

Selenium WebDriver

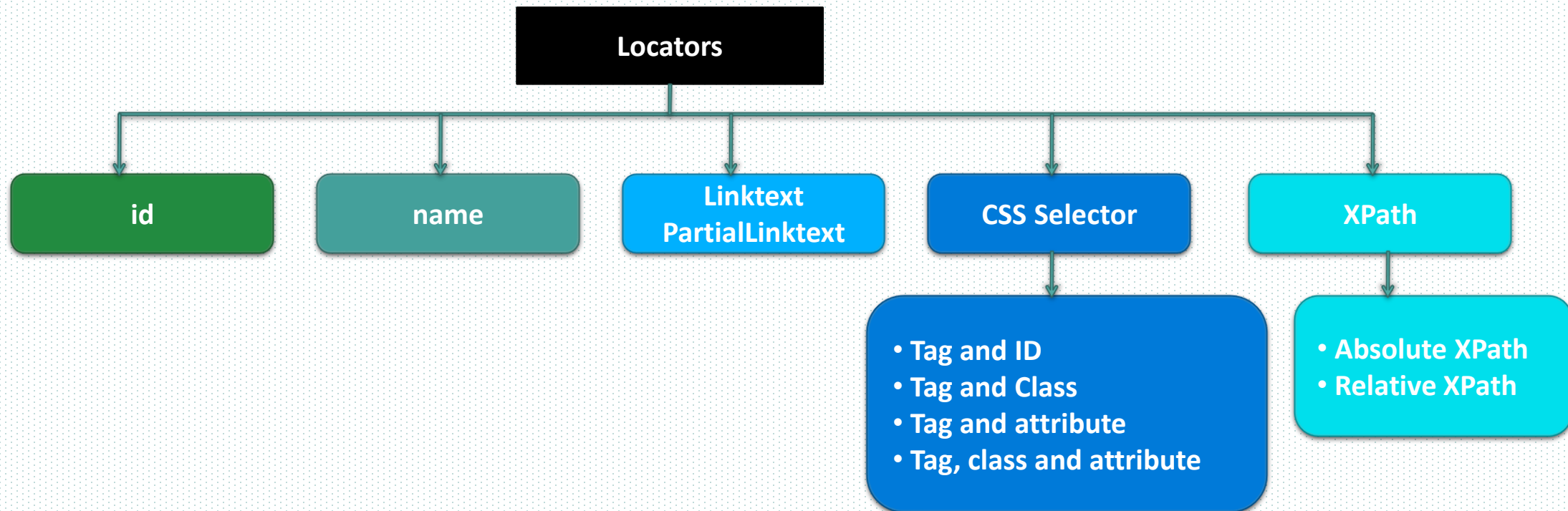
Agenda



- Locators in WebDriver

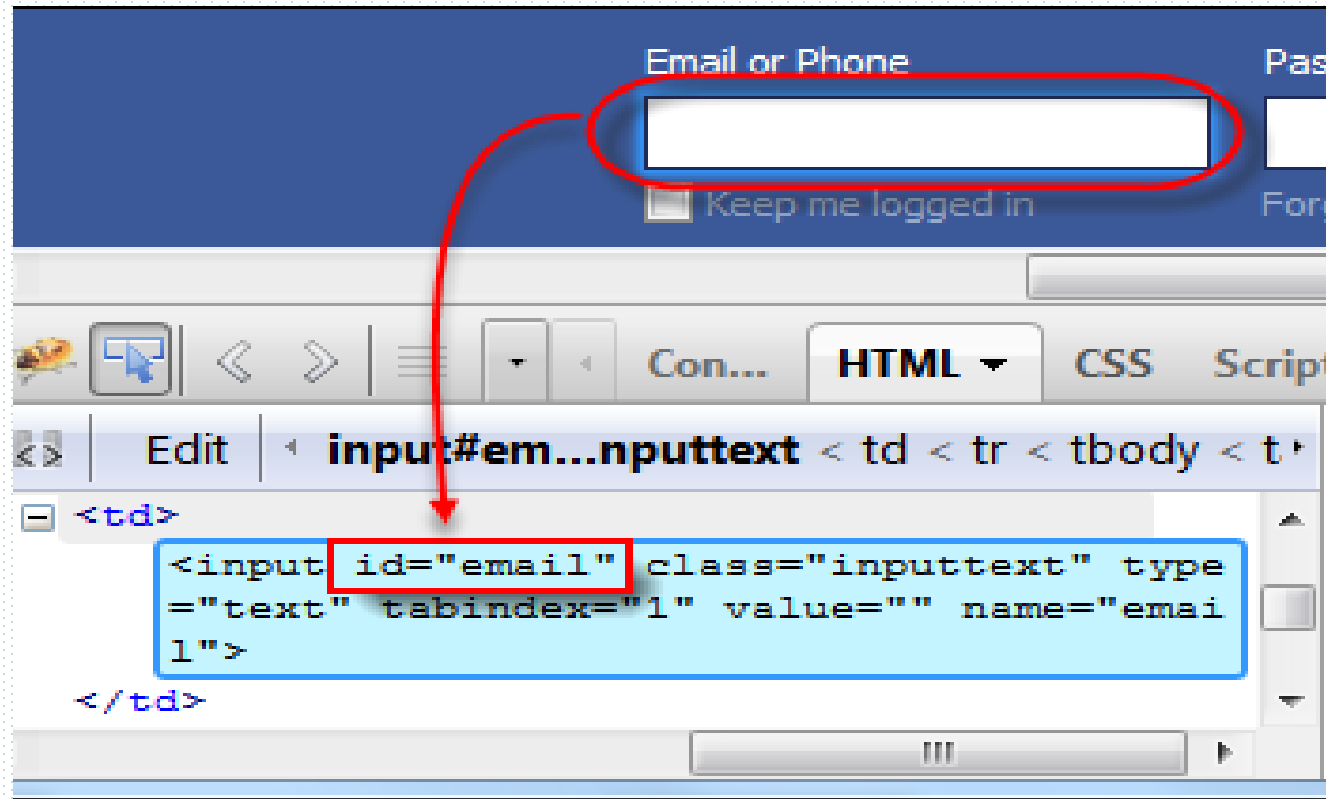
Locators

- We can identify various elements on the web using **Locators**.
- Locators are addresses that identify a web element uniquely within the page.



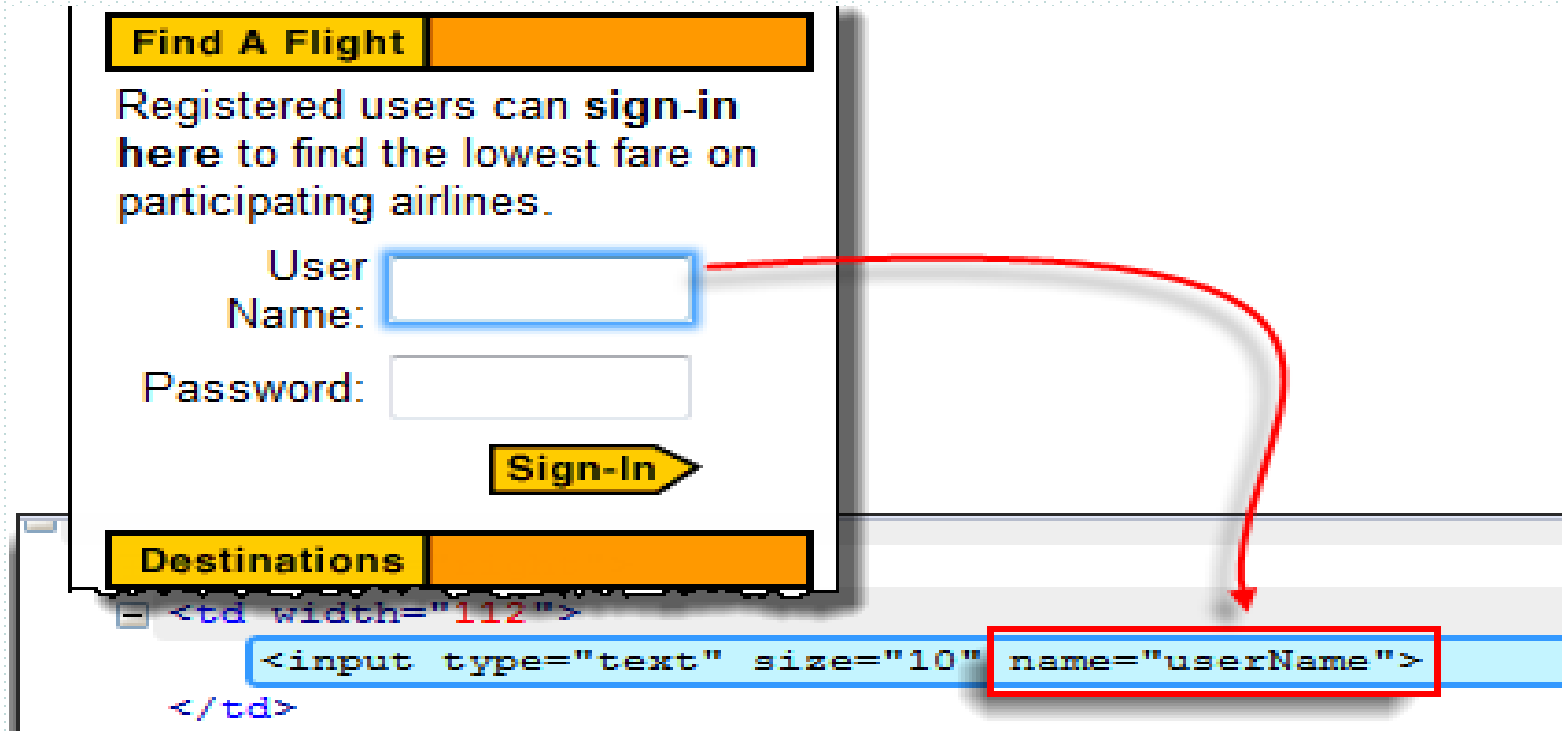
Locating by ID

```
driver.findElement(By.id("email")).sendKeys("xxxxx@gmail.com");
```



