

As a quality engineer, we all know that for testing a web-based application, we need to perform specific actions(*such as click, type, etc.*) on the *HTML* elements. Now, when we go for automation of those applications, the automation tool should also be capable of performing the same operations on the *HTML* elements that a human is capable of. So, now the question comes, how do these automation tools identify on which HTML element they need to perform a particular operation? The answer to this is "**Locators in Selenium**".

Locators are the way to identify an *HTML* element on a web page, and almost all UI automation tools provide the capability to use locators for the identification of *HTML* elements on a web page. Following the same trend, Selenium also possesses the ability to use "**Locators**" for the identification of HTML elements and are popularly known as "**Selenium Locators**". Selenium supports various kinds of locators. Let's understand the concept of "**Selenium Locators**" in depth by covering the details under the following topics:

What are locators in Selenium?

How to locate a web element in DOM?

What locators are supported by Selenium?

How to use locators to find web elements with Selenium?

How to locate elements in Selenium using By, ID, Name, Xpath, etc

Example showing usage of all the locators?

Best Practices For Using Locators In Selenium

What are Locators in Selenium?

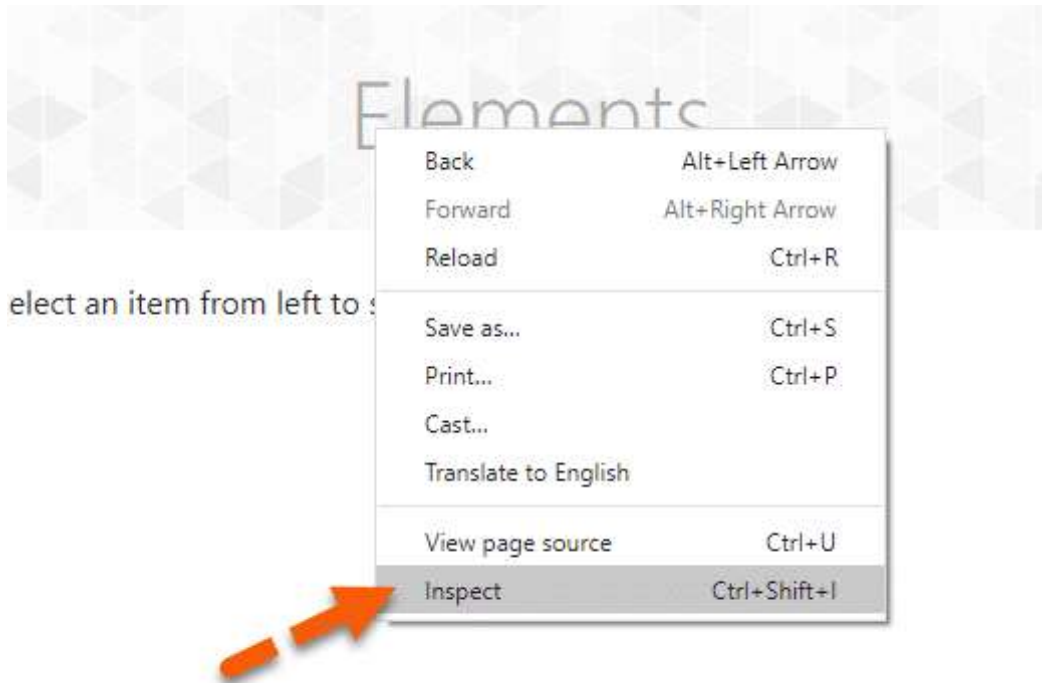
As discussed above, identification of the correct GUI element on a web page is pre-requisite for creating any successful automation script. It is where locators come into the picture. **Locators** are one of the essential components of Selenium infrastructure, which help Selenium scripts in **uniquely identifying the WebElements**(*such as text box, button, etc.*) present on the web page. So, how do we get the values of these locators? And how to use the same in the automation framework? Let's first understand on the whole page, how we can identify a specific web element in **DOM** and then we will try to grab its locator:

How to locate a web element in DOM?

