**Xpath:**

It is the path of the element in the html tree. In order to identify the elements, we can use xpath expression where we specify the tag name after the “/” or “//”

We have **2-types** of xpaths.

1. **Absolute Xpath:**

Specifying complete path of the element is called as absolute xpath.

1. **Relative Xpath:**

Writing absolute xpath on real-time application will be very difficult because length of expression will be more.

In order to reduce the length of the xpath we skip the elements wherever it is possible & navigate to the destination element. For this we use ‘**Descendant’** which is written as **‘//**’.

In relative xpath we use ‘//’ (double slash) which represents ‘Descendant’ element.

**🡪** We can join multiple xpath expression using ‘**|**’ (**pipe**)

**Html example:**

|  |
| --- |
| <html>  <body>  <div>  <input type="text" value="A">  <input type="text" value="B">  </div>  <div>  <input type="text" value="C">  <input type="text" value="D">  </div>  </body>  </html> |
| Web Page:   |  |  | | --- | --- | | A | B | | C | D | |

**Xpath examples:**

|  |  |  |
| --- | --- | --- |
| **Absolute Xpath** | **Relative Xpath** | **Matching Elements** |
| /html/body/div/input | //input | A, B, C, D |
| /html/body/div[1]/input | //div[1]/input | A, B |
| /html/body/div[2]/input | //div[2]/input | C, D |
| /html/body/div[1]/input[1] | //div[1]/input[1] | A |
| /html/body/div[1]/input[2] | //div[1]/input[2] | B |
| /html/body/div[2]/input[1] | //div[2]/input[1] | C |
| /html/body/div[2]/input[2] | //div[2]/input[2] | D |
| /html/body/div/input[1] | //input[1] | A, C |
| /html/body/div/input[2] | //input[2] | B, D |

Even if we use xpath by relative position, we should have good knowledge of html tree. Instead of this we can use **xpath by attribute**.

1. **Xpath by attribute:**

While writing xpath expression if we use index it will not identify the element of the location if the element changes.

Hence, whenever is possible avoid using index, instead of this use attribute.

|  |
| --- |
| **Syntax**:- //***htmlTag[@attribute=’value’]***  **Ex**:- //input[@value=’A’] |

We can use multiple attributes in single xpath expression. For this we use ‘**and’** & ‘**or’**.

|  |
| --- |
| **Syntax**:- //***htmlTag [@attribute=’value’ and @attribute=’value’]***  **Ex:** //input[@value=’A’ and @type=’text’] |

1. **Xpath By text() function:**

If the attributes are not present for an element or if attribute is matching with multiple elements then we can identify the element using its text.

|  |
| --- |
| **Syntax**:- ***//htmlTag [text()=’value’]***  Example html 🡪 <div>Login </div>  Example xpath 🡪 //div[text()=’Login ’] |

**Q) What is the difference between ‘text’ & ‘text()’?**

🡪 ‘text’ represents value of an attribute whereas ‘text()’ represents text of an element.

**Ex**:- //input[@type=’text’] 🡪 <input type=”text”>

//input[text()=’Vijay’] 🡪 <input>Vijay</input>

1. **Xpath by contains function:**

If an element’s attribute value or text value is partially changing then we use contains function.

|  |
| --- |
| **Syntax:**   1. ***//HtmlTag[contains(@attribute, ‘value’)]*** 2. ***//HtmlTag[contains(text(), ‘value’)]***   Examples:   1. //img[contains(@src, ‘timer’)] 2. //span[contains(text(), ‘actiTime’)] |

**Note**:- Using ‘contains’ function we cannot handle the element if its value changes completely.

**Advantages of contains () function:**

1.We can identify the element when the value has additional spaces

2.we can identify the element when the value is very lengthy

3.we can identify the element when the value(partially) is keep on changing (ex: inbox (91))