Xpath vs CSS Selector: Everything you need to know about Xpath and CSS

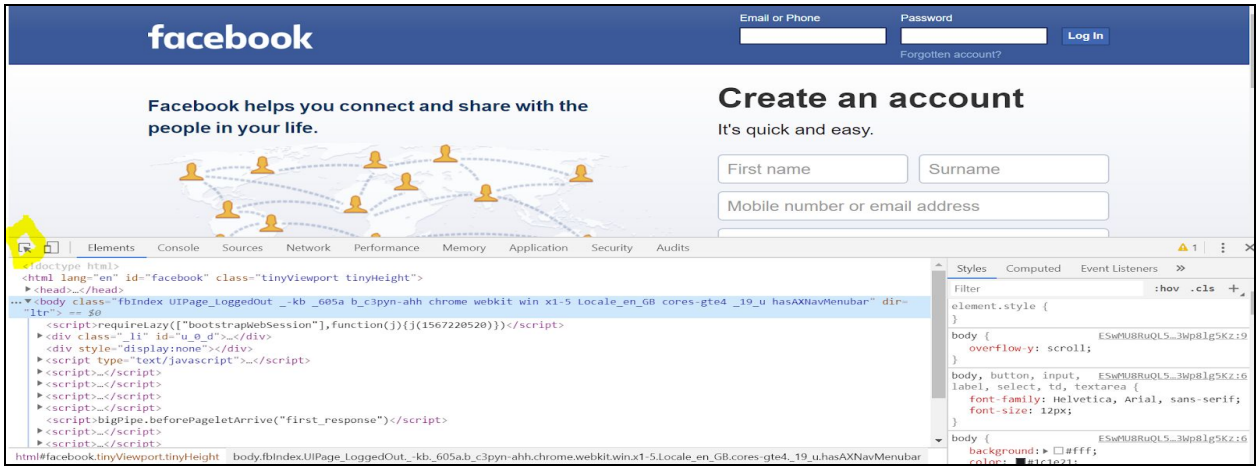
Automation testing is nothing but automating functionality of a web page running on a browser. And a web page is all about buttons, text boxes, dropdowns etc which we call elements. In Selenium, there are mainly 8 ways to find an element:

1. Id
2. Name
3. Class Name
4. Xpath
5. CSS Selector
6. Link text
7. Partial link text
8. Tag name

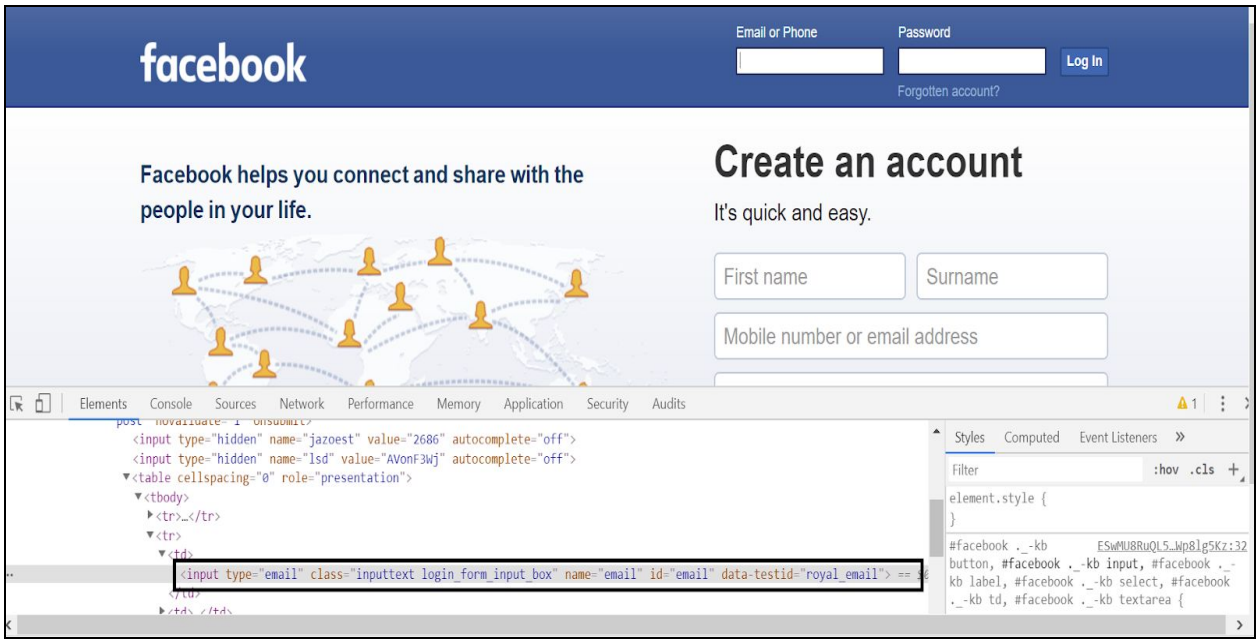
As a matter of fact, we should always go for id or name as they are the fastest as well as easier to make your code readable. However, there are instances when you are not able to get unique id, name or class name. For those cases, Xpath and CSS Selector comes handy. There are two main differences between Xpath and CSS Selector. First, CSS Selector is faster than Xpath and second, CSS Selector can not traverse DOM backward like Xpath. With this basic understanding, let's move on to learn how to find Xpath and CSS Selector.

The syntax of Xpath is ***//tagname[@attribute='value']*** where tagname is the tag of your element for example button for a button, input for a textbox or a checkbox, a for anchor tag etc.