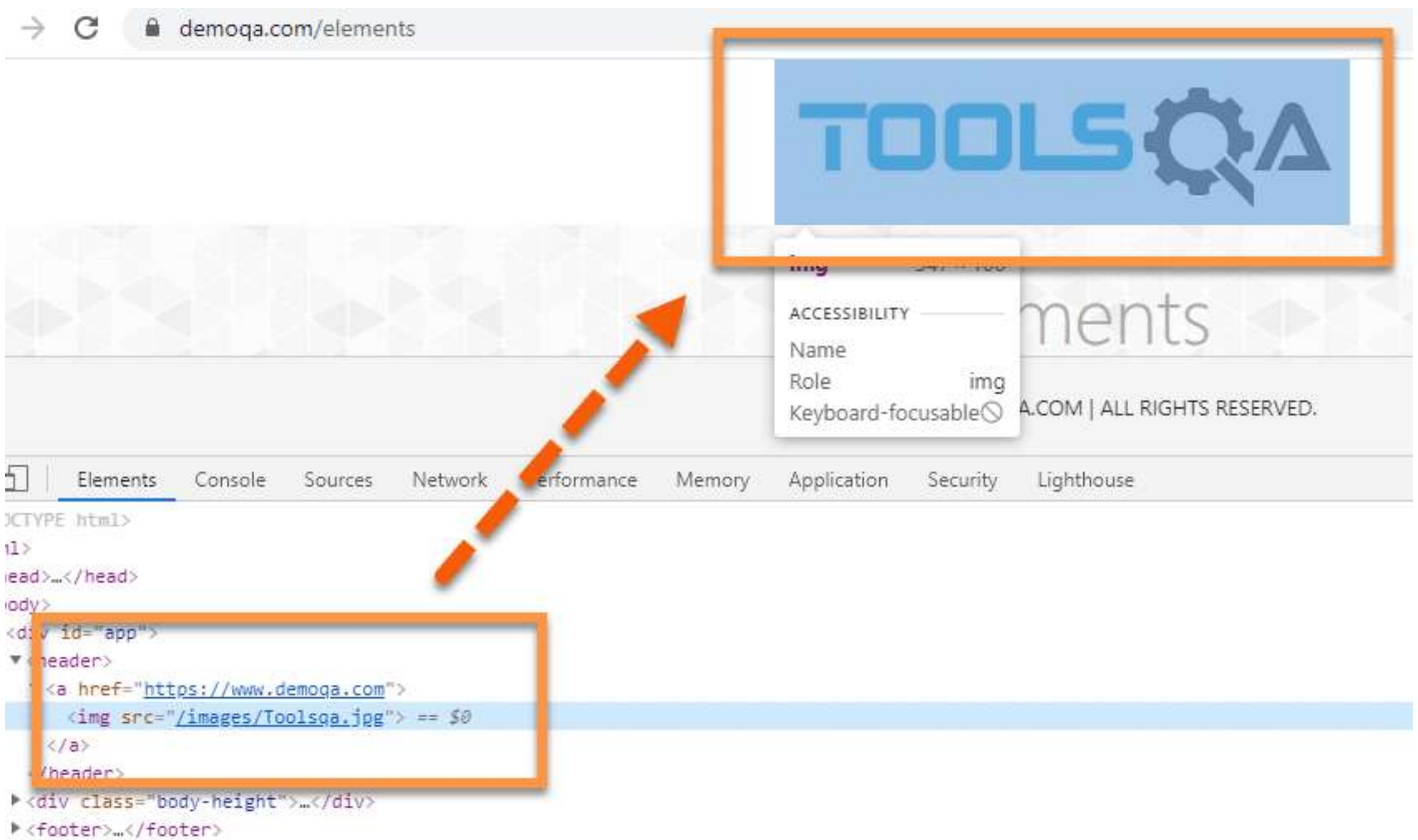
As we all know, the first thing to start with is to find the *HTML* element in **DOM** (*Document Object Model*), for which we need to grab the locator. We can follow the following steps to identify the web element in *DOM* on a web browser:

Note: We are showing the steps to identify the web element in **Google chrome**. The steps for other browsers will almost be the same, apart from the way how you can open the DOM in that browser. We are using the ToolsQa demo site "<https://demoqa.com/>" for the reference.

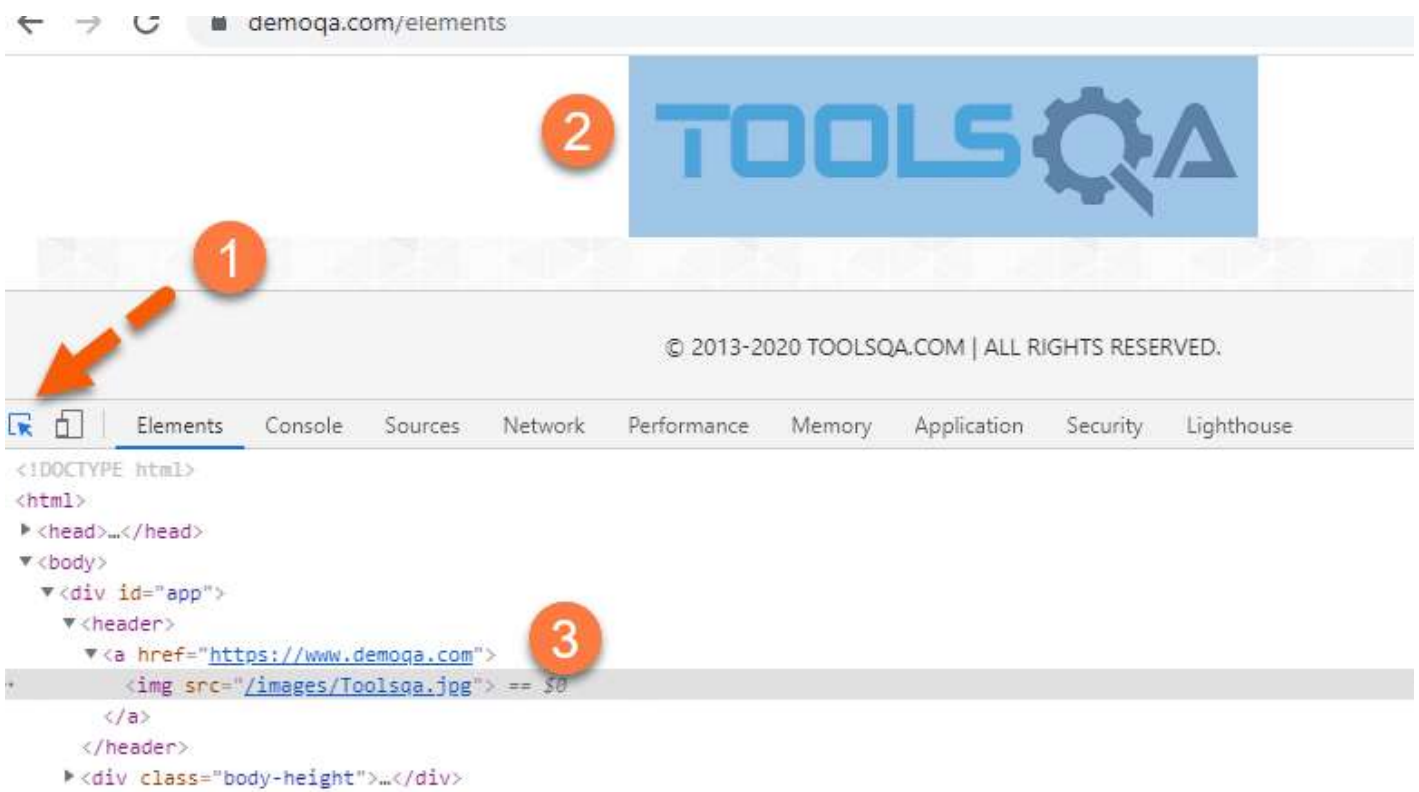
1. **DOM** can be accessed in Google Chrome either by pressing **F12** or by **right click** on the web page and then by selecting **Inspect** (as shown in the screenshot below).