Locating by the name

```
driver.findElement(By.name("userName")).sendKeys("mercury");
```



The screenshot shows a web form titled "Find A Flight" with a sub-header "Destinations". Below the header, there is a text input field labeled "User Name:" and a "Sign-In" button. A red arrow originates from the "User Name:" input field and points to the Selenium code snippet below. The code snippet is a snippet of HTML code showing an input field with the attribute `name="userName"` highlighted in a red box.

```
<td width="112">  
  <input type="text" size="10" name="userName">  
</td>
```

Locating by the Link Text

```
driver.findElement(By.linkText("REGISTER")).click();
```



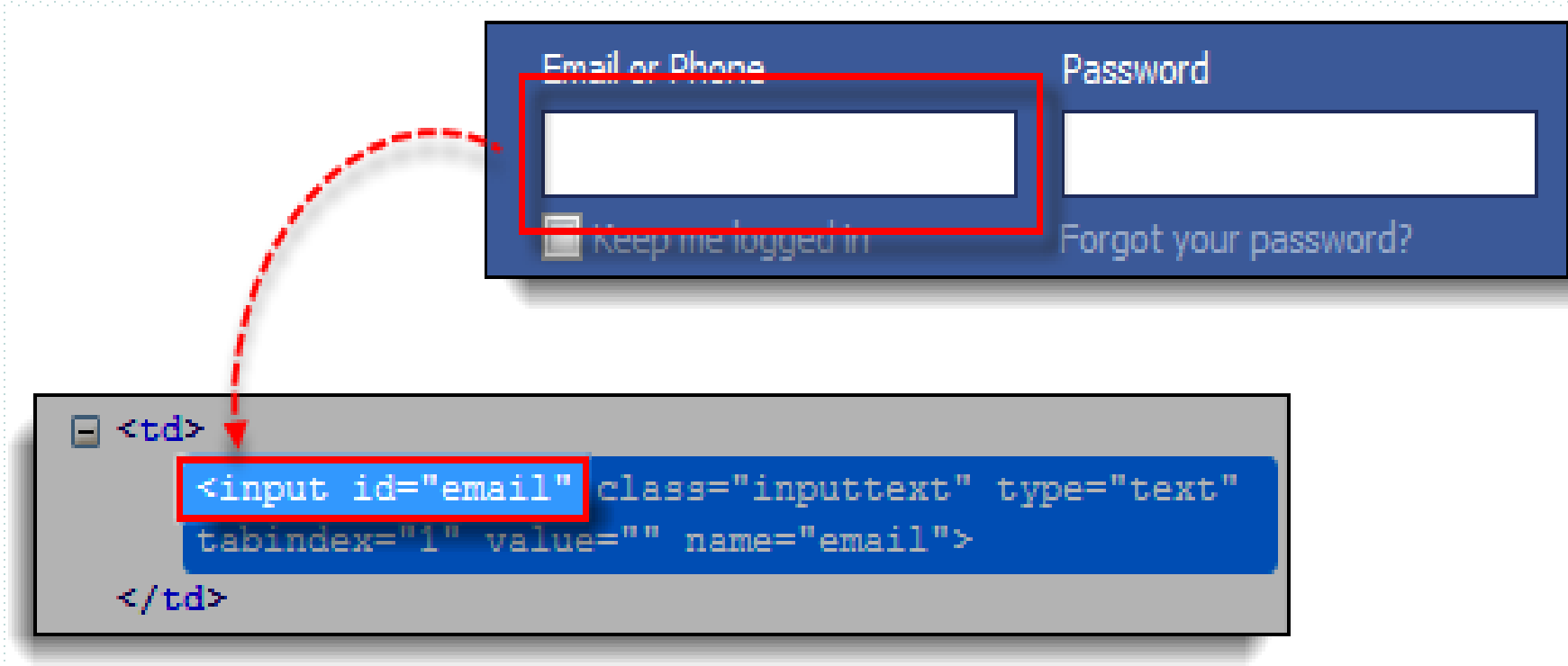
```
'mouseover') " onmouseout="changeStyle(th  
is, 'mouseout') ">  
  <a href="mercuryregister.php">REGISTER<  
  /a>  
</td>  
+ <td class="mouseout" width="73" height="
```

CSS Selector - Cascading Style Sheets

- Tag and ID
- Tag and Class
- Tag and attribute
- Tag, class and attribute

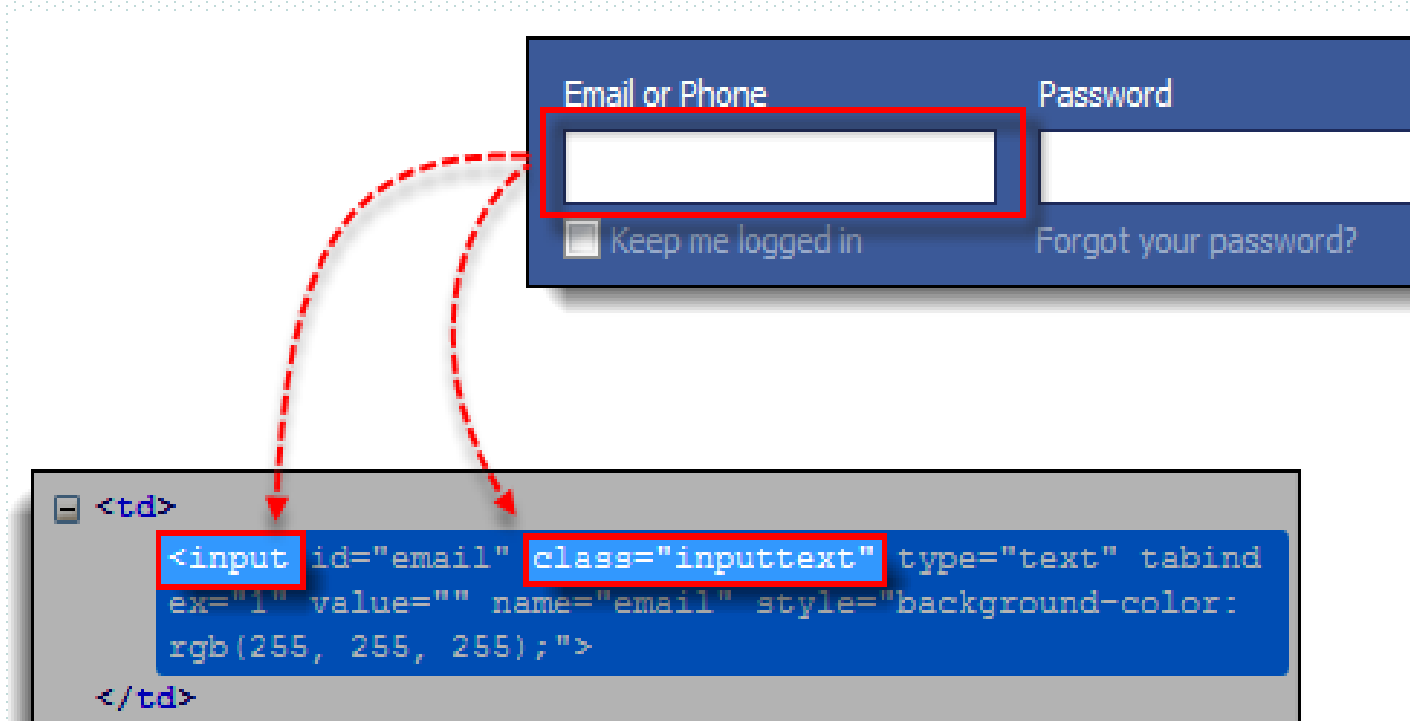
CSS Selector – *Tag* and *ID*

```
driver.findElement(By.cssSelector("input#email")).sendKeys("xxxxxxx@gmail.com");
```



CSS Selector – *Tag* and *Class*

```
driver.findElement(By.cssSelector("input.inputtext")).sendKeys("xxxxxx@gmail.com");
```



Multiple elements have the same tag and class name

- when multiple elements have the same HTML tag and name, only the first element in source code will be recognized.

```
driver.findElement(By.cssSelector("input.inputtext")).sendKeys("xxxxxxxx@gmail.com");
```

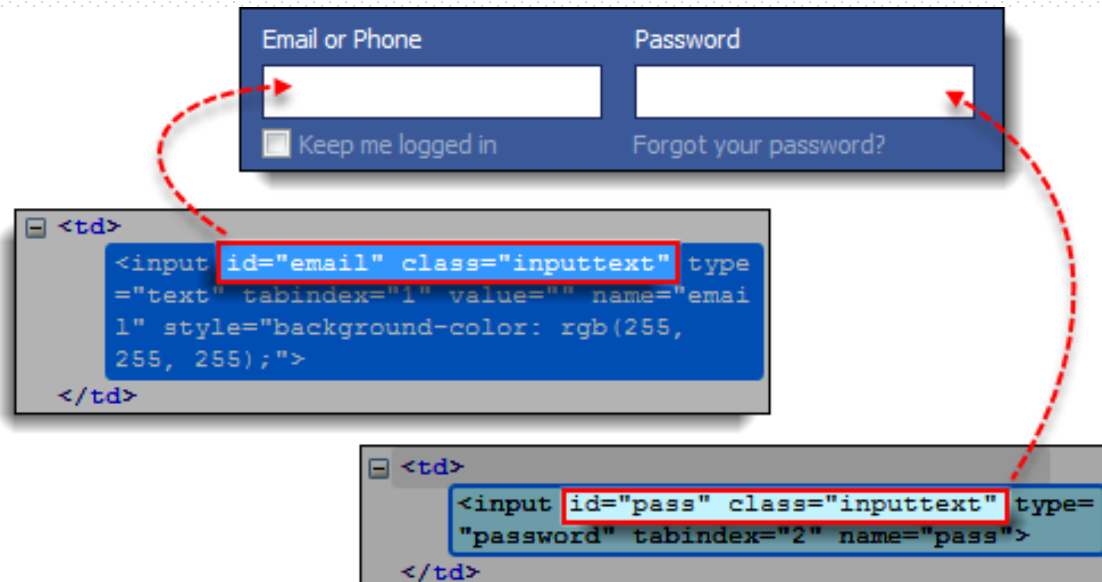


Diagram illustrating the HTML source code for a login form with two input fields. The first input field is labeled "Email or Phone" and the second is labeled "Password". Red dashed arrows connect the input fields to their respective HTML snippets.

```
<td>
  <input id="email" class="inputtext" type="text" tabindex="1" value="" name="email" style="background-color: rgb(255, 255, 255);">
</td>
```

```
<td>
  <input id="pass" class="inputtext" type="password" tabindex="2" name="pass">
</td>
```

