**LOCATORS**

**What is it -**

* Locator is the foremost feature in Selenium to identify the objects / web elements which are present on the web page. There are multiple locators available in Selenium with which we can make use of in selenium scripts

**Why is it used -**

* Every web page has elements with which you need to interact to perform any kind of operations such as clicking, entering text, hovering over and so on
* Without identifying the element on the page, we cannot perform any operation on it through selenium. Hence, identifying element is the fundamental step to act upon the element which is why we are using locators (“By” class in Selenium) to control the elements

**Types of Locators -**

1. ID
2. Name
3. Class Name
4. Tag Name
5. Link Text
6. Partial Link Text
7. XPath
8. CSS Selector

**How to use Locators–**

**Step 1:** Inspect the target element in the HTML DOM of the web page using in-built inspector feature available in browser(Example in chrome browser: Right click anywhere on window and click on “Inspect” or use keyboard shortcut Ctrl + Shift + I to open elements inspector window)

**Step 2:** Make use of any locators in selenium scripts to identify the element using the differentmethods available in the “By” class:**org.openqa.selenium.By**. You can find the syntax below.

Syntax:

**driver.find\_element\_by\_name("Ban")**

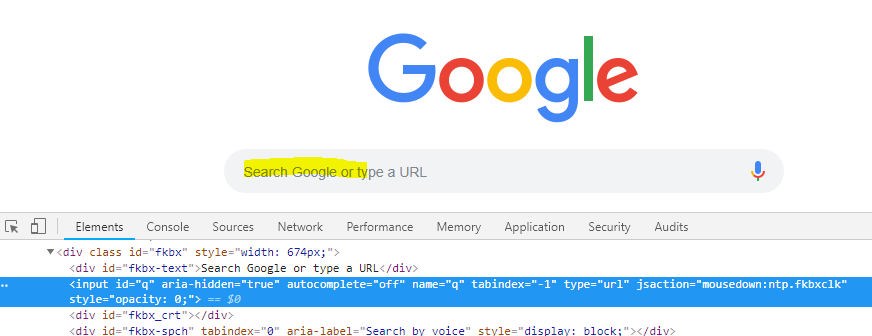
* where driver is the object of WebDriver
* where findElement is the method of WebDriver
* where “By” is the static class which is used for identifying locators
* where “locatorName” is where we need to mention any one of the locators (method name of the locatorpresent in By class) out of the 8 locators
* where xxx denotes the actual value of the locator which you find in inspector using step 1
* where “click()” is the actual operation we need to perform on the element

**1) Using ID:**

**driver.find\_element\_by\_id("Ban").click()**

* where “id” is the method name to find the element using ID
* where yyy denotes the value of the attribute “id” in inspector window

**Example:** When you inspect google search box, we can see that the value of “id” is “q”. So we can use **driver.find\_element\_by\_id(“q”).click()** to identify the search box and to click on it



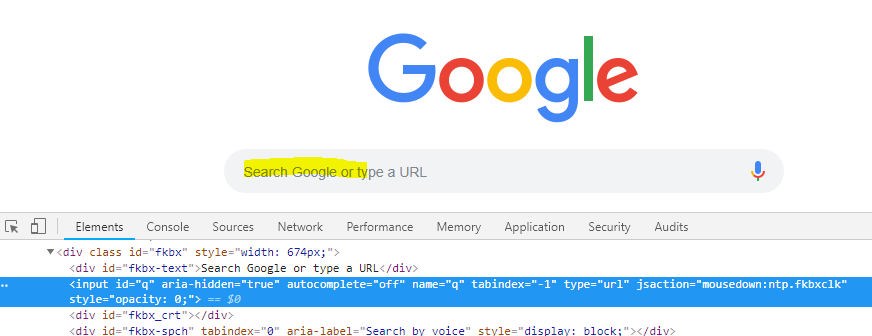
**Limitation**: IDs are quick way to find the element however it may not be present for all elements and could be dynamic also. In such cases, go ahead by using other locators and avoid the usage of ID