. Once we click on the "**Inspect option**", it will open the **Developer Tools console**, as shown below. By default, it will open the "**Elements**" tab, which represents the complete **DOM** structure of the web page. Now, if we hover the mouse pointer over the **HTML** tags in the DOM, it will highlight the corresponding elements it represents on the webpage.



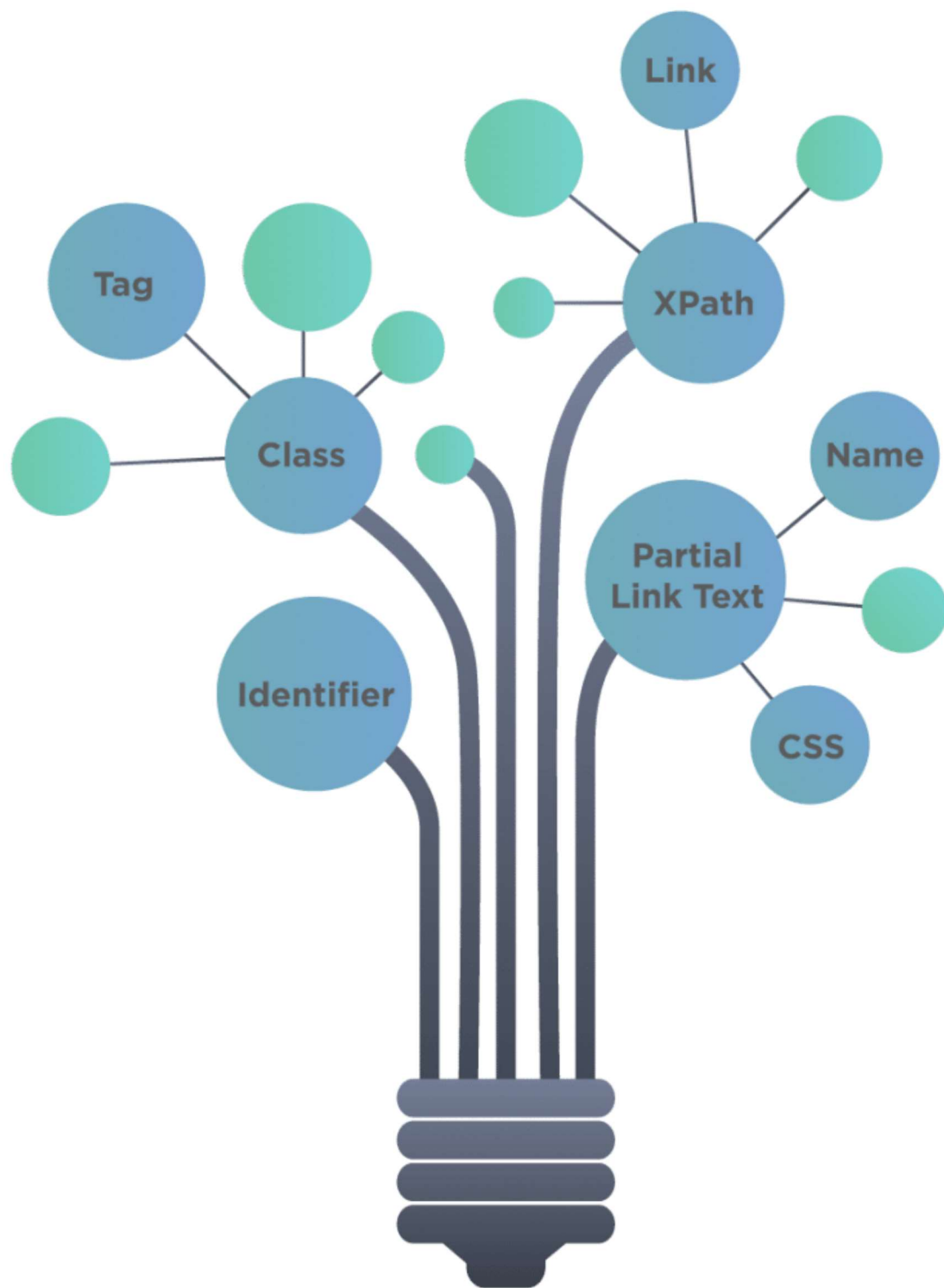
3. Now, the main point is, how do we find the web element in the **DOM**. Click on the "**Mouse Icon**" arrow (as designated by *Marker 1* in below screenshot) and then select the web element on the web page. It will automatically highlight the corresponding **HTML** element in the **DOM**. Suppose we want to find the **HTML** elements corresponding to the banner image (as shown below by *marker 3*). When we select the mouse point and click on the banner image, it will automatically highlight the corresponding HTML element, as shown by **marker 2**, in the below screenshot:



So, this way, we can easily search the HTML element in the DOM corresponding to a web element on the webpage. Now, let's try to understand how we can grab a locator for these web elements, which can be directly used by Selenium to identify these web elements uniquely:

What locators are supported by Selenium?