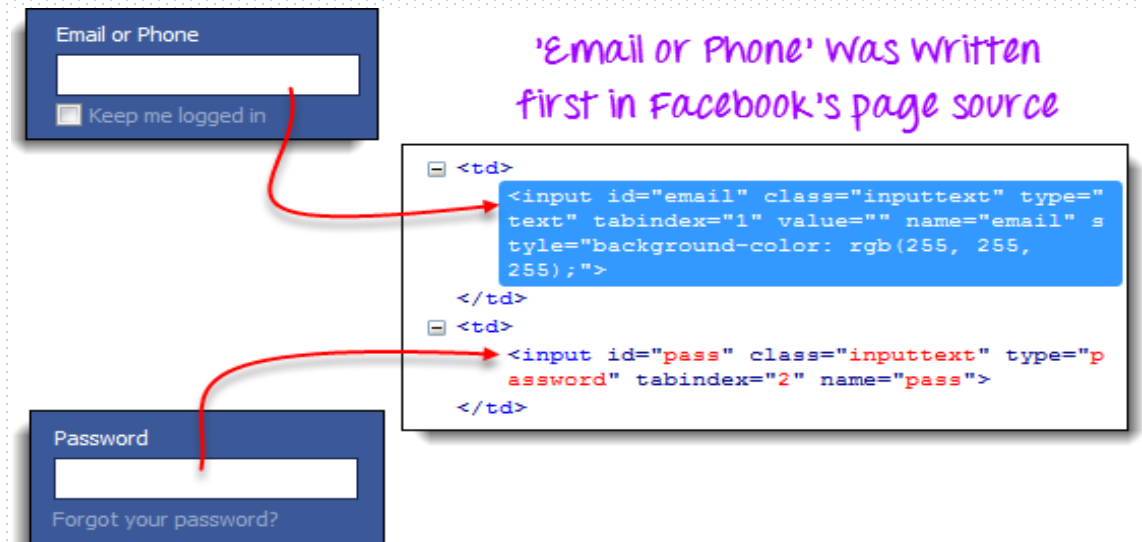


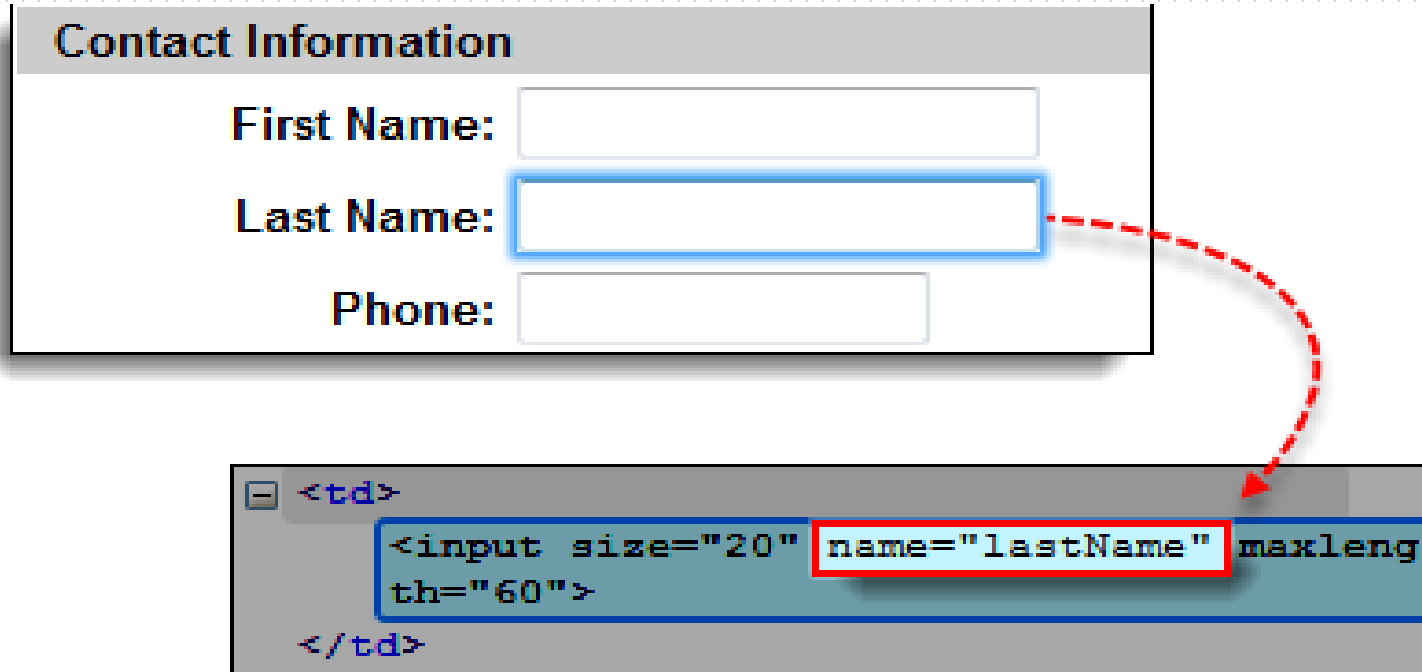
Diagram illustrating the HTML source code for a login form with two input fields. The first input field is labeled "Email or Phone" and the second is labeled "Password". Red solid arrows connect the input fields to their respective HTML snippets.

'Email or Phone' was written first in Facebook's page source

```
<td>
  <input id="email" class="inputtext" type="text" tabindex="1" value="" name="email" style="background-color: rgb(255, 255, 255);">
</td>
<td>
  <input id="pass" class="inputtext" type="password" tabindex="2" name="pass">
</td>
```

CSS Selector – *Tag* and *Attribute*

```
driver.findElement(By.cssSelector("input[name=lastName]")).sendKeys("xxxxx");
```



Contact Information

First Name:

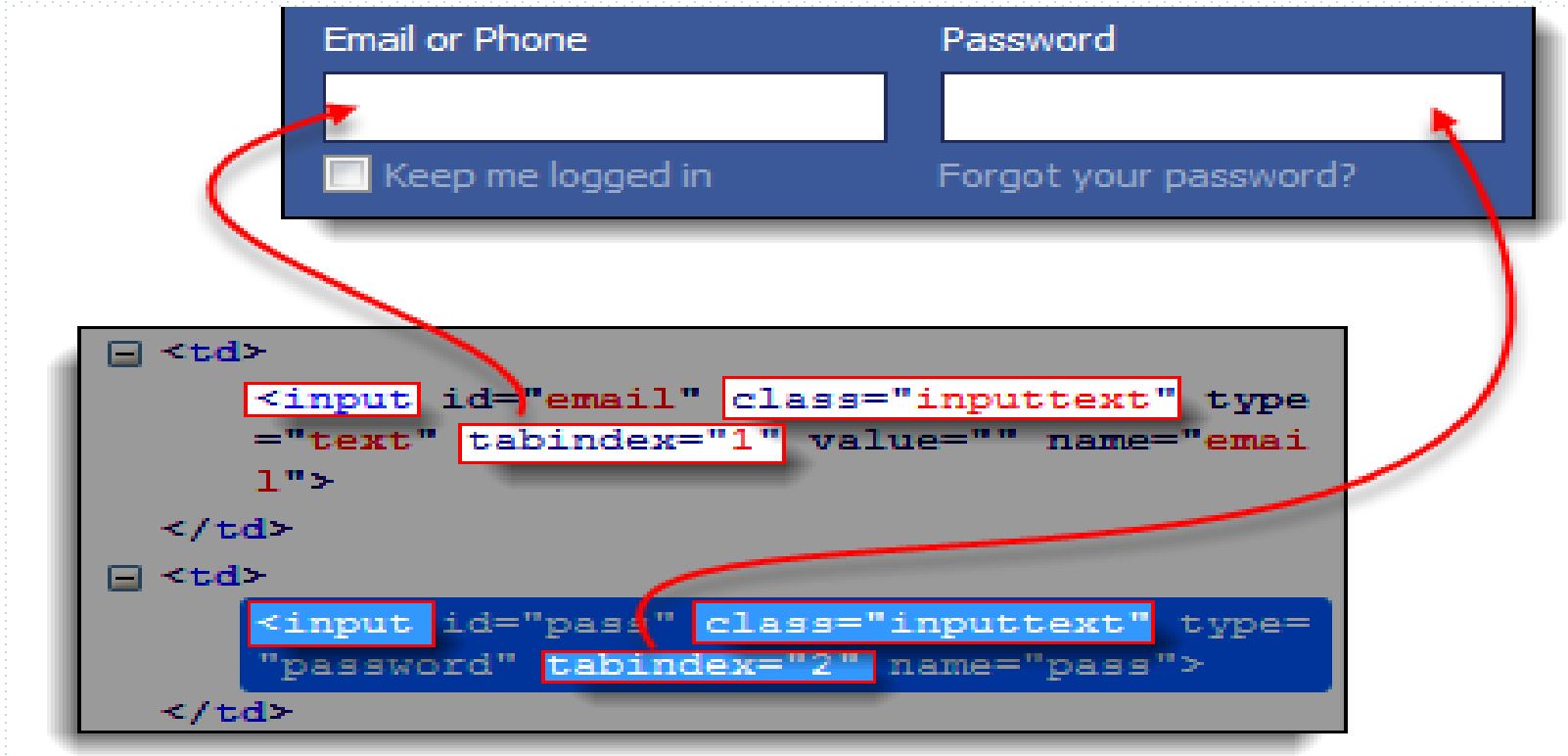
Last Name:

Phone:

```
<td>  
  <input size="20" name="lastName" maxleng  
    th="60">  
</td>
```

CSS Selector - *Tag*, *class* and *attribute*

```
driver.findElement (By.cssSelector("input.inputtext[tabindex=2]").sendKeys("xxxx");
```



The diagram illustrates the mapping between a web form and its underlying HTML structure. The form at the top contains two input fields: "Email or Phone" and "Password". Below the "Email or Phone" field is a checkbox labeled "Keep me logged in". Below the "Password" field is a link labeled "Forgot your password?". The HTML code at the bottom shows the structure of these elements. The first code block, enclosed in `<td>` tags, defines the "Email or Phone" input field with attributes `id="email"`, `class="inputtext"`, `type="text"`, `tabindex="1"`, `value=""`, and `name="email"`. The second code block defines the "Password" input field with attributes `id="pass"`, `class="inputtext"`, `type="password"`, `tabindex="2"`, and `name="pass"`. Red arrows originate from the "Email or Phone" and "Password" input fields in the form and point to their respective `<input>` tags in the HTML code, highlighting the `class="inputtext"` and `tabindex` attributes used in the CSS selector.

```
<td>  
  <input id="email" class="inputtext" type="text" tabindex="1" value="" name="email">  
</td>  
<td>  
  <input id="pass" class="inputtext" type="password" tabindex="2" name="pass">  
</td>
```

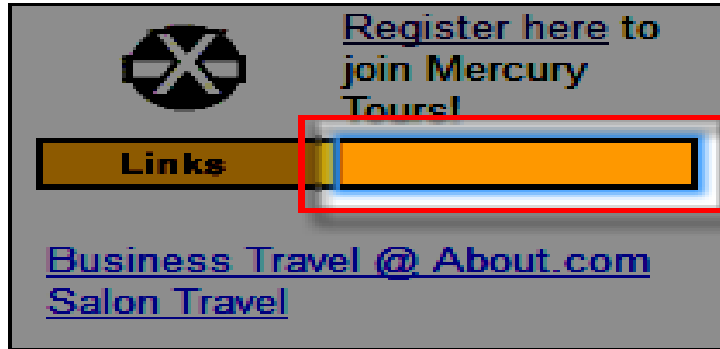
XPath

- XPath is for locating the elements or nodes in XML documents
- XML and HTML has similar syntax (HTML is a subset of XML)
- Hence XPath can be used to locate elements in HTML pages (web pages).

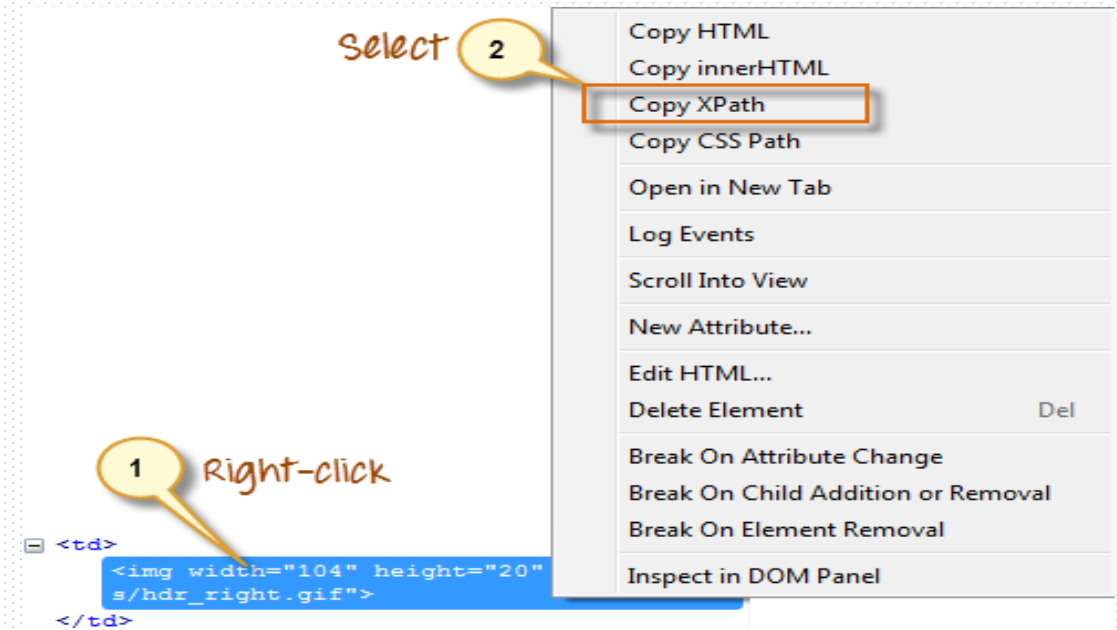
```
<div id="rb_shell">
  <div id="omnibarAd">
    <header id="rbHeader" class="pfABTest noTextAd" name="rb_header"
      <div class="int">
        <nav id="primaryNav">
          <ul id="primaryNavBar" class="section" section="brandnav">
            <li class="expandable editions">
              <li class="expandable site7">
                <a class="menuHead " href="http://reviews.cnet.com/">
                  <div class="menuWrapper">
                    <nav>
                      <h3 id="topCategoriesNav">Top Categories</h3>
                      <h3 id="moreCategoriesNav">More Categories</h3>
                      <div id="reviewsNavLinkCategories">
                        <ul>
                          <li>
                            <a href="http://reviews.cnet.com/car-tech/"
                              </li>
```

header/nav/ul/li[2]/div/nav/div/ul/li/a

Capture XPath



```
<td>  
    
</td>
```



Types of XPath

➤ There are two types of XPath:

1. Absolute XPath .
2. Relative XPath .

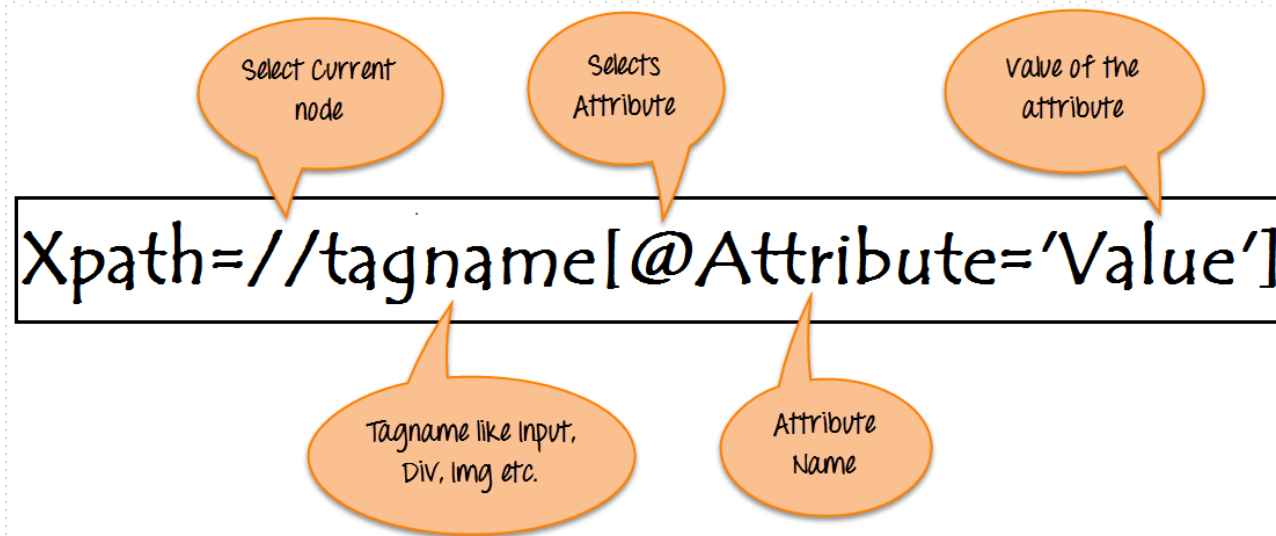
Absolute XPath

- It is the direct way to find the element, but the disadvantage of the absolute XPath is that if there are any changes made in the path of the element then that XPath gets failed.
- The key characteristic of XPath is that it begins with the single forward slash(/) ,which means you can select the element from the root node.



Relative XPath

- For Relative Xpath the path starts from the middle of the HTML DOM structure. Its start with the double forward slash (//), which means it can search the element anywhere at the webpage.



Syntax for XPath

- XPath contains the path of the element situated at the web page.
- Standard syntax for creating XPath is.

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

// : Select current node.

Tagname: Tagname of the particular node.

@: Select attribute.

Attribute: Attribute name of the node.