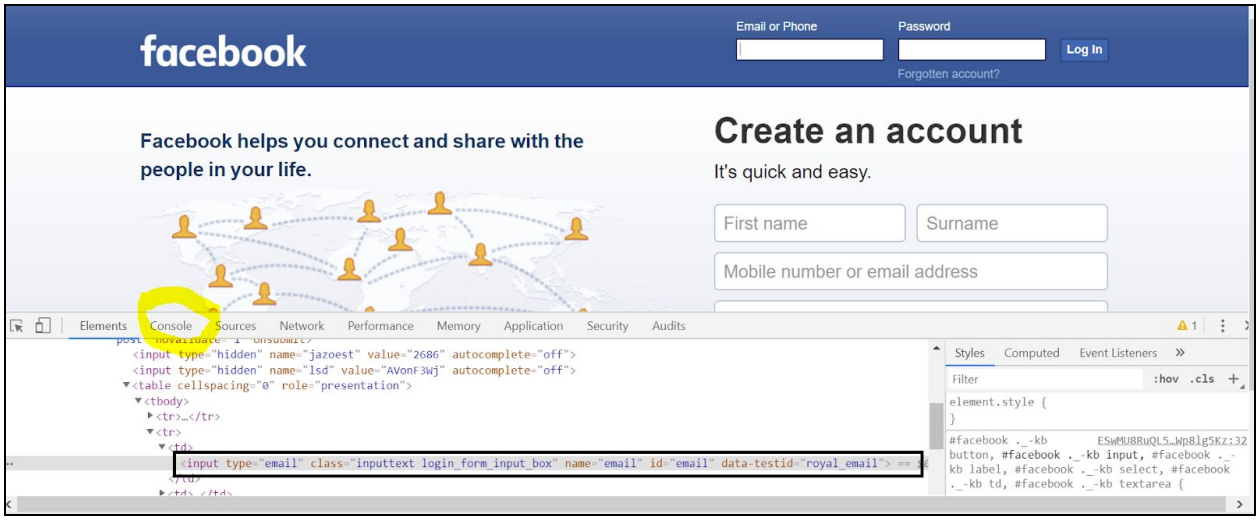
In order to find any locator, you need to click F12 key on your webpage and developer tools will be opened like the one shown below. I have highlighted an arrow button.



Now let’s move on to find our first Xpath. Click on the arrow button and then click on the Email or Phone field to get DOM of the username field.

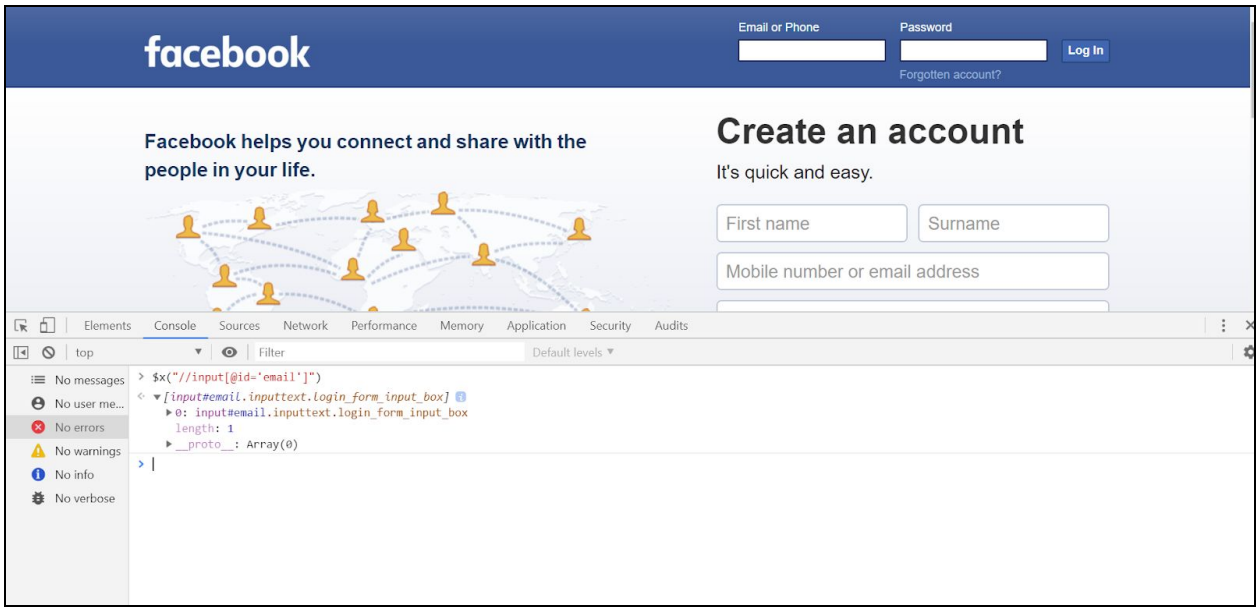


Click on Console to open a console window where you can write your Xpath and CSS Selector and you can test whether it is correct or not.



Now we know that Email or Phone field has a tag of input, it has a type attribute whose value is email, it has a name attribute whose value is email, it has an id attribute whose value is email and has a class with value or name as inputtext_login_form_input_box so our Xpath syntax is ***//tagname[@attribute=’value’]*** so for Email or Phone field our Xpath as per the syntax will be *//input[@id=’email’]* and to check whether its correct or not, we will use *$x* in the console.