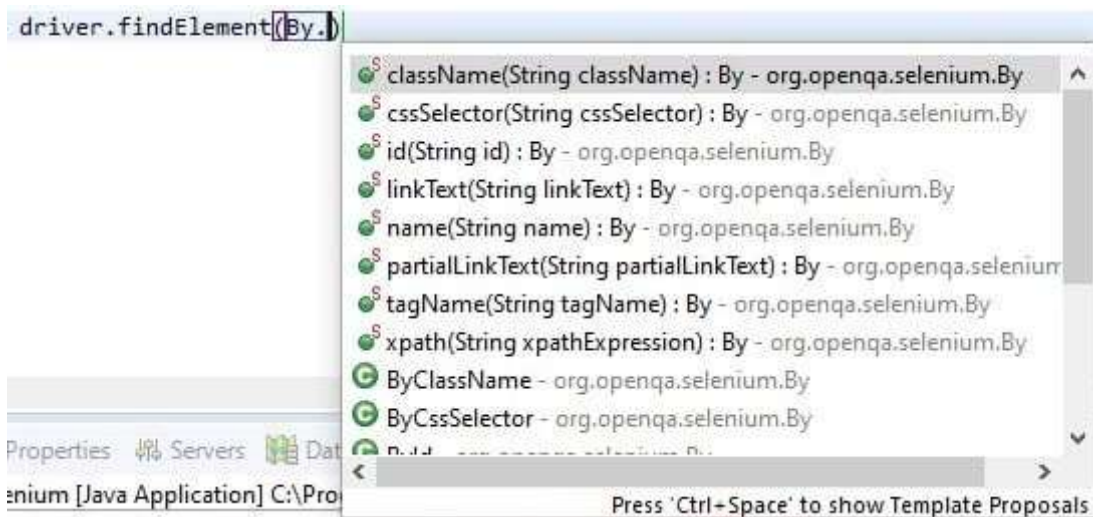
There are various types of locators, using which we can identify a web element uniquely on the Webpage. The following figure shows a good depiction of several types of locators that Selenium supports.



Selenium Locators

To access all these locators, Selenium provides the “**By**” class, which helps in locating elements within the DOM. It offers several different methods (*some of which are in the image below*) like ***className***, ***cssSelector***, ***id***, ***linkText***, ***name***, ***partialLinkText***, ***taName***, and ***xpath***, etc., which can identify the web elements based on their corresponding locator strategies.

You can quickly identify all the supported locators by Selenium by browsing all the visible methods on the "**By**" class, as shown below:



As we can see, Selenium supports the following locators:

ClassName – A ClassName operator uses a class attribute to identify an object.

cssSelector – CSS is used to create style rules for webpages and can be used to identify any web element.

Id – Similar to class, we can also identify elements by using the 'id' attribute.

linkText – Text used in hyperlinks can also locate element

name – Name attribute can also identify an element

partialLinkText – Part of the text in the link can also identify an element

tagName – We can also use a tag to locate elements

xpath – Xpath is the language used to query the XML document. The same can uniquely identify the web element on any page.

Now, let's understand the usage of all these types locators in the Selenium framework:

How to use locators to find web elements with Selenium?

As we mentioned above, Selenium supports various types of locators. Let's under the details and usage of all of them:

How to locate a web element by using the "id" attribute?

"ID" as a locator is one of the most common ways of identifying elements on a web page.

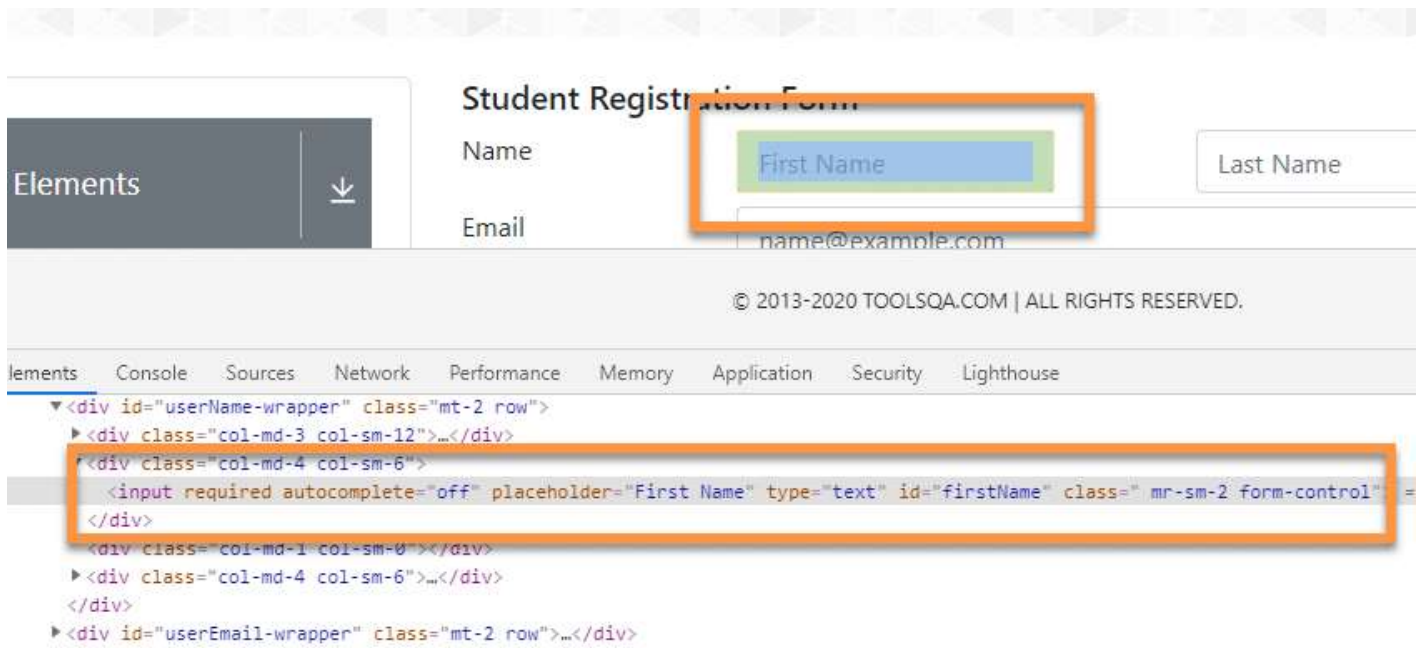
According to **W3C**, an ID for a web element always needs to be unique. The **ID** is one of the fastest and unique ways of locating web elements and is considered as one of the most reliable methods for determining an element. But with the advancement in technologies and more of the **dynamic**

web pages, "**IDs**" are generated dynamically and generally not the reliable way to locate a web element, as they change for different users.