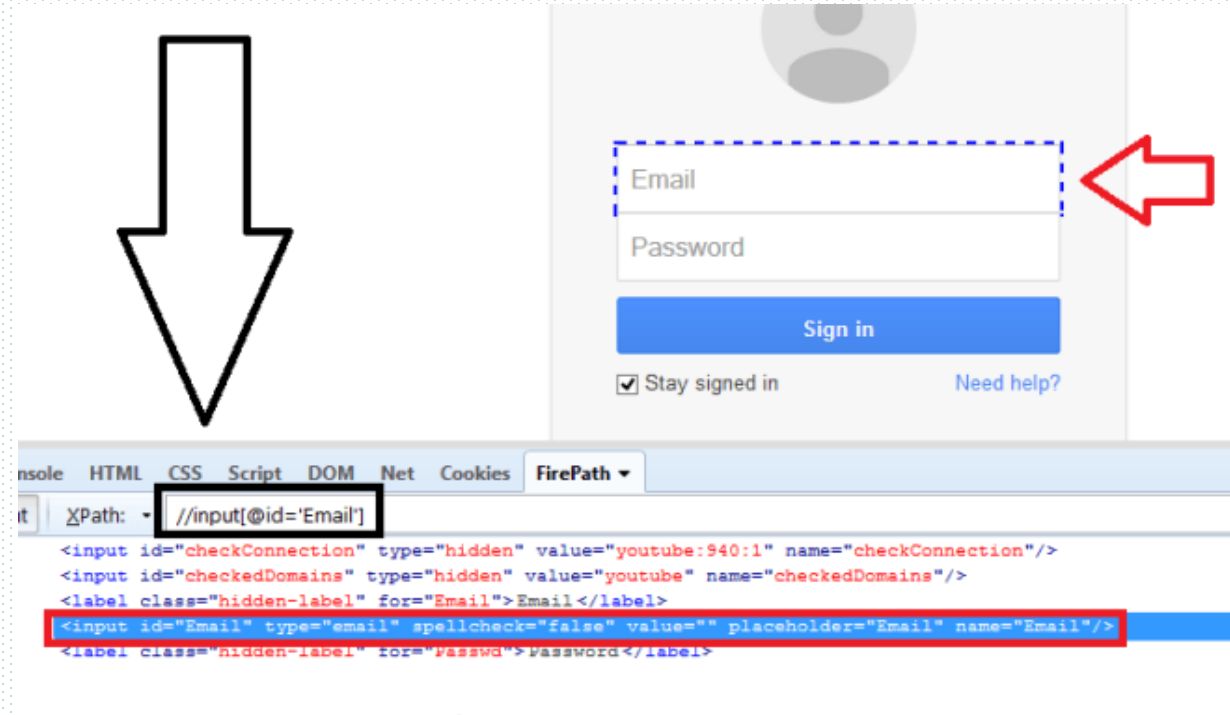
Value: Value of the attribute.

XPath with Single attribute

`// tagname[@attribute-name='value1']`

Examples:

```
// a [@href='http://www.google.com']
//input[@id='name']
//input[@name='username']
//img[@alt='sometext']
```



The screenshot shows a web form with an email input field, a password input field, and a blue 'Sign in' button. A red arrow points to the email input field. Below the form is the FirePath DOM viewer. The XPath expression `//input[@id='Email']` is entered in the search bar. The DOM tree shows the following structure:

```
<input id="checkConnection" type="hidden" value="youtube:940:1" name="checkConnection"/>
<input id="checkedDomains" type="hidden" value="youtube" name="checkedDomains"/>
<label class="hidden-label" for="Email">Email</label>
<input id="Email" type="email" spellcheck="false" value="" placeholder="Email" name="Email"/>
<label class="hidden-label" for="Password">Password</label>
```

The `<input id="Email" type="email" spellcheck="false" value="" placeholder="Email" name="Email"/>` line is highlighted in blue.

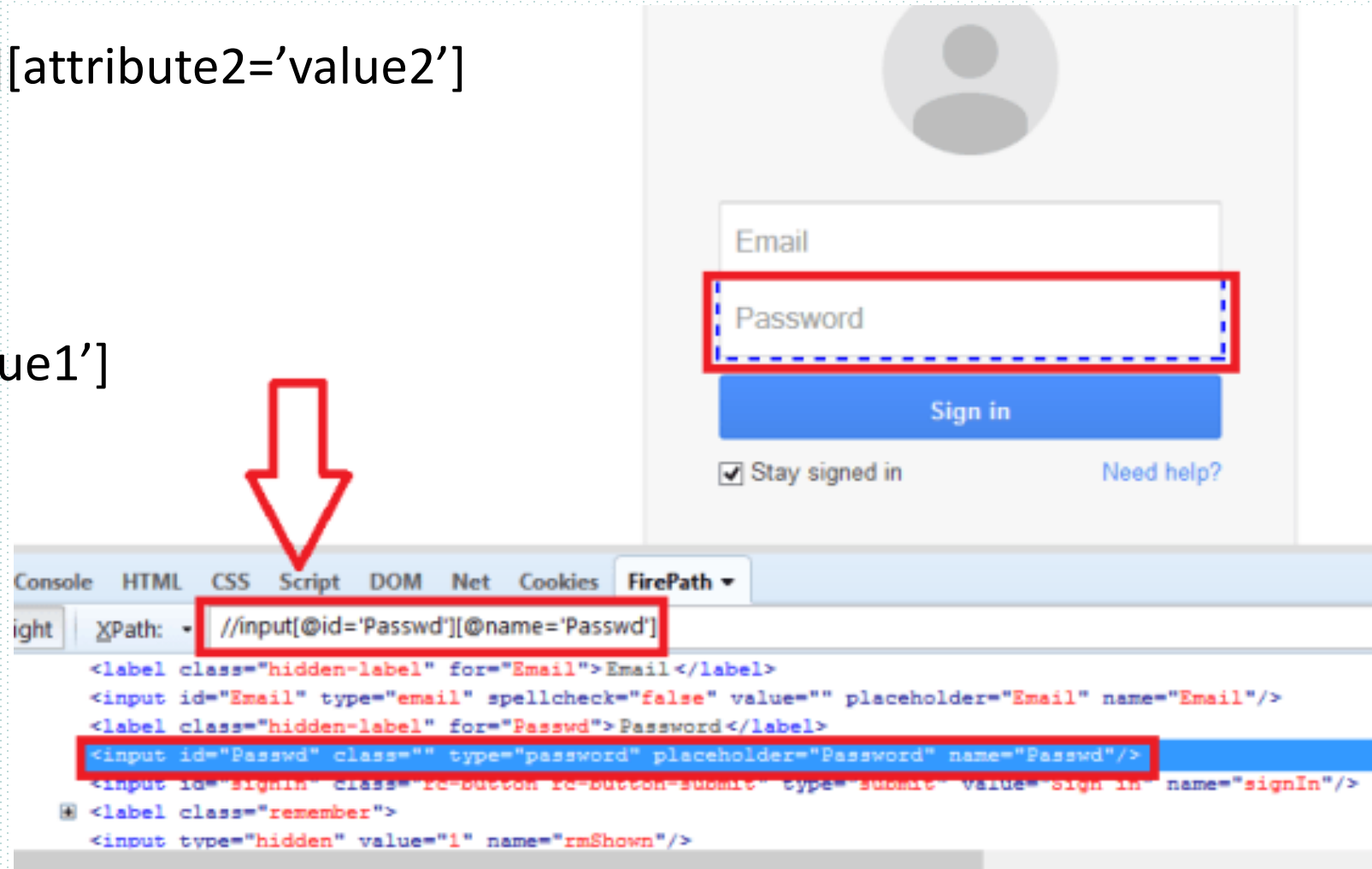
XPath with Multiple attributes

`//tagname[@attribute1='value1'][attribute2='value2']`

Examples:

`//a[@id='id1'][@name='namevalue1']`

`//img[@src=''][@href='']`



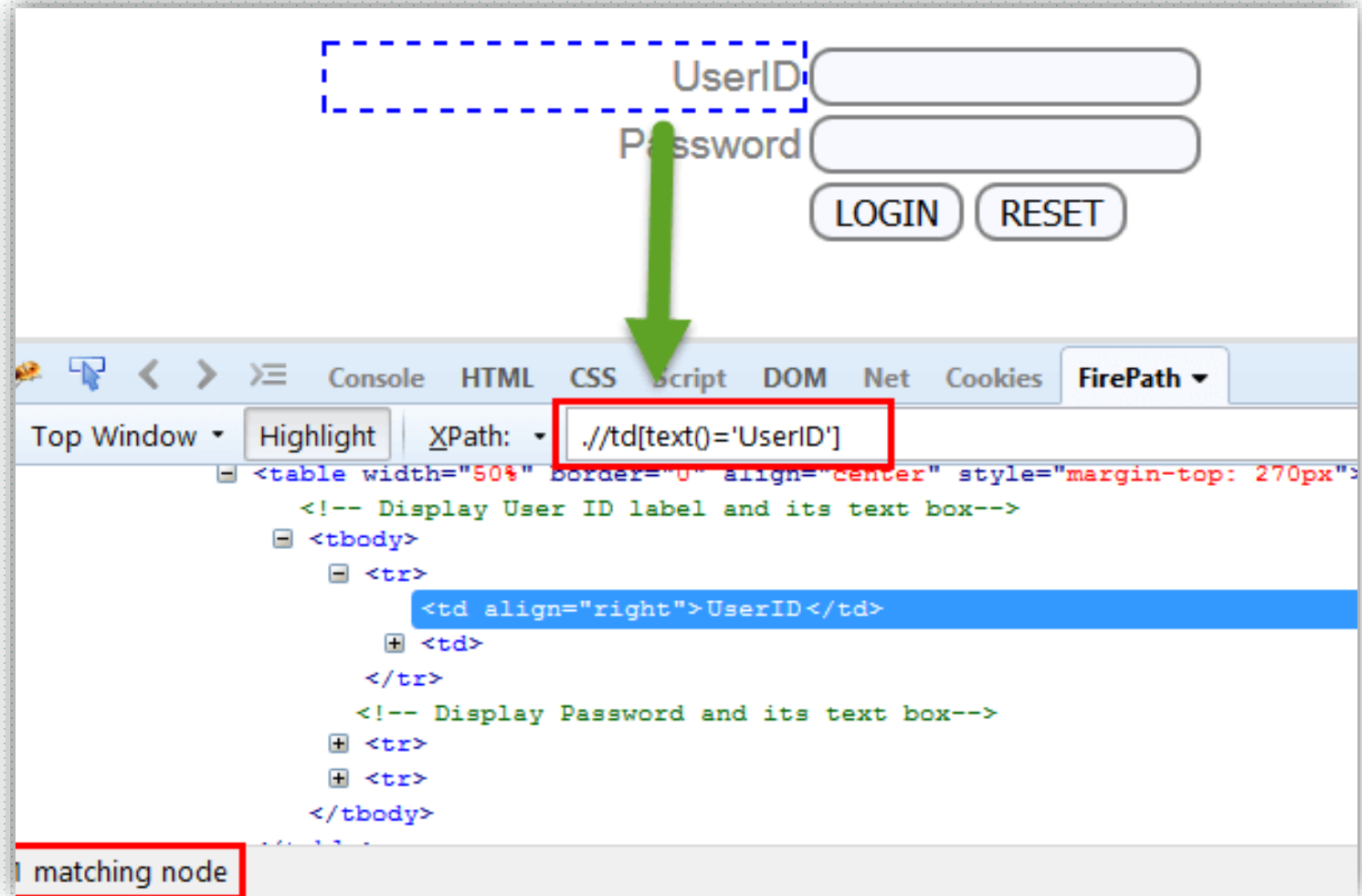
The image shows a web form with a login section. The form includes an email input field, a password input field (highlighted with a red dashed border), a 'Sign in' button, a 'Stay signed in' checkbox, and a 'Need help?' link. Below the form is the FirePath DOM inspector. A red arrow points from the password input field to the FirePath console. The console shows the XPath expression `//input[@id='Passwd'][@name='Passwd']` and the corresponding HTML structure. The HTML structure includes labels for 'Email' and 'Password', and input fields for 'Email', 'Password', 'Sign in', and a hidden input for 'rmShown'.

```

XPath: //input[@id='Passwd'][@name='Passwd']
<label class="hidden-label" for="Email">Email</label>
<input id="Email" type="email" spellcheck="false" value="" placeholder="Email" name="Email"/>
<label class="hidden-label" for="Passwd">Password</label>
<input id="Passwd" class="" type="password" placeholder="Password" name="Passwd"/>
<input id="signin" class="rc-button rc-button-submit" type="submit" value="Sign in" name="signIn"/>
<label class="remember">
  <input type="hidden" value="1" name="rmShown"/>
  
```

XPath with text() method

- Xpath=//td[**text()='UserID'**]



The screenshot illustrates the process of finding an element using XPath. At the top, a login form is shown with a 'UserID' label and input field, a 'Password' label and input field, and 'LOGIN' and 'RESET' buttons. A green arrow points from the 'UserID' label to the FirePath tool below. The FirePath tool shows the XPath query `./td[text()='UserID']` entered in the 'XPath:' field. The DOM tree below shows the HTML structure, with the `<td align="right">UserID</td>` element highlighted. The status bar at the bottom indicates '1 matching node'.

```

<table width="50%" border="0" align="center" style="margin-top: 270px">
  <!-- Display User ID label and its text box-->
  <tbody>
    <tr>
      <td align="right">UserID</td>
      <td>
        <input type="text"/>
      </td>
    </tr>
    <!-- Display Password and its text box-->
    <tr>
      <td align="right">Password</td>
      <td>
        <input type="password"/>
      </td>
    </tr>
  </tbody>
</table>

```

XPath with Contains() method

`//tagname[contains(@attribute,'value1')]`

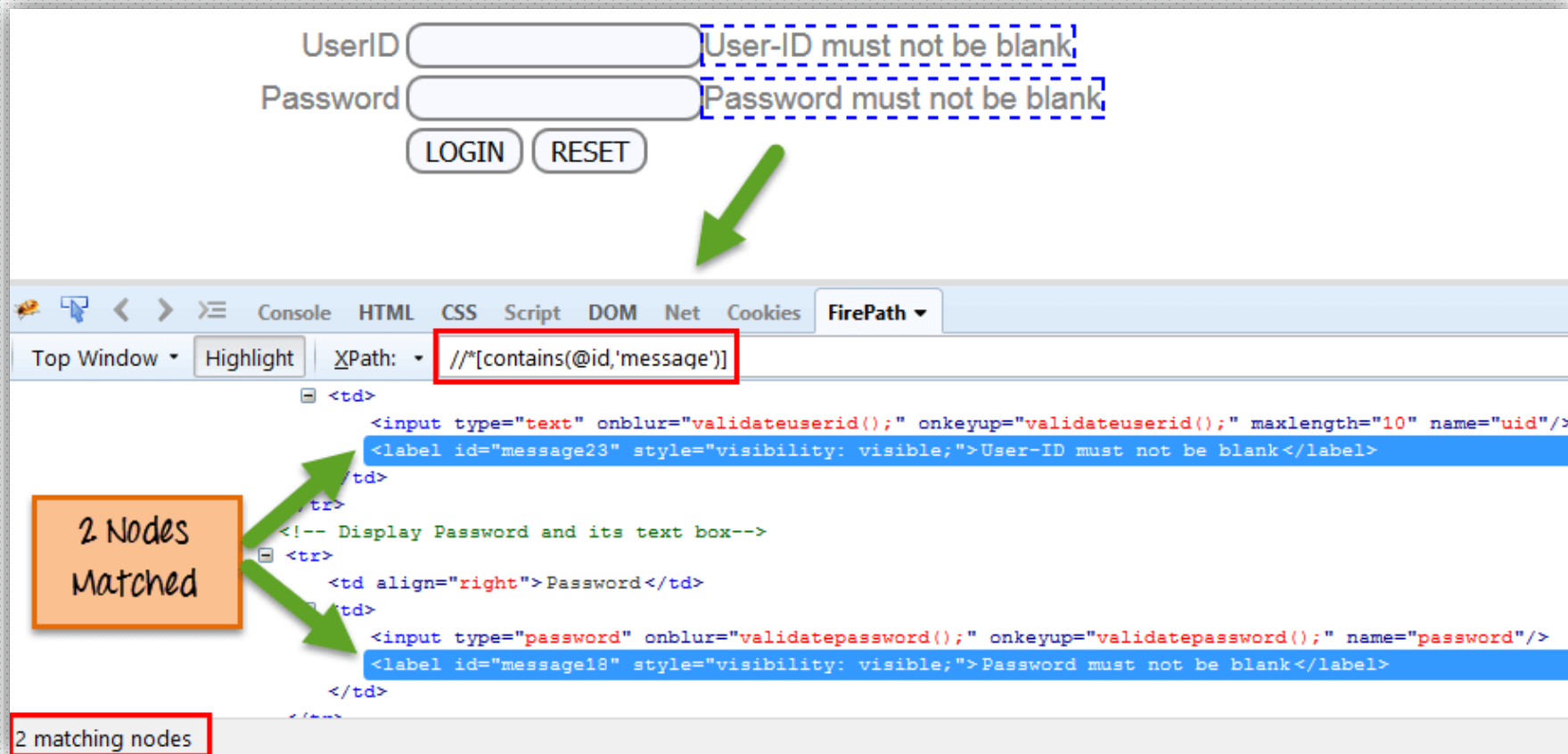
`//input[contains(@id,'')]`

`//input[contains(@name,'')]`

`//a[contains(@href,'')]`

`//img[contains(@src,'')]`

`//div[contains(@id,'')]`



The screenshot illustrates the use of the XPath `contains()` method. At the top, a web form shows input fields for 'UserID' and 'Password', each with a corresponding error message: 'User-ID must not be blank' and 'Password must not be blank'. A green arrow points from these messages to the FirePath tool below.

The FirePath tool's XPath bar contains the query `//*[contains(@id,'message')]`. The results pane shows two matching nodes from the HTML DOM:

- `<label id="message23" style="visibility: visible;">User-ID must not be blank</label>`
- `<label id="message18" style="visibility: visible;">Password must not be blank</label>`

An orange box on the left states '2 Nodes Matched', and a red box at the bottom left of the results pane states '2 matching nodes'.