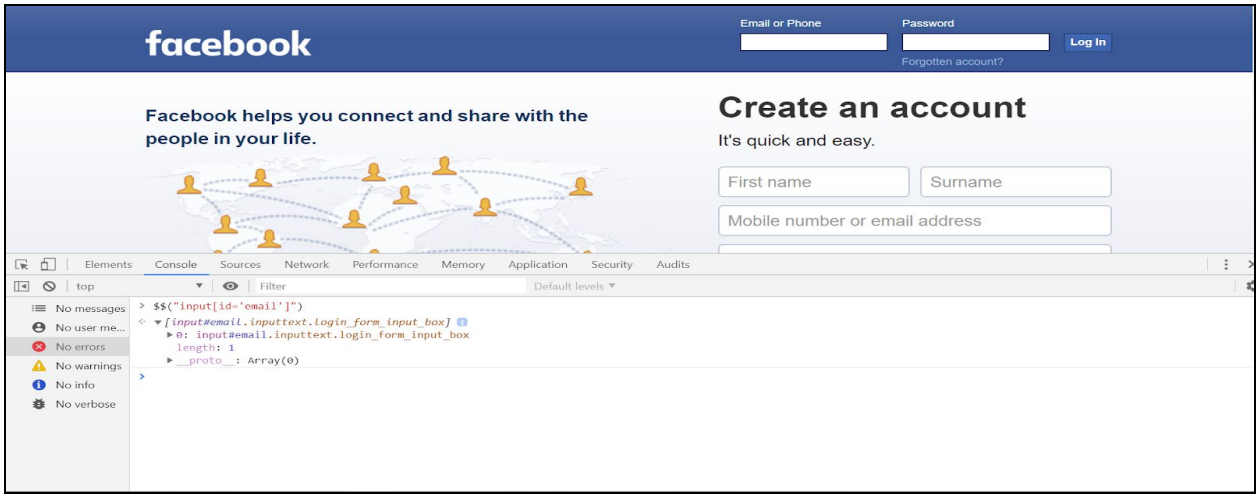
So, we will write *$x("//input[@id='email']")* in the console and we will get result with the element's class name and its matching count. In our case, we have uniquely identified the element so we will the length as 1.



So now you know that to test your Xpath, you just need to open console and write your Xpath inside $x() with double quotes.

But what if your team lead or manager asks you not to use Xpath and rather they want you to use CSS Selector. Well, nothing to worry it also as simple as Xpath.

The syntax of CSS Selector is ***tagname[attribute='value']*** so a corresponding CSS Selector for the above Xpath of Email or Phone field will be like *input[id='email']* and to test it in Console you will have to simply write your CSS Selector inside $$("your CSS Selector") and you will get the result. That's it.



But Simplicity of CSS Selector is not just limited to removing // before the tagname or @ before the attribute. It goes beyond that. For an Xpath with id attribute like *//button[@id='submit']* you can simply write a CSS Selector like *#submit* and for an Xpath like *//a[@class='mylink']* you can simply write a CSS Selector like *.mylink* that means you can prefix a # before id name and a . before class name to write a CSS Selector. Isn't it amazing? Your longer version of Xpath is reduced to just a word. But remember, this will work only for id and class name.

Next, I will discuss about starts-with, ends-with and contains methods of Xpath. Sometimes, despite our best efforts, we are not able to uniquely identify an element. It could be that id, name or class might be dynamically changing with only start part, end part or something in middle constant. For example, if you remember Facebook's Email or Phone field's class name is *inputtext.login_form_input_box* and if this is dynamically changing then you can do something like below:

Starts-with: //input[starts-with(@class, 'input')]

Ends-with: //input[ends-with(@class, 'input_box')]

Contains: //input[contains(@class, 'inputtext.login')]

Remember starts-with, ends-with, contains functions work not just with class name, but any attribute type like id, name, type, value etc.

Now its time to know similar functions in CSS Selector. Well in CSS Selector its extremely use to work with starts-with, ends-with and contains. See below.

Starts-with: input[class^='input']

Ends-with: input[class$='input_box']

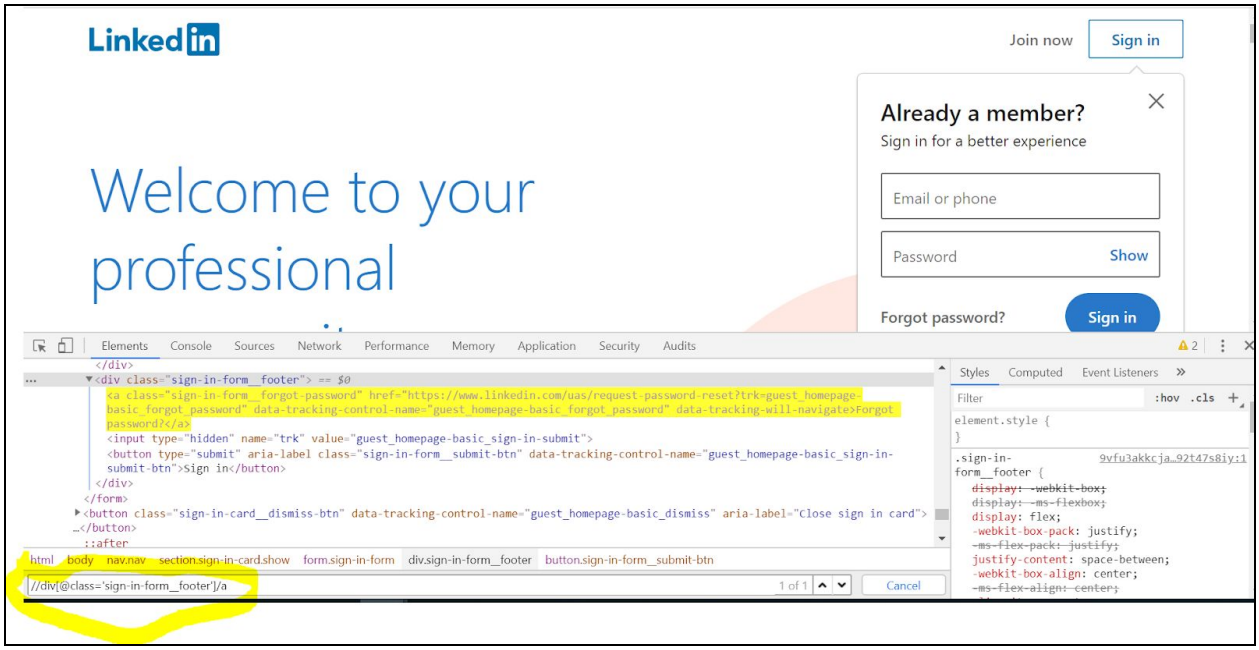
Contains: input[class*='inputtext.login']