Let's understand the usage of the "**id**" attribute for locating a web element with the help of an example.

Suppose we have to locate the text box of First Name on the web page

"<https://demoqa.com/automation-practice-form>". When we inspect it in *DOM*, we see the following *DOM* structure:



As we can see, the "**Input**" HTML tag has the following properties and attributes:

```
<input required="" autocomplete="off" placeholder="First Name" type="text" id="firstName"
```

As we can see, the HTML tag contains the attribute "**id**" inside the input tag. The **id** here used is the "**firstName**" which we can use to locate this element in the web page. Now, to find the "**First Name**" text box on the web page, we can use the following syntax:

```
By.id("firstName");
```

As we can see that we have used the "**By.id()**" method, and we have passed the "**firstName**", which is the "**id**" of the text box we are trying to locate. So, this way, we can locate any web element which has a specified "**id**" attribute associated with it.

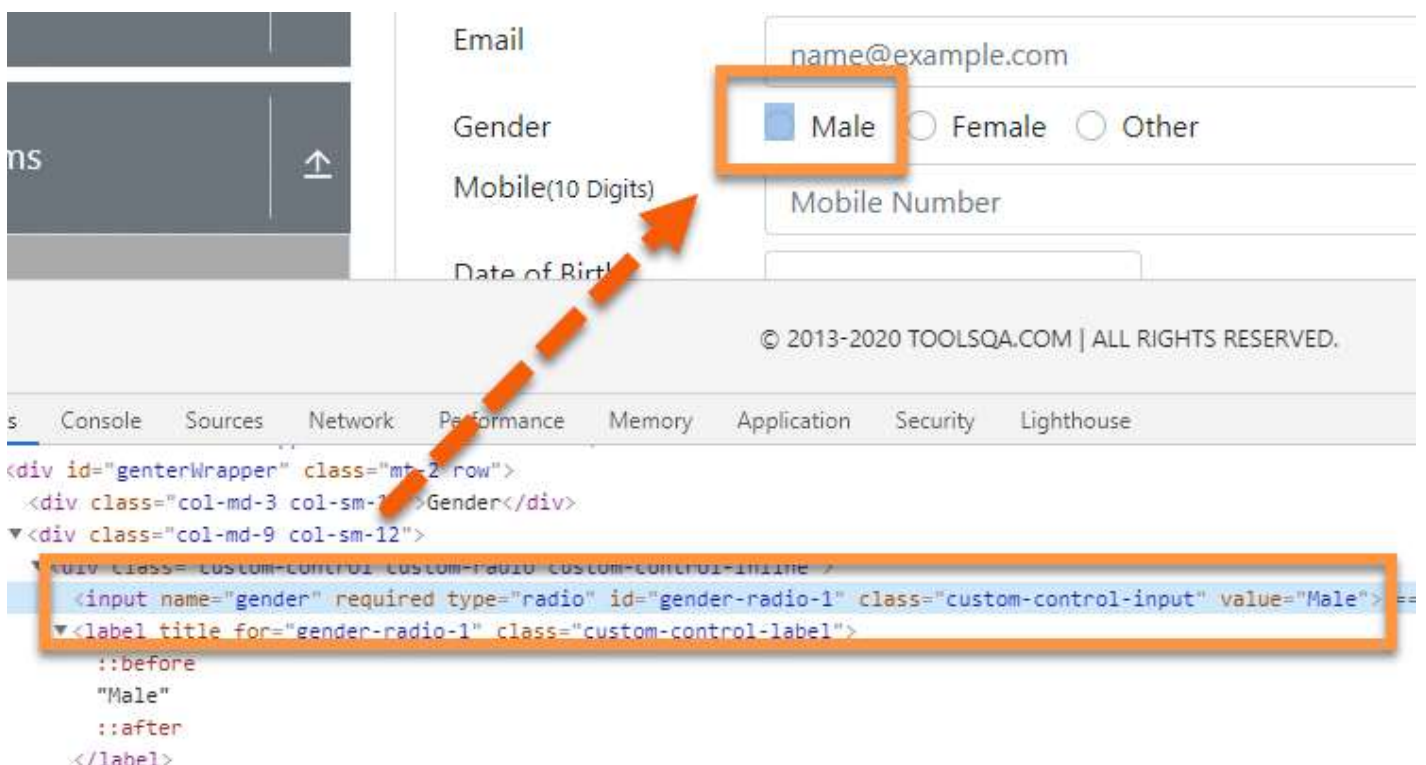
How to locate a web element by using the "name" attribute?

Similar to the **id** attribute, Selenium also offers users a way to identify the element using the **name** attribute. But contrary to id, a web page can have multiple elements with the same "**name**" attribute. So, if we are going to use the name attribute for the identification of a web element, we should always make sure that the name attribute must contain a unique value. Otherwise, it may identify several different elements on the same page, which may have the same name value.

If multiple elements have the same value of the '**name**' attribute, then, Selenium will just select the first values in the page which matches the search criteria. So, we always recommend making sure that the value of the **name** attribute should be unique when we select it for locating a web element.

Let's understand the usage of the "**name**" attribute for locating a web element with the help of an example.

For example, suppose we need to locate the checkbox in the gender section on the web page "<https://demoqa.com/automation-practice-form>", as highlighted below:



The screenshot displays a web form with fields for Email, Gender, Mobile, and Date of Birth. The 'Male' radio button in the Gender section is highlighted with an orange box. A red dashed arrow points from this box to the corresponding HTML code in the developer tools. The HTML code shows an `<input>` tag with `name="gender"`, `required`, `type="radio"`, `id="gender-radio-1"`, and `class="custom-control-input"`, followed by a `<label>` tag with `title for="gender-radio-1"` and `class="custom-control-label"`.