XPath with starts-with() method

`//tagname[starts-with(@attribute-name,"")]`

Examples:

`//id[starts-with(@id,"")]`

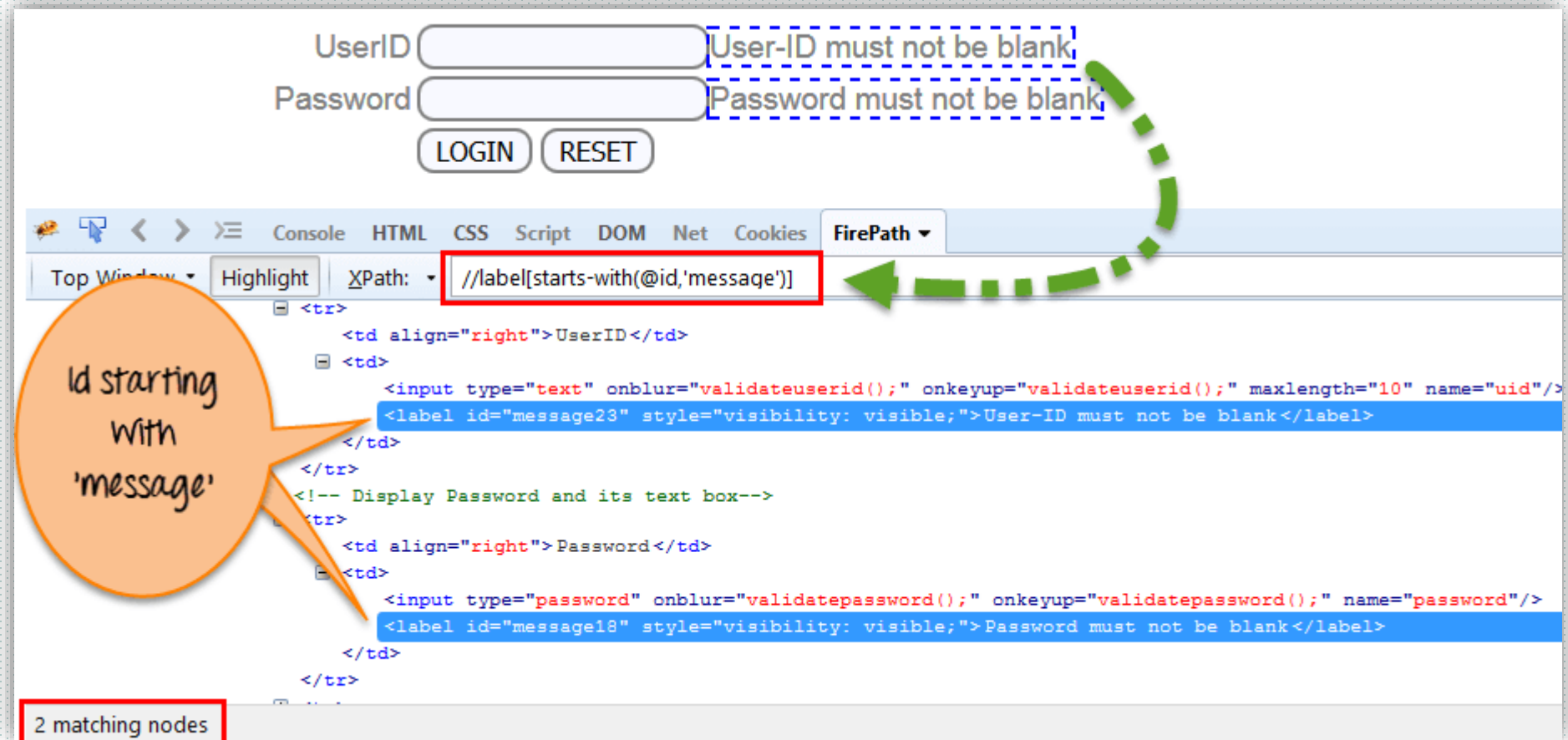
`//a[starts-with(@href,"")]`

`//img[starts-with(@src,"")]`

`//div[starts-with(@id,"")]`

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

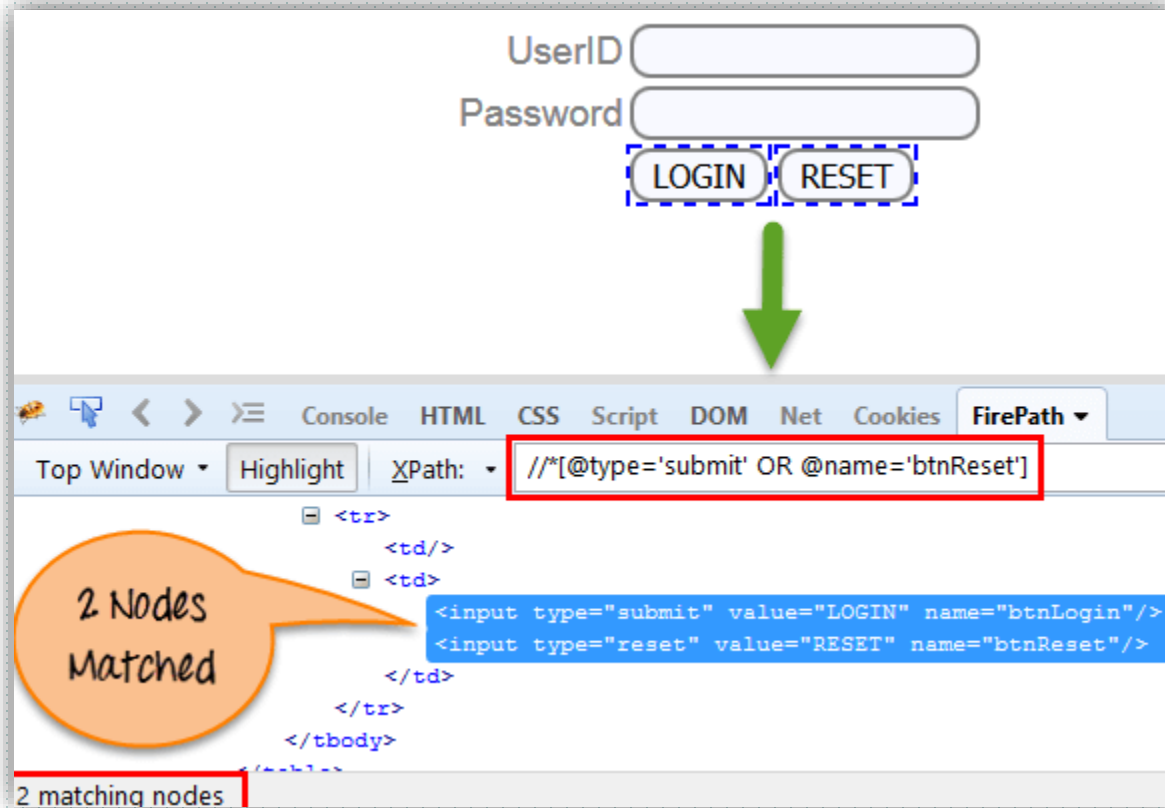
`//button[starts-with(@id,"")]`



The screenshot shows a web browser with a login form. The form has two input fields: "UserID" and "Password", each followed by a message: "User-ID must not be blank" and "Password must not be blank". Below the inputs are "LOGIN" and "RESET" buttons. The FirePath tool is open, showing the XPath expression `//label[starts-with(@id,'message')]` in the search bar. A green dashed arrow points from the error messages to the XPath expression. An orange speech bubble points to the XPath expression with the text "Id starting with 'message'". The bottom of the FirePath window shows "2 matching nodes".

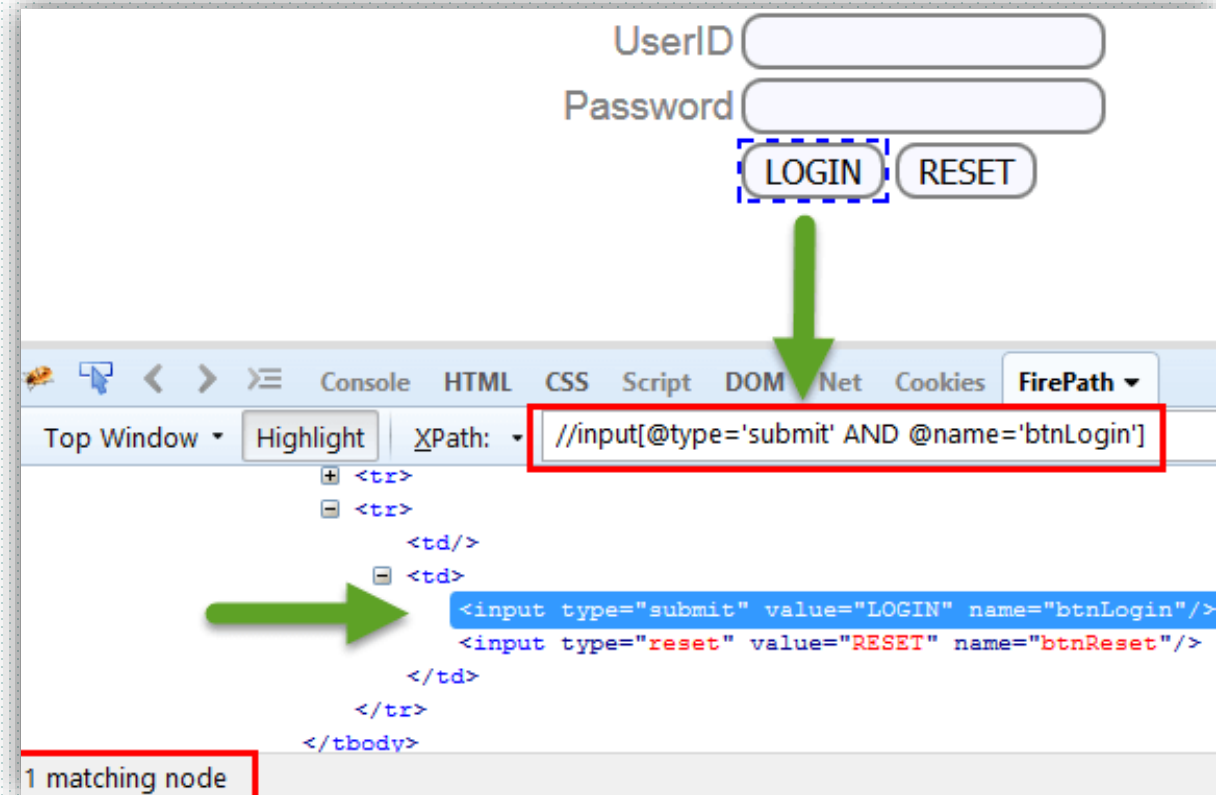
Using OR, AND with XPath

`//*[@type='submit' OR @name='btnReset']`



The screenshot shows a web form with 'UserID' and 'Password' fields, and 'LOGIN' and 'RESET' buttons. A green arrow points from the buttons to the FirePath tool. The XPath query `//*[@type='submit' OR @name='btnReset']` is entered and highlighted in a red box. The DOM tree shows two nodes highlighted in blue: `<input type="submit" value="LOGIN" name="btnLogin"/>` and `<input type="reset" value="RESET" name="btnReset"/>`. An orange speech bubble on the left says '2 Nodes Matched'. At the bottom, a red box contains the text '2 matching nodes'.