So in a nutshell, in CSS Selector ^ stands for starts with, $ stands for ends with and * stands for contains. That's it.

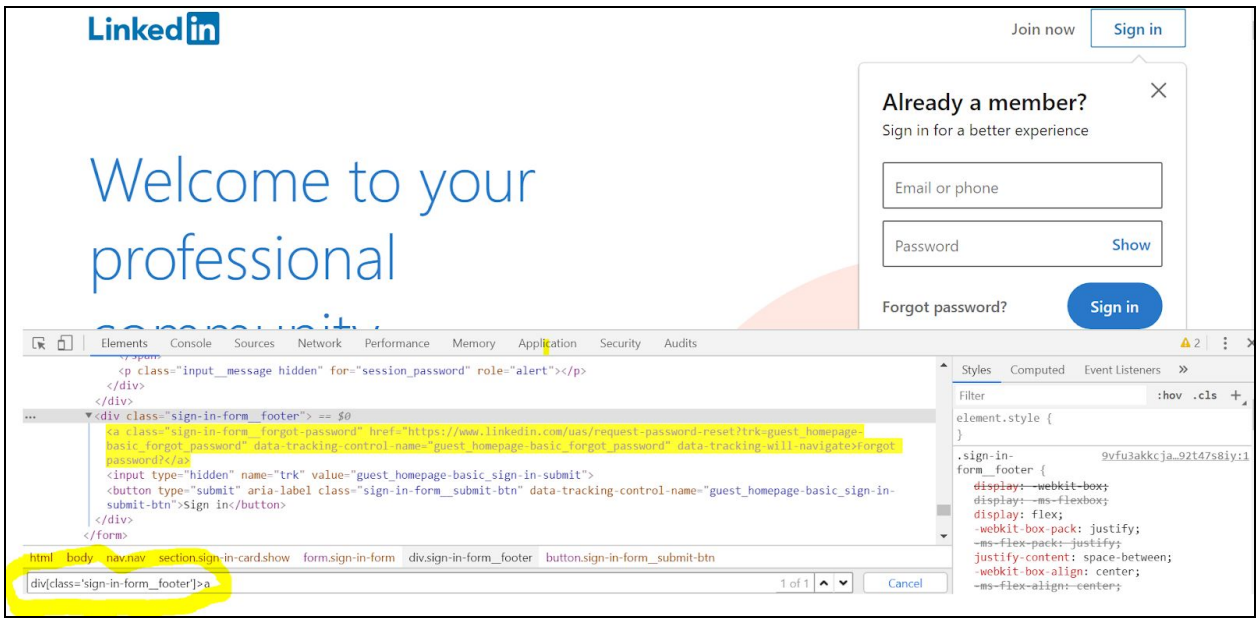
How to find direct child and subchild:

In interviews you are generally asked what is / and // in Xpath. Well, they are for navigating to children of an element. / is used to navigate to direct child while // to navigate to direct child as well as child inside any other child.

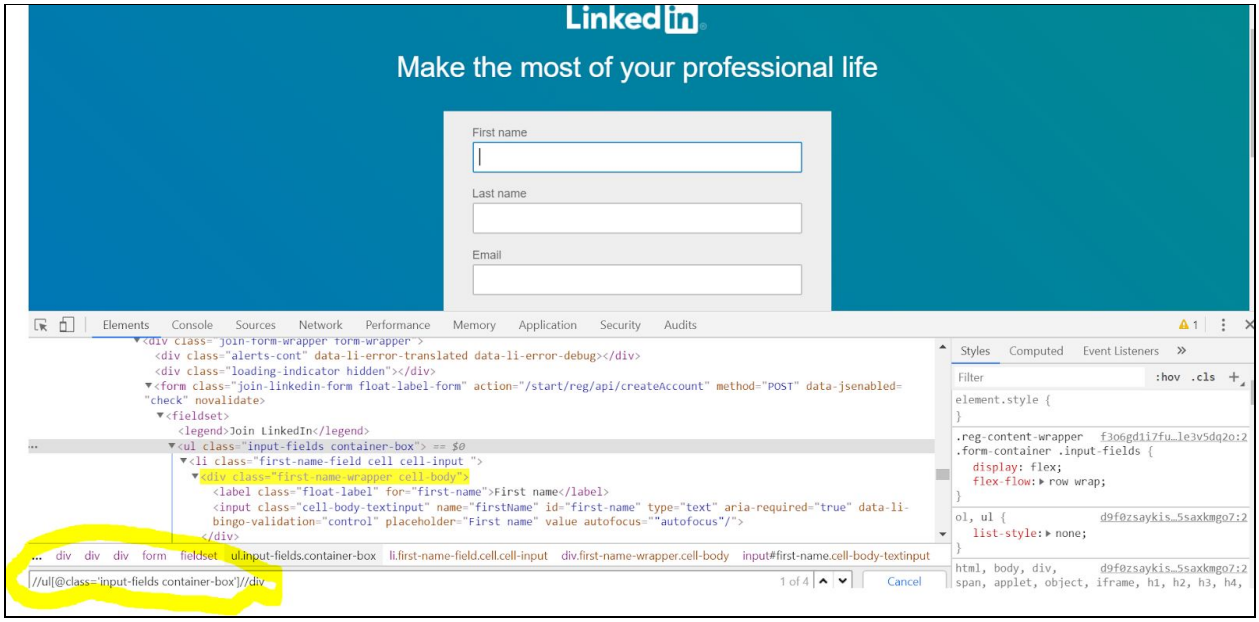
I will give you an example from LinkedIn login page. Forgot password and Sign in page are inside same div. They are the child of this div. So after finding Xpath of this div, you can find any of its children using / as shown in the below image. You can use /a or /input for input tag, /button for button tag and so on.



