tr[td[text()='Java']]]]

//td[text()='Java']/../../..
```

Independent and dependent Xpath

- If the element is completely dynamic or if the element is duplicate, then we can identify that element using some other element by applying a technique called independent, dependent XPATH.
- The element which is identified with respect to some other element is called as dependent element.

Let's take the below example:

1	Java	200
2	Unix	300

'XPATH' to identify cost of Java:

```
//td[text()='300']
```

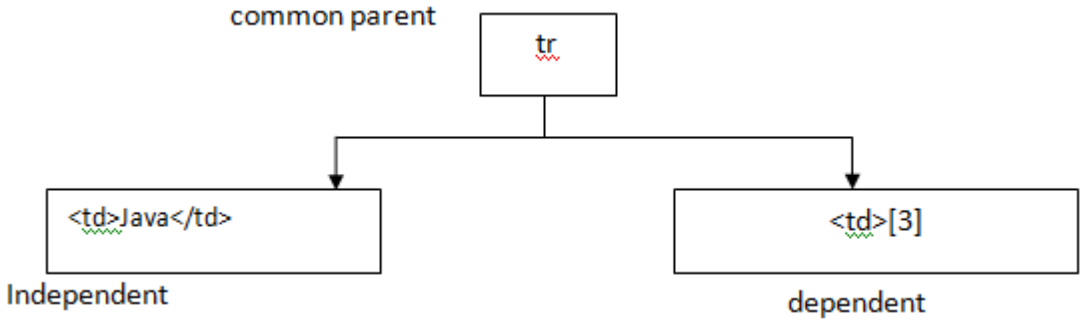
- In the above 'XPATH' expression we are identifying the cost directly.
- If the cost of the Java book changes then this 'XPATH' will not identify the element.

```
//td[text()='Java']/../td[3]
```

- In the 2nd 'XPATH' expression we are identifying the cost using the name of the subject.
- In this example Java is called as independent element and cost is called as dependent element. This will identify the cost even if it completely changes.

Steps to derive independent and dependent XPATH

1. First identify dependent and independent Element.
2. Inspect the independent element and note the source code.
3. Find the common parent.
   To find the common parent: Place the mouse pointer on source code of independent element and move the mouse pointer on upward direction step by step till it highlights both independent and dependent element. It will be the common parent. Add it to HTML tree.
4. Navigates from common parent to dependent element using arrow key and add it to HTML tree as shown below.

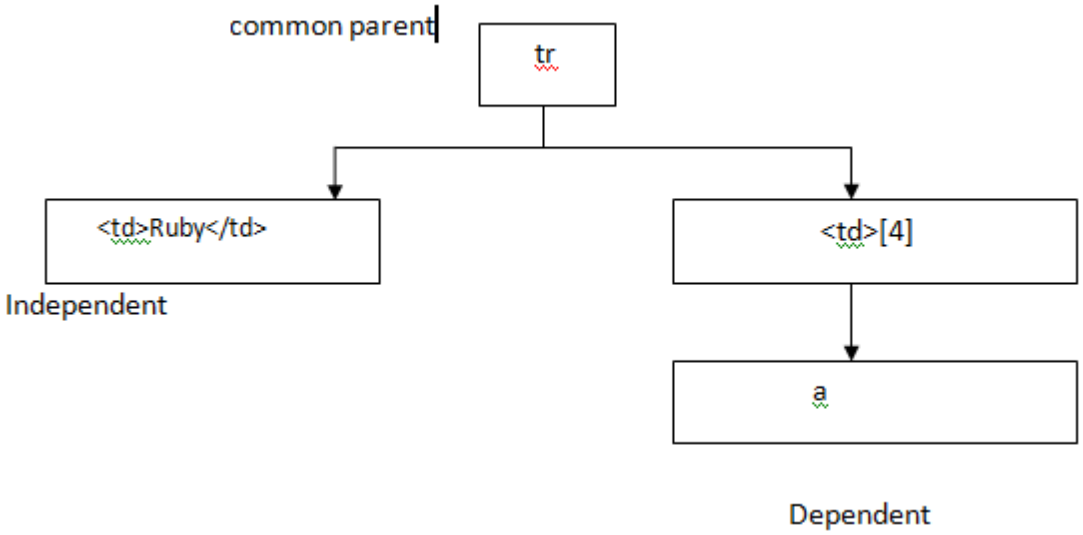


5. Derive the 'XPATH' which navigates from independent element to common parent and then to dependent element.
   - ➔ First derive the xpath expression for independent element
   - ➔ Second derive the xpath expression from independent element to common parent
   - ➔ Lastly derive the xpath expression from common parent to dependent element.

Example:

```
//td[text()='Java']/../td[3]
```

Derive an 'XPATH' which identifies download link of ruby which is present in Selenium download page?




```
Xpath: //td[text()='Ruby']/../td[4]/a
```

- The above 'XPATH' expression identifies the download link only if it is present in 4th column. If the column keeps changing we can use the below 'XPATH'.

```
Xpath: /td[text()='Ruby']/..//a[text()='Download']
```

Wild card character with XPATH

"*"

- Is the one of most used wild card character with XPATH in selenium webdriver, we can use it instead of tag name and attribute

//*

- Matches all the elements present in the html (including html)

//div/*

- Matches all the immediate element(s) inside the **div** tag

//input[@*]

- Matches all the element(s) with **input** tag and have at least one attribute, attribute value may or may not present

//*[@*]

- Matches all the element(s) which have at least one attribute.

How to write XPath if I have 'apostrophe' in my XPath element?

Let's take the below example,

/ Women's fashion /

- In the above example, the visible text contains the special character called apostrophe.
- In this case the only reliable way of using XPath in Selenium WebDriver for text with apostrophes (single quotes) is to use double quotes for the expression of the XPath.

```
//*[@attribute/text()=\"text's\"]
```

Example:

```
//span[text()=\"Women's fashion\"]
```

Example:

```
from selenium import webdriver
from time import sleep

driver = webdriver.Firefox()
driver.maximize_window()
driver.get('https://www.craftsvilla.com/cvfeeds/craftsvilla-brands')

woman_fashion_link = driver.find_element_by_xpath("//span[text()=\"Women's fashion\"]")
woman_fashion_link.click()
sleep(4)

driver.close()
```

XPATH by Axes functions

- Xpath axes search different nodes in XML document from current node.
- Xpath axes are the methods used to find dynamic elements.
- Axes methods are used to find those elements, which dynamically change on refresh or any other operations.
- Xpath axes are classified into two types
  1. Forward axes functions
     - Following
     - Child
     - Descendant
     - Following sibling
  2. Backward axes functions
     - Preceding
     - Ancestor
     - Parent
     - Preceding sibling

HTML Code:

```
<html><head>
    <title>Qspiders</title>
</head>
  <body style="font-family:arial">
    <ul id="parent">
      <li id="One"><a href="" class="listLink"><span class="position">1</span>One</a></li>
      <li id="Two"><a href="" class="listLink"><span class="position">2</span>Two</a></li>
      <li id="Three"><a href="" class="listLink"><span class="position">3</span>Three</a>
        <ul>
            <li id="A"><a href="" class="listLink"><span class="position">1</span>A</a></li>
            <li id="B"><a href="" class="listLink"><span class="position">2</span>B</a></li>
            <li id="C"><a href="" class="listLink"><span class="position">3</span>C</a></li>
            <li id="D"><a href="" class="listLink"><span class="position">4</span>D</a></li>
            <li id="E"><a href="" class="listLink"><span class="position">5</span>E</a></li>
            <li id="F"><a href="" class="listLink"><span class="position">6</span>F</a></li>
        </ul>
    </li>
    <li id="Four"><a href="" class="listLink"><span class="position">4</span>Four</a></li>
    <li id="Five"><a href="" class="listLink"><span class="position">5</span>Five</a></li>
    <li id="Six"><a href="" class="listLink"><span class="position">6</span>Six</a></li>
    </ul>
    </body>
</html>
```

Following

- This function selects all the following elements in the document of the current node (element).

Syntax:

```
//Current-element-expression//following::TagName
```

Example:

```
//a[contains(text(),'One')]//following::a
```

Preceding

- Selects all the elements that come before the current element.

Syntax:

```
//Current_Element_Expression//preceding::Tag_Name
```

Example:

```
//a[contains(text(),'Six')]//preceding::a
```

Ancestor

- The ancestor function is used to identify all the ancestor elements (Parent, Grand Parent, and Great Grand Parent) of the current element.

Syntax:

```
//Current_Element_Expression//ancestor::Tag_Name
```

Example:

```
//a[contains(text(),'One')]//ancestor::ul
```

Parent

- Selects the parent of the current element.

Syntax:

```
//Current_Element_Expression//parent::Tag_Na
```

Example:

```
//a[contains(text(),'One')]//parent::li
```

Child

- Selects all the child (Direct) elements of the current element.

Syntax:

```
//Current_Element_Expression//child::Tag_Name
```

Example:

```
//li[@id='One']//child::a
```

Descendant

- Selects all the child's (Descendants) of the current element.

Syntax:

```
//Current_Element_Expression//child::Tag_Name
```

Example:

```
//li[@id='Three']//descendant::a
```

XPATH by following-sibling

- The following-sibling axis function used to identify the following sibling of the context node.

Syntax:

```
//XpathExpression/following-sibling::SiblingTagNameame
```

Example:

```
//input[@id='username']/following-sibling::input
```

XPATH by preceding-sibling

- The preceding-sibling axis function used to identify the preceding sibling of the context node.

Syntax:

```
//XpathExpression/preceding-sibling::tagName
```

Example:

```
//input[@id='password']/preceding-sibling::input
```

XPATH by group index

- Sometimes we may have to handle the elements with XPATH index but index may give more than one match, which are under different parents, in these situations index might not help you. We may have to use Group index in these kind of scenarios.

Let's consider the below HTML code.

```
<html>
    <body>
        <div id="fruit"><br><br><br>
            <button type="button">Blueberry</button><br><br>
            <button type="button"  >Banana</button><br><br>
            <button type="button">Strawberry</button><br><br>
        </div>
        <div id="fruit">
            <button type="button">Apple</button><br><br>
            <button type="button" >Orange</button><br><br>
            <button type="button">Grape</button><br><br>
        </div>
    </body>
</html>
```

Now, write an XPATH for identifying the first element which is present in the first div using the Indexing logic.

Below is the XPATH.

```
//div[@id='fruit']/button[1]
```

It will find the two matching nodes.

It identifies the duplicate element. Coz, XPATH by indexing will find the elements which are under different parents. In this kind of scenarios, we go for using the XPATH by group indexing.

"Group index puts all matches into a list and gives indexes them. So here we will not have any duplicates matches"

```
Syntax : (//XPathExpression)[index]
```

We have to use parenthesis to make a XPATH into group XPATH after it index the XPATH.

Now, for the same above scenario, write the XPATH using the group indexing.

```
(//div[@id='fruit']/button)[1]
```

Above XPATH will identify only one XPATH.

Important:

1. What is the difference between //a, //a[1] and (//a) [1]?

- //a – Matches with all the links present in the entire web page
- //a[1]- Matches with all the first link
- (//a)[1] – Matches with only the first link

- //input[@type='checkbox'] – Matches with all the checkbox in the entire web page
- (//input[@type='checkbox'])[1]- Matches with the first checkbox
- (//input[@type='checkbox'])[5] - Matches with the fifth checkbox

In XPATH by group index, we use a function called "last ()"to select the last element.

Below is the syntax to select the last node in XPATH by group index.

```
Syntax : (//XPathExpression)[last()]
```

Derive the 'XPATH' which matches with first and last image?

```
(//img)[1]|(//img)[last()]
```