`//input[@type='submit' and @name='btnLogin']`

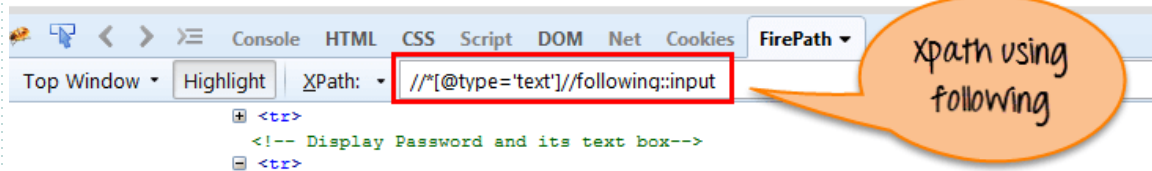
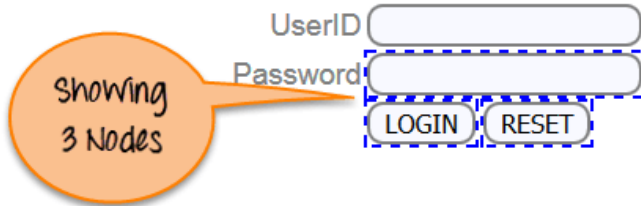


The screenshot shows the same web form. A green arrow points from the 'LOGIN' button to the FirePath tool. The XPath query `//input[@type='submit' AND @name='btnLogin']` is entered and highlighted in a red box. The DOM tree shows one node highlighted in blue: `<input type="submit" value="LOGIN" name="btnLogin"/>`. At the bottom, a red box contains the text '1 matching node'.

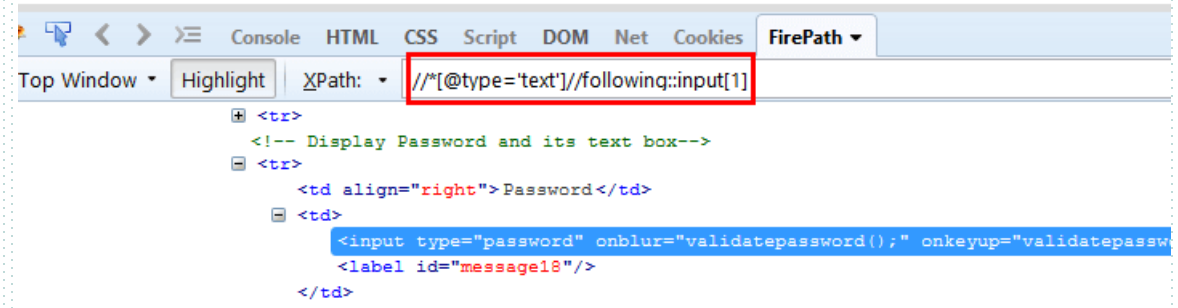
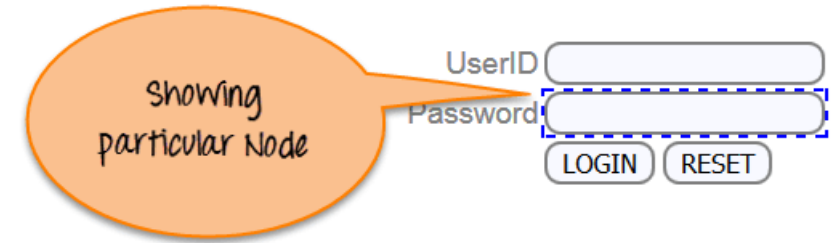
Following

- Selects all elements in the document of the current node

`//*[@type='text']//following::input`



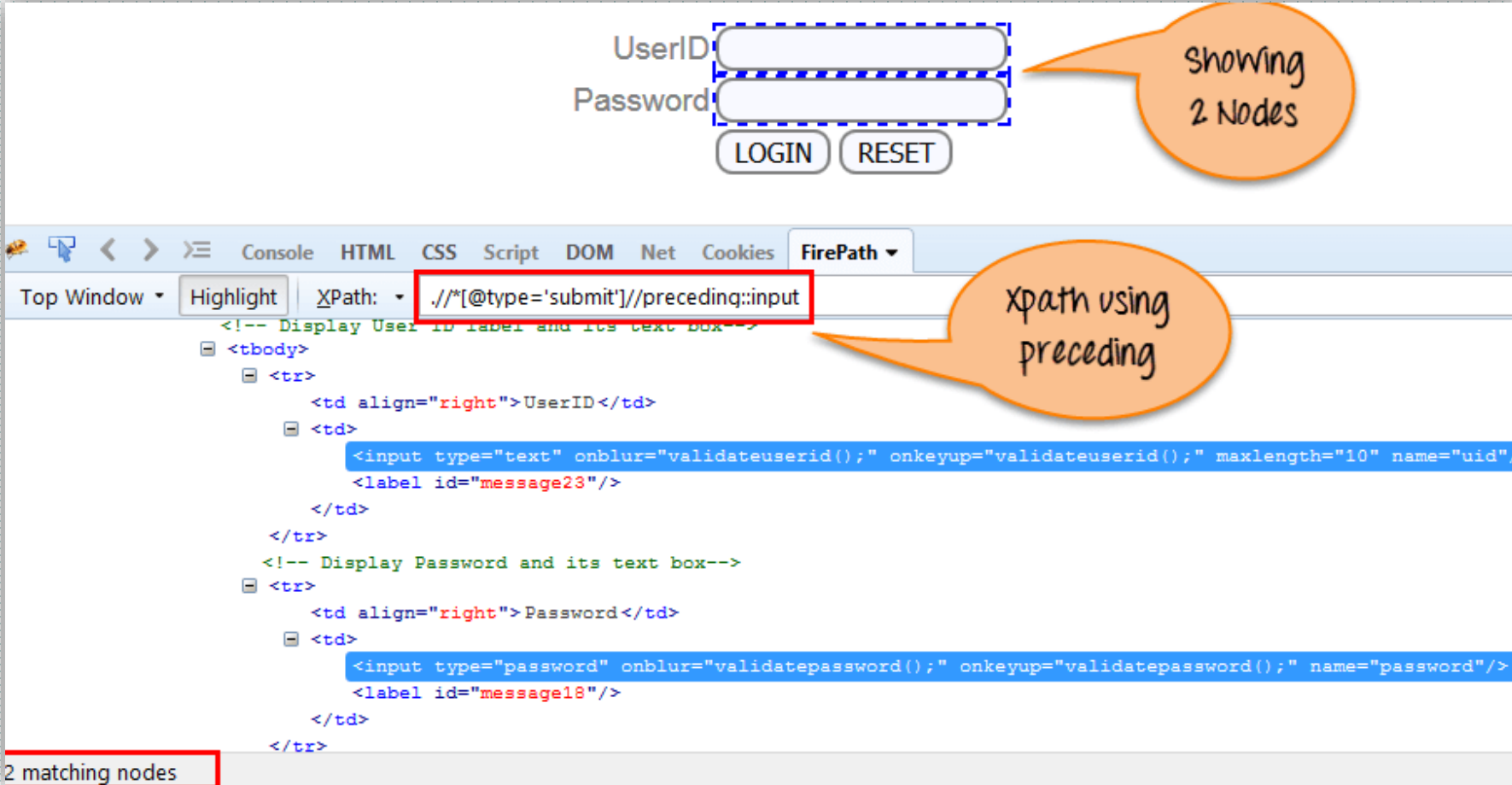
`//*[@type='text']//following::input[1]`



Preceding

- Select all nodes that come before the current node

`//*[@type='submit']//preceding::input`



UserID

Password

LOGIN RESET

Showing 2 Nodes

Top Window ▾ Highlight XPath: ▾ `//*[@type='submit']//preceding::input`

Xpath using preceding

```
<!-- Display User ID label and its text box-->
<tbody>
  <tr>
    <td align="right">UserID</td>
    <td>
      <input type="text" onblur="validateuserid();" onkeyup="validateuserid();" maxlength="10" name="uid"/>
      <label id="message23"/>
    </td>
  </tr>
  <!-- Display Password and its text box-->
  <tr>
    <td align="right">Password</td>
    <td>
      <input type="password" onblur="validatepassword();" onkeyup="validatepassword();" name="password"/>
      <label id="message18"/>
    </td>
  </tr>
</tbody>
```

2 matching nodes

Summary of Locators

Locators	Description	Example
By.className	finds elements based on the value of the "class" attribute	<code>findElement(By.className("someClassName"))</code>
By.cssSelector	finds elements based on the driver's underlying CSS Selector engine	<code>findElement(By.cssSelector("input#email"))</code>
By.id	locates elements by the value of their "id" attribute	<code>findElement(By.id("someId"))</code>
By.linkText	finds a link element by the exact text it displays	<code>findElement(By.linkText("REGISTRATION"))</code>
By.name	locates elements by the value of the "name" attribute	<code>findElement(By.name("someName"))</code>
By.partialLinkText	locates elements that contain the given link text	<code>findElement(By.partialLinkText("REG"))</code>
By.tagName	locates elements by their tag name	<code>findElement(By.tagName("div"))</code>
By.xpath	locates elements via XPath	<code>findElement(By.xpath("//html/body/div/table/tbody/tr/td[2]/table/tbody/tr[4]/td/table/tbody/tr/td[2]/table/tbody/tr[2]/td[3]/form/table/tbody/tr[5]"))</code>