As we can see, the "**Input**" HTML tag representing the *Male* Checkbox has the following properties and attributes:

```
<input name="gender" required="" type="radio" id="gender-radio-1" class="custom-control-i
```

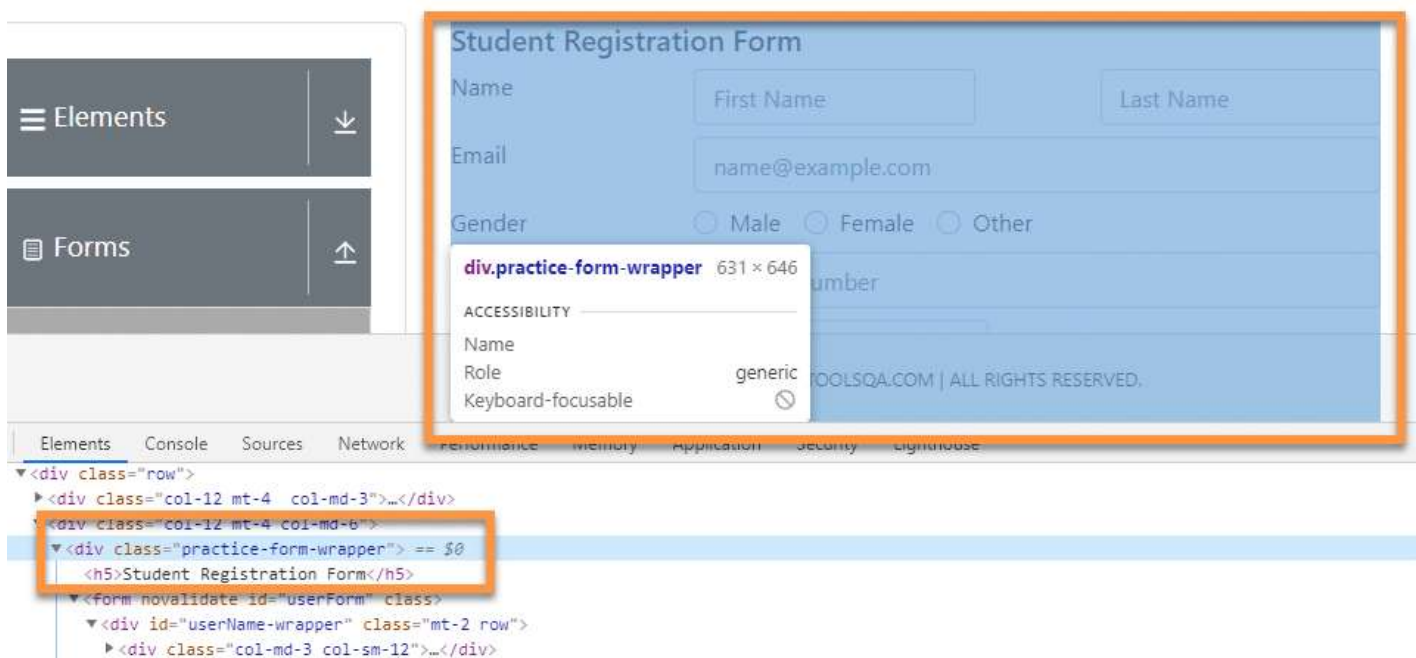

We can see the attribute 'name' with value as 'gender'. We can use the following syntax to locate the web element using the **"By"** class.

```
By.name("gender");
```

As we can see, the **"By"** class provides the **"name"** method, which accepts the value of the **"name"** attribute of the web element. So, using this, we can locate a web element that has a unique **name** attribute.

How to locate a web element by using the "ClassName" attribute?

ClassName allows Selenium to find web elements based on the **class** values of the DOM. For example, if we want to identify or perform any operation on the form element, we can use the class to identify it.



If we look at the *DOM* structure of the form, we can see the following snippet enclosing the entire form.

```
<div class="practice-form-wrapper">
```

We can use the **class** attribute value from the above DOM to identify the form. To identify the same on the webpage, we can use the following syntax.

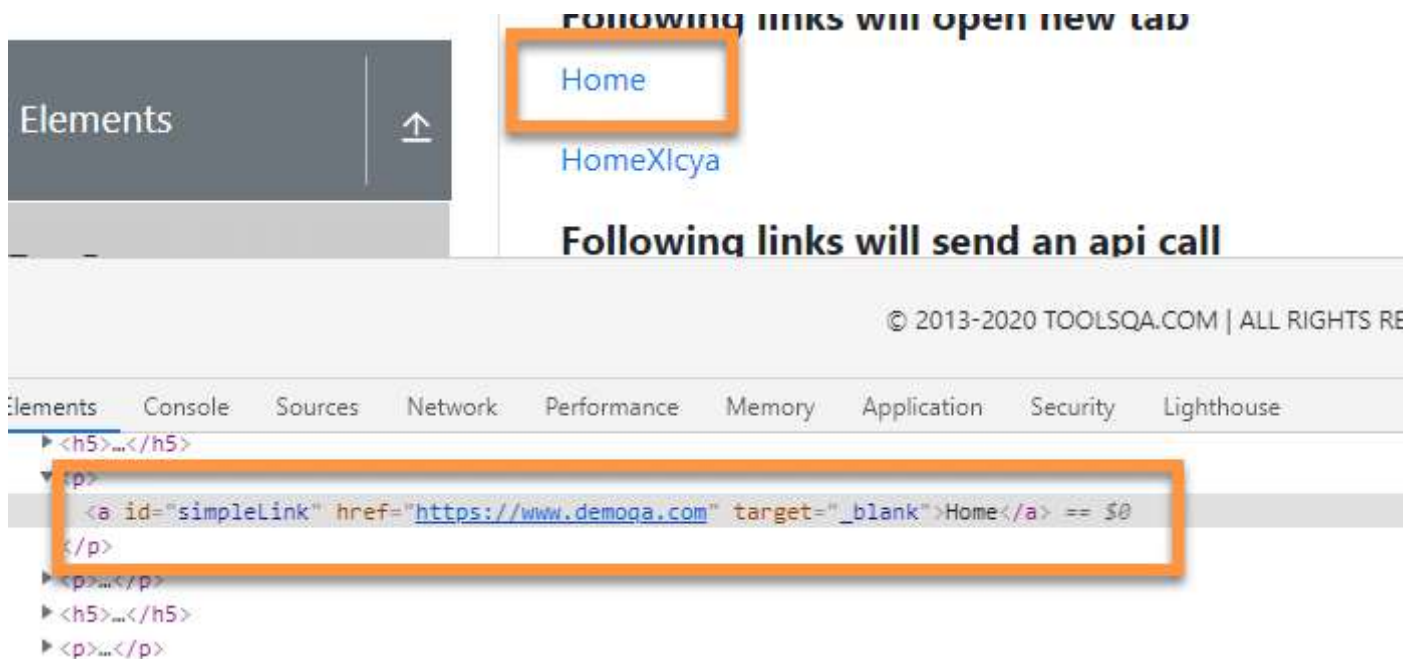
```
By.className("practice-form-wrapper");
```