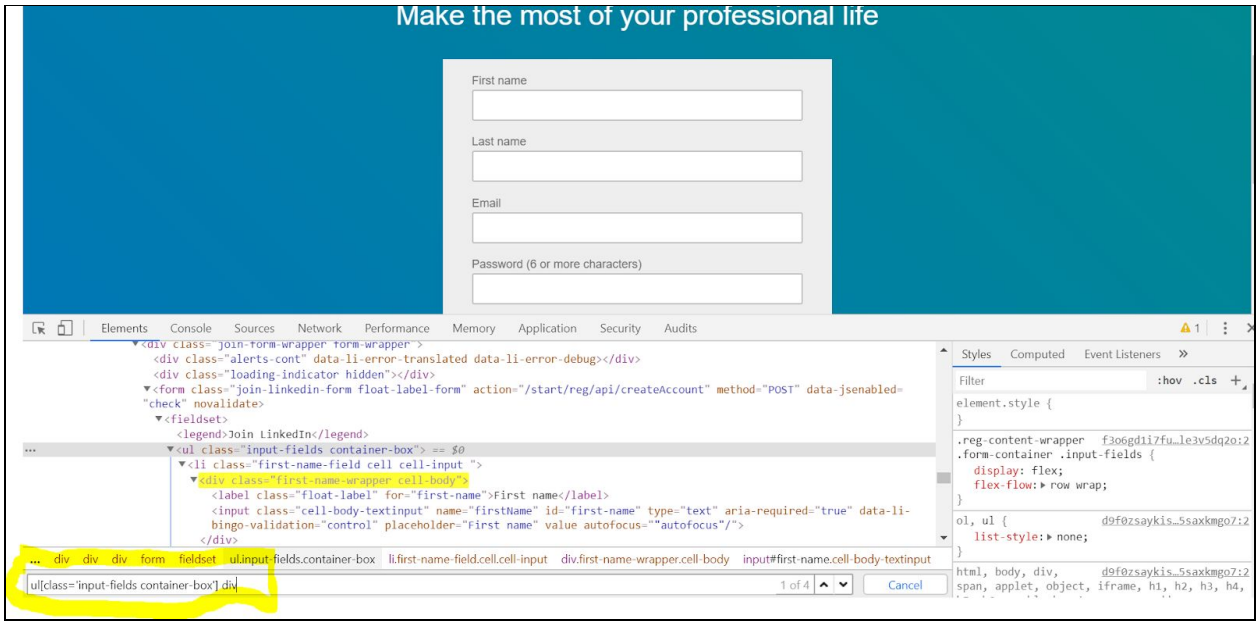
CSS Selector equivalent of / is > so in order to find direct child in CSS Selector you can do >input or >button or >a with the div as shown in blow image



Now let's come to // and its usage. You can find all the child and child within its child using // as shown in below image:

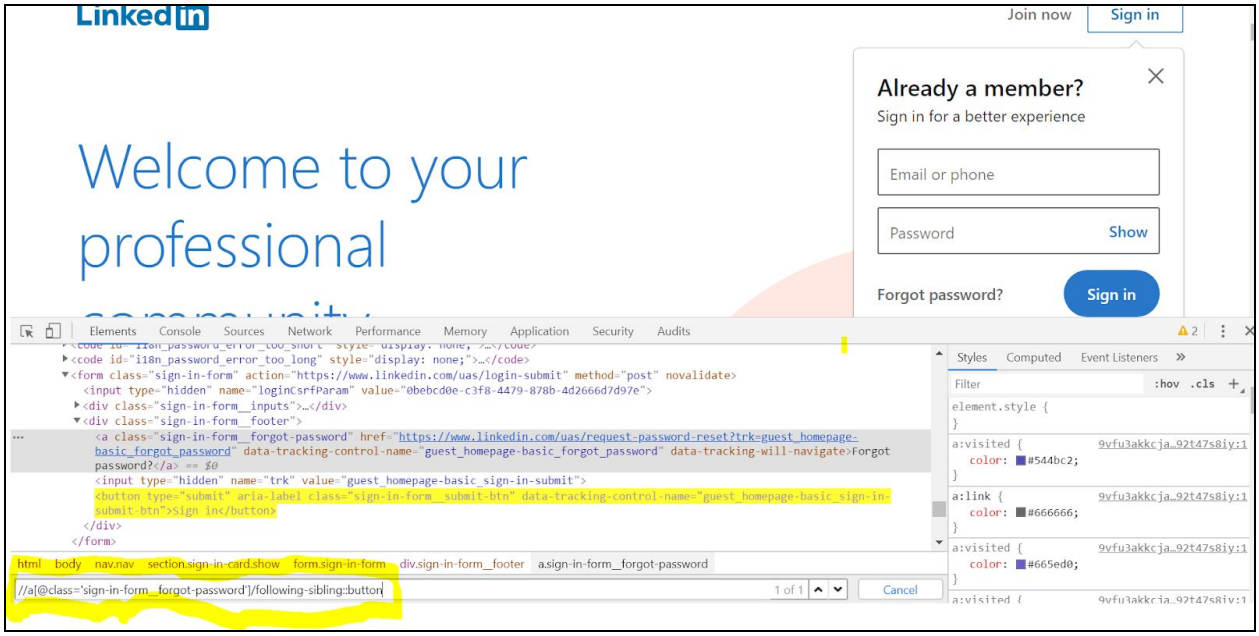


CSS equivalent of // is just a space followed by your CSS Selector as shown in the image:

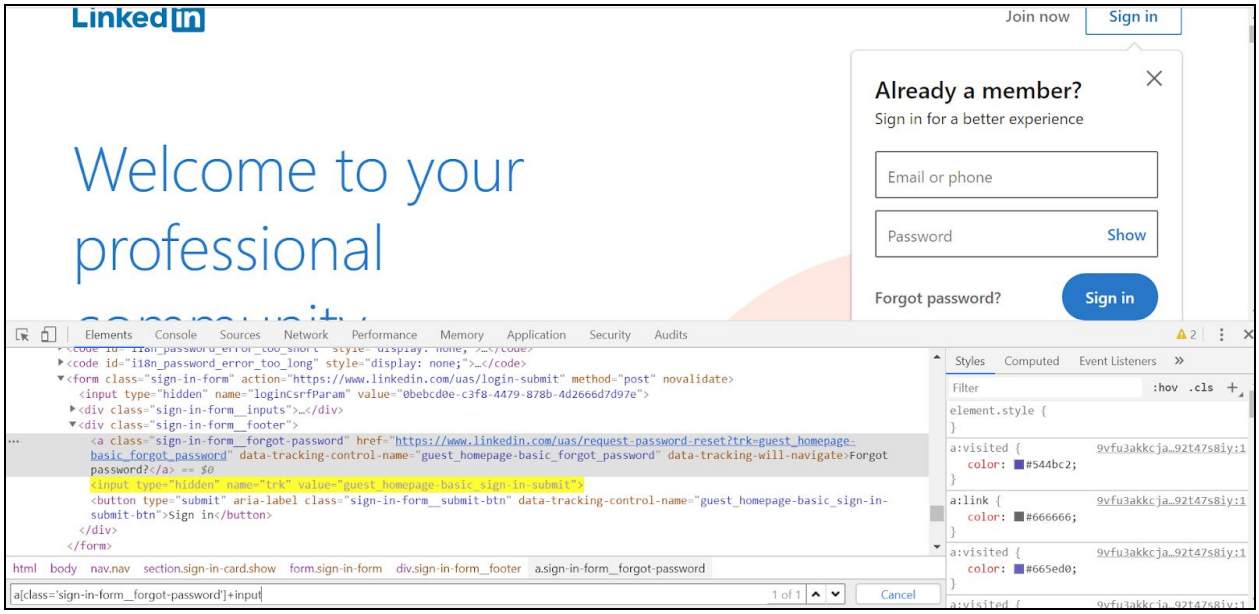


How to find Locators by Siblings:

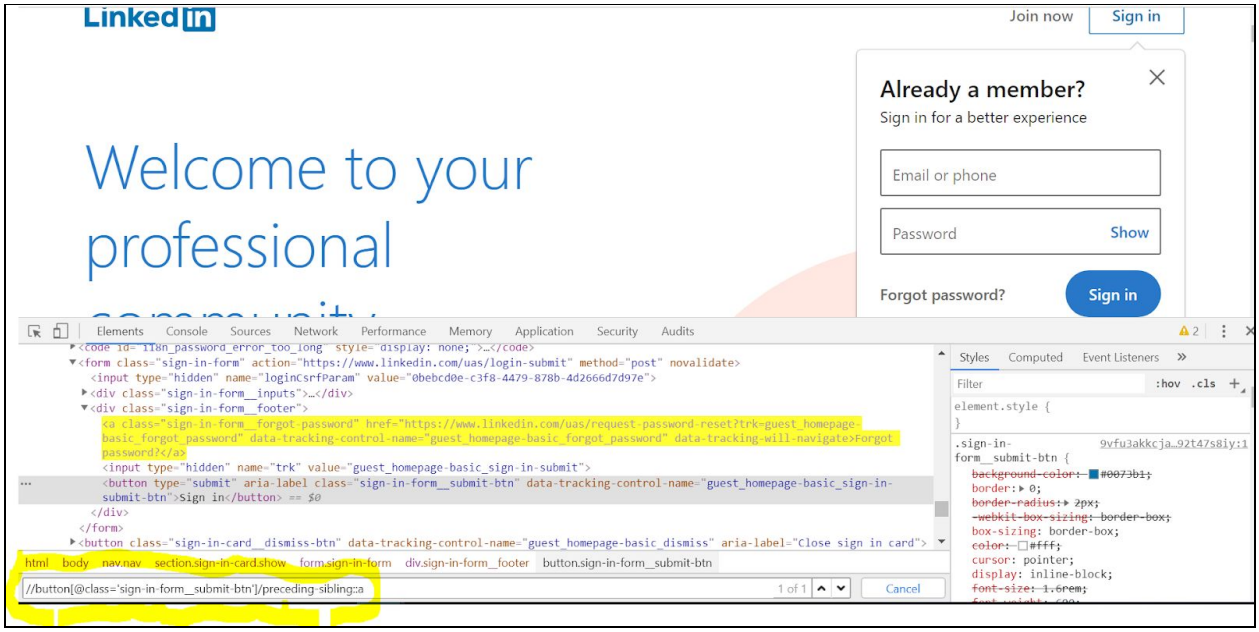
DOM elements at node level are considered siblings. To summarize, if their parent node is same then they are the siblings. For example, on LinkedIn login page, forgot password and Sign in buttons are siblings since their parent is same div with sign in form footer class. In Xpath, you can find locators by following-sibling and preceding-sibling. So you can find Sign in button locator using Xpath of forgot password as shown in below image:



CSS Selector equivalent of following-sibling is just a + sign. Once more CSS Selector simplicity comes into picture as shown in the image:



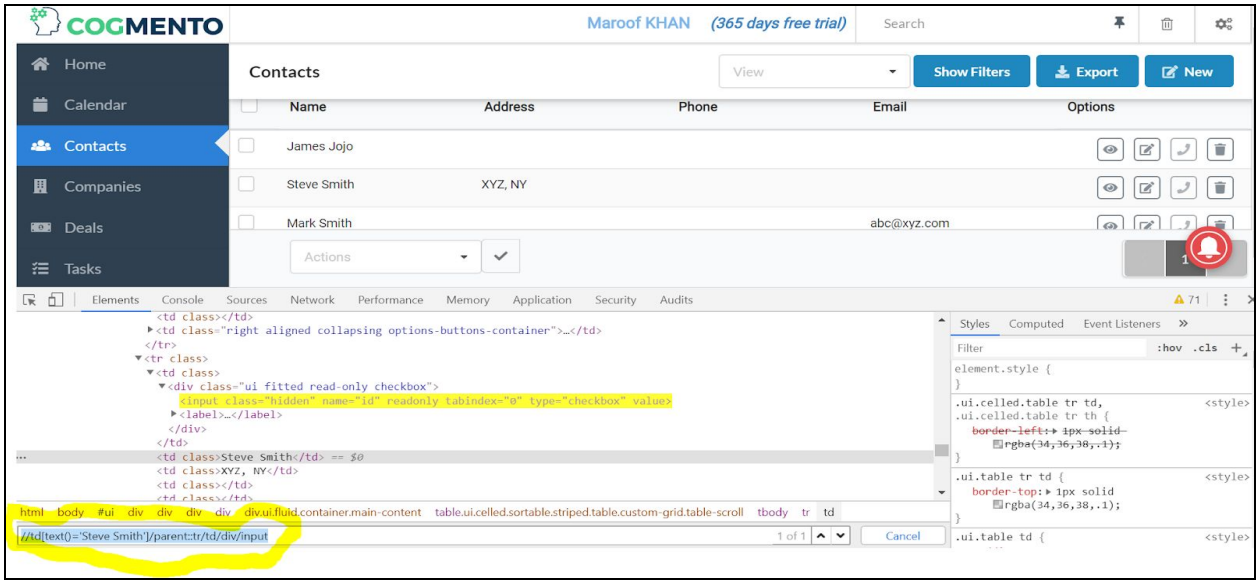
To demonstrate you preceding-sibling concept, I will simply find Forgot password element locator using Sign in Button's Xpath.



Do you remember that while beginning the discussion I told you the differences between Xpath and CSS Selector and I told you we can not traverse backward in CSS Selector. Yeah, so unfortunately, we do not have any preceding-sibling equivalent of Xpath in CSS Selector.

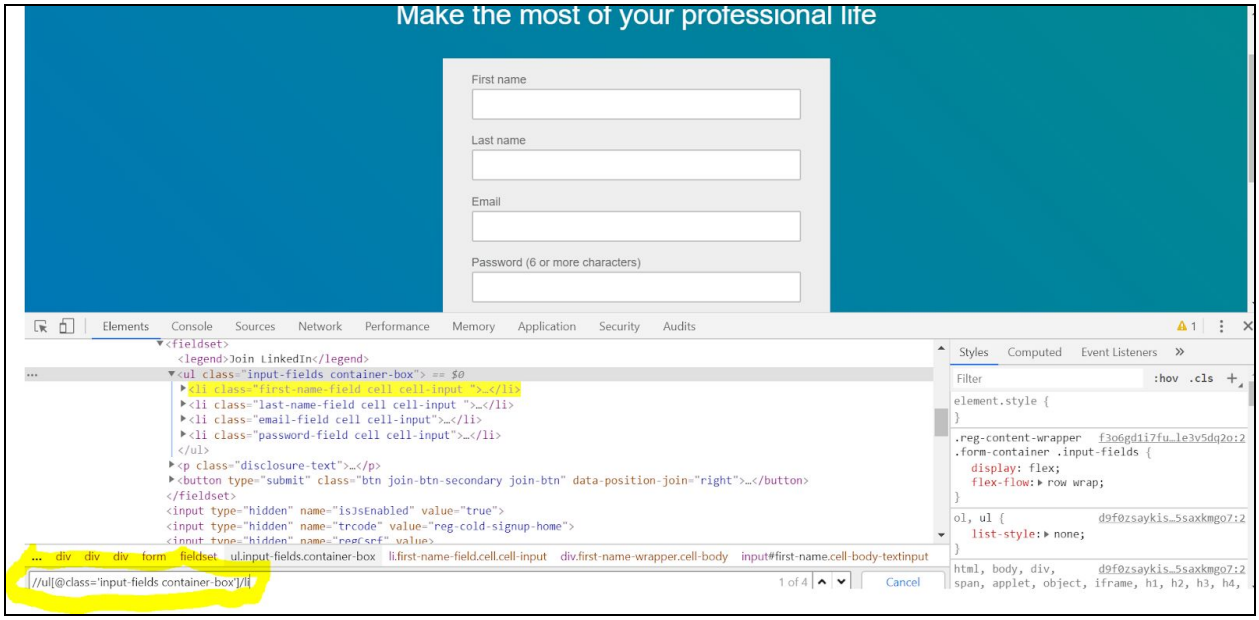
How to find Locators by Parent:

You can also traverse to parent node using Xpath. As mentioned above, this is not available in CSS Selector. This comes very handy when you have a table and you have to click on a checkbox corresponding to a user or based on some other criteria. For example, in the table mentioned in image, I want to select a contact named Steve and delete it. To do it, I must click on checkbox next to Steve. So I will first find out locator for Steve and then traverse to the checkbox as mentioned in the below image.