To identify it successfully, we need to make sure that the **class name** value that we are using for locating the web element, i.e., web form, is unique, and any other class doesn't have the same value. If any other class contains the same value as this, then Selenium will select whichever element comes first while scrapping through the web page.

How to locate a web element by using the "LinkText" and "partialLinkText" attribute?

LinkText and **partialLinkText** both are quite similar in functionality and locate web elements by using the hyperlink texts. We can only use them for elements containing **anchor(<a>)** tags. Similar to other locator strategies, if multiple hyperlinks with the same texts are present on the page, then Selenium will always choose the first one.

Suppose we need to click on the link as shown below for the "**Home**" link:



As we can see, the anchor element has the following properties and attributes:

```
<a id="simpleLink" href="https://www.demoqa.com" target="_blank">Home</a>
```

For identifying elements using **link text** or **partial link text**, we need to use the hyperlink text, as shown below:

```
By.linkText("Home");
```

Similarly, partial link text can also recognize the element by using ***part of the hyperlink text***, as shown below:

```
By.partialLinkText("Ho");
```

It will identify any link that contains 'Ho' in the hyperlink text. Likewise, if several links are containing "***Ho***", then Selenium will select the first one.

How to locate a web element by using the "TagName"?

This locator type uses ***tag names*** to identify elements in a web page. The ***tag name*** is the *HTML* tag, such as `***input`, `div`, `anchor tag`, `button`, etc.

The following syntax finds the web elements with a ***tagName***:

```
By.tagName("a");
```

The ***tagName*** locator returns all the elements from the page that contains a specified tag.

How to locate a web element by using the "CSSselector"?

CSS or Cascading style sheets are used extensively to style webpages and hence can be an effective medium to locate various web elements. These days most of the web pages are dynamically designed. Thus its quite challenging to get a unique id, name, or class to locate element. In this type of situation, ***CSS selectors*** can be a great alternative as these are way faster compared to another locator strategies.

The basic syntax of identifying a web element using ***CSS*** is as follows:

```
css=(HTML Page)[Attribute=Value]
```

For example, let's say we want to find the following input textbox using CSS selector.



As we can see, the input element has the following properties and attributes:

```
<input autocomplete="off" placeholder="Full Name" type="text" id="userName" class="mr-sm-2 form-control">
```

Now, to find the element using the CSS selector, we have to use the following syntax:

```
By.cssSelector("input[id= 'userName']");
```

Similarly, we can use other attributes along with the tags to define different CSS locators. We will discuss more details about the CSS selector in another tutorial.

How to locate a web element by using the "xpath" attribute?

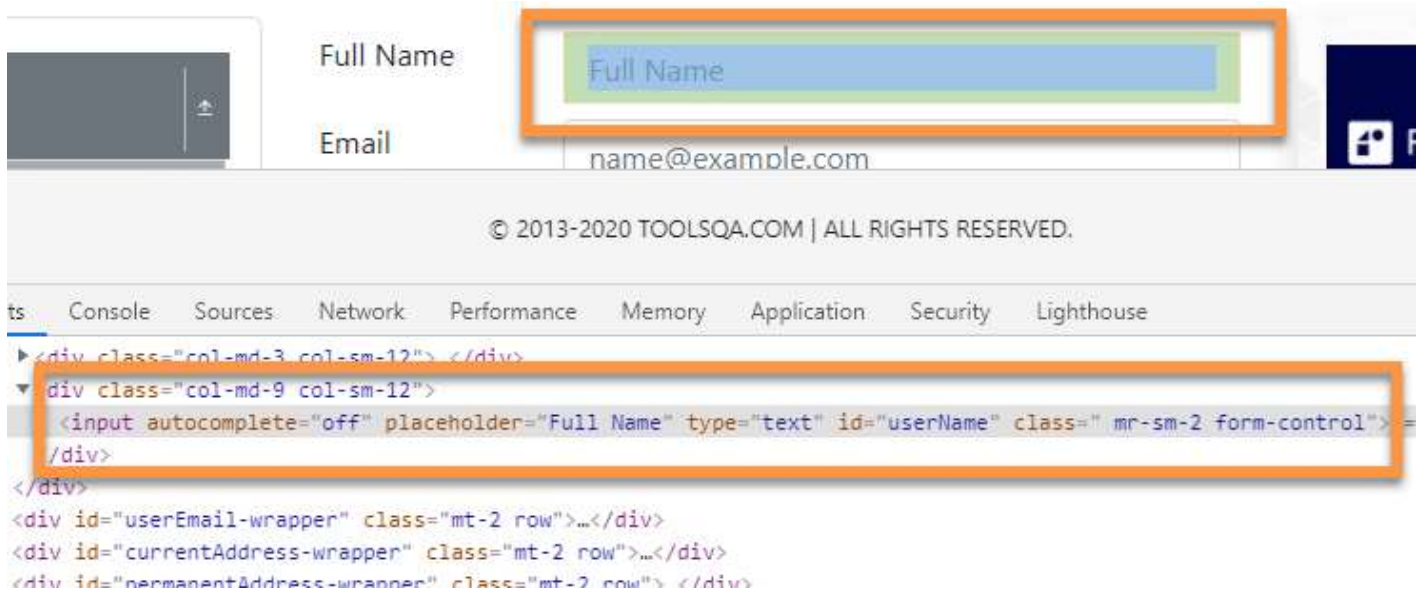
XPath uses the **XML** expression to locate an element on the webpage. Similar to CSS selectors, Xpath is quite useful in locating dynamic elements on a webpage. Xpath can access any element present in the webpage even when they have dynamic properties.