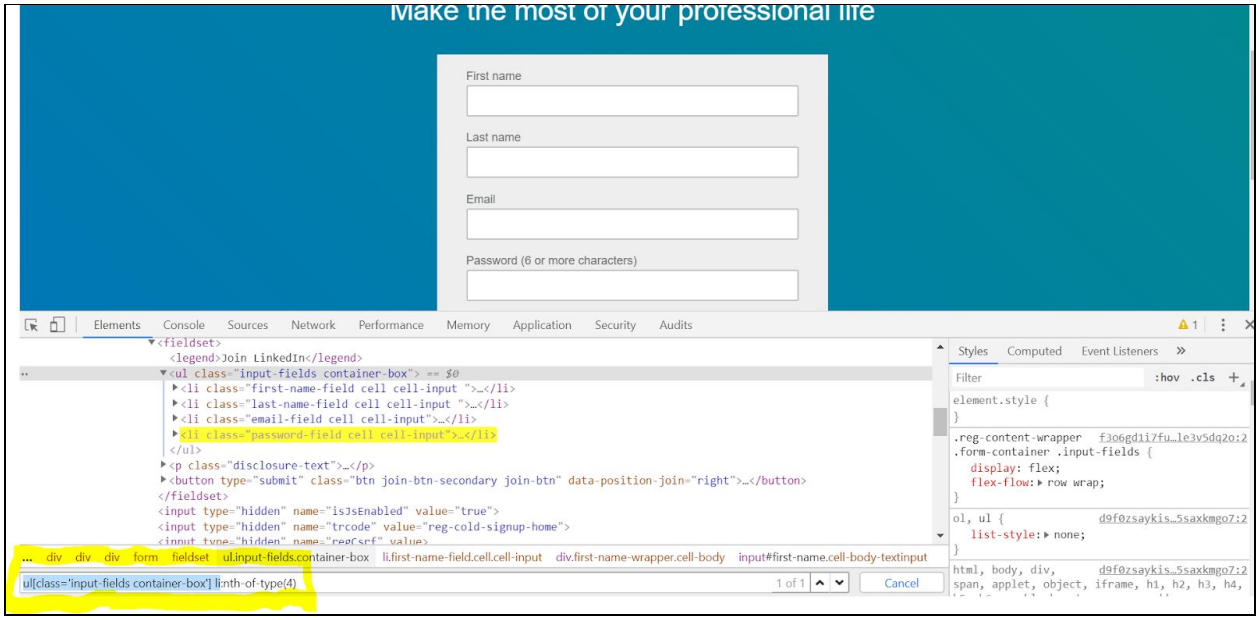


How to find a specific match:

Sometimes, you are not able to find an element uniquely and you get multiple matches. In Xpath you can use index to find specific match. For example if you on LinkedIn Sign Up page using this Xpath you will get 4 matches. Now if you want to select the second locator, you should use li[2]



In CSS you have nth-of-child() and nth-of-type() for indexing, so when in above scenario using CSS Selector *ul[class='input-fields container-box'] li* you will get 4 matches, you should use its indexing concept i.e. nth-of-child() or nth-of-type() as shown is below image:



Time for Some tidbits now:

1. *In Xpath you can also use text to find out a locator. For example, If a label has text like click me you can create your Xpath like* ***//label[text()='click me']*** *also you can use contains function with text like* ***//label[contains(text(),'click me')]*** *but this feature is not available in CSS Selector.*
2. *In Xpath you have a predicate like last() which is very useful while dealing with dynamic tables. For example, if you have to find last row of a table that has dynamic rows then you can simply use* ***//table[@name='Book Table']/tr[last()]*** *similarly for last cell of the table you can use td[last()] and you will navigate to the last cell.*
3. *In Xpath you can use * as a wildcard to ignore tag name so if you want to find all locators with attribute id whose value is username you can simply do* ***//*[@id='username']*** *while corresponding CSS Selector will be* ***[id='username'].***
4. *You can also pass a variable in Xpath. For example, in our scenario where we created a user and then we have to select checkbox next to the user to delete it. We can pass user's name or id from a variable so corresponding Xpath in our code will look like this*

//td[text()="'+your variable+'"]/parent::tr/td/div/input

5. *If same Xpath is giving you multiple matches you can wrap your Xpath and give it a index like* ***(//input[@id='username'])[1]***

P.S: You might not need to find locators manually in real time. You can use tools like ChroPath to find locators. You can also use a chrome add ins called Page Object Generator to extract all locators of an entire page in one go. But

remember, it is always good to have strong fundamental knowledge of locator strategy so that you can work in a situation when external tools or plugins are not available. For example, I was working for a banking customer and due to security issues, no third-party tool or website was allowed except Stackoverflow (for our luck😊) so in a situation like these, your knowledge of Xpath and CSS Selector will come handy.

So that's it for this article. I will be back with some knowledge sharing articles. Happy reading! 😊