The basic syntax of identifying a web element using the XPath locator strategy is as follows:

```
//tag_name[@attribute_value]
```

Where the **tag_name** is the name of the tag in the **DOM structure**, and the attribute is an attribute of the target element that can identify the web element uniquely.

For example, let's say we want to find the following input textbox using **XPath**:



The XPath for the above example will be:

```
//input[@id='userName']
```

So, sample code to find the element using XPath will be the following:

```
By.xpath("//input[@id='userName']");
```

Here we have used the input **tag and id attribute** to identify the element. We will cover the more in-depth details of the "**XPath locator**" strategy in another article.

Let's now see the usage of these locators' strategies to identify various web elements.

Example showing usage of all the locator's strategies in Selenium:

Suppose we want to automate specific actions on the web page: on the "<https://demoqa.com/automation-practice-form>". The following code snippet shows the usage of various types of Selenium locators to identify and searches the elements on the web page:

```

package TestPackage;

import java.util.List;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;

public class Selenium {

    public static void main(String[] args) {

        String exePath = "C:\\Selenium\\chromedriver\\chromedriver.exe";
        System.setProperty("webdriver.chrome.driver", exePath);
        WebDriver driver = new ChromeDriver();
        driver.get("https:\\demoqa.com\\");

        /**Locate by ID Attribute
         * URL - https://demoqa.com/automation-practice-form
         */

        driver.get("https://demoqa.com/automation-practice-form");
        driver.findElement(By.id("firstName"));

        /**
         * Locate by Name attribute
         * URL - https://demoqa.com/automation-practice-form
         */

        driver.get("https://demoqa.com/automation-practice-form");
        driver.findElement(By.name("gender"));

        /**
         * Locate by className attribute
         * URL - https://demoqa.com/automation-practice-form
         */

        driver.get("https://demoqa.com/automation-practice-form");
        driver.findElement(By.className("practice-form-wrapper"));

        /**
         * Locate by linkText and ParticalLinkText attribute
         * URL - https://demoqa.com/links
         */

        driver.get("https://demoqa.com/links");
        //linkText
        driver.findElement(By.linkText("Home"));
        //partialLinkText
        driver.findElement(By.partialLinkText("Ho"));

        /**
         * Locate by tagName attribute
         * URL - https://demoqa.com/links
         */

        driver.get("https://demoqa.com/links");
        List <WebElement> list = driver.findElements(By.tagName("a"));
    }
}

```



```

    /**
     * Locate by cssSelector attribute
     * URL - https://demoqa.com/text-box
     */

    driver.get("https://demoqa.com/text-box");
    driver.findElement(By.cssSelector("input[id= 'userName']"));

    /**
     * Locate by xpath attribute
     * URL - https://demoqa.com/text-box
     */

    driver.get("https://demoqa.com/text-box");
    driver.findElement(By.xpath("//input[@id='userName']"));

}

}

```

Now, the interaction with any web application requires the **Selenium driver** to identify **web elements** on the page. Until and unless an element gets correctly identified, it's impossible to trigger **events** like **click**, **enter**, **etc.** on any web element. Now to search the web elements, Selenium provides two methods:

findElement: find and return a single web element based on the search criteria of the locator.
findElements: find and return all the web elements based on the search criteria of the locator.

Both of these methods accept the object of the "**By**" class and then return the corresponding web element. We have used the **findElement** of the **Selenium WebDriver** to search the various web elements and have passed different types of locators by invoking the various method of the "**By**" class.

Best Practices For Selenium Locators

Choosing the correct locator for recognizing a web element is quite essential in Selenium. Listed below are some of the best practices that a quality engineer needs to follow to make efficient use of locators in the **Selenium WebDriver based automation framework**.

*Do not use **dynamic attribute** values to locate elements, as they may change frequently and result in breakage of locator script. It also severely affects the **maintainability**, **reliability**, and **efficiency** of the automation script.*

ID and **name** attributes take precedence over other locators if your web page contains unique **ID** and **name**, then it's always advisable to use them instead of **XPath** as they are faster and more efficient.

While using **locators**, make sure that your locator points precisely to the required element. If the needed scenario needs to perform some operation on a single element, then make sure that your locator exactly matches to only one element. If the locator points to several different elements, then it may cause breakage in your script.

Never use locators to locate auto-generated elements on the web page. Sometimes in a dynamic web environment, element attribute properties are generated at run time. One should avoid these elements as they may cause breakage during script execution.

While working with **XPath** or **CSS locators**, one should avoid directly using the one generated by the **Chrome Dev Tools**. It may seem one of the easiest ways to generate **XPath**, but in the long run, it induces reliability issues, code breakage, maintainability issues, etc. It may look tempting to use these, but you would be better off creating your customized **XPath** in the longer run.

Key Takeaways

Selenium supports 8 different types of locators namely `id`, `name`, `className`, `tagName`, `linkText`, `partialLinkText`, `CSS selector` and `xpath`.

Using `id` is one of the most reliable and fast methods of element recognition. Usually, the `id` is always unique on a given web page.

`CSS selector` and `XPath` can identify dynamic elements on a web page. A combination of different attributes and tag names can be used with `CSS` and `xpath` to identify any given element.