2) **Using Name:**

driver. .find\_element\_by\_name (“yyy”).click()

* where “name” is the method name to find the element using name
* where yyy denotes the value of the attribute “name” in inspector window

**Example:** When you inspect google search box, we can see that the value of “name” is “q”. So we can use **driver.findElement(By.name(“q”).click()** to identify the search box and to click on it



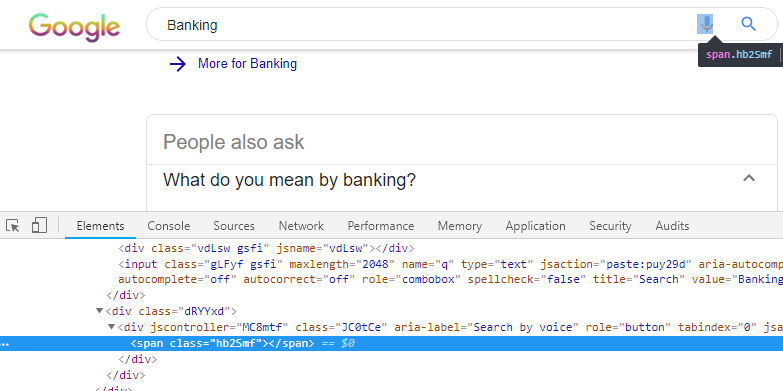
**Limitation**: Name may not be always present for all elements. Use other locators in such situations

3) **Using Class Name:**

driver .find\_element\_by\_class\_name (“yyy”).click()

* where “className” is the method name to find the element using class
* where yyy denotes the value of the attribute “class” in inspector window

**Example:** When you inspect voice icon in google search box, we can see that the value of “class” is “hb2Smf”. So we can use **driver.findElement(By.className(“hb2Smf”).click()** to identify the voice icon and to click on it



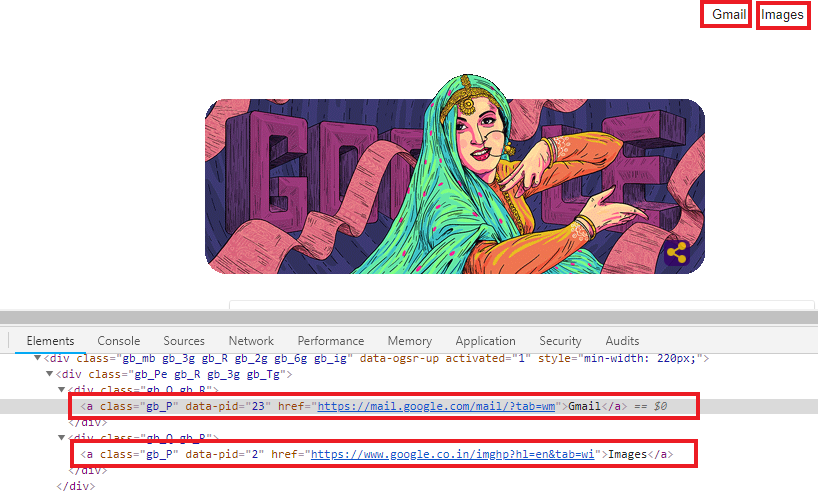
**Limitation**: We cannot use this locator when the class name has spaces in between

4) **Using Tag Name:**

driver. find\_element\_by\_tag\_name (“y”).click()

* where “tagName” is the method name to find the element using html tag
* where y denotes the html tag name which you can find in inspector window

**Example:** When we inspect links such as ‘Gmail’ and ‘Images’ in google homepage, we can see that it’s pointing to the anchor tags <a> in HTML DOM.We can make use of **driver.findElements(By.tagName(“a”).size()** to find the number of links present on the page. Note that we are using findElements instead of findElement as our target is not on single element (we are targeting more than one link)



5) **Using Link Text:**

driver.find\_element\_by\_link\_text(“yyy”).click()

* where “linkText” is the method name to find the element using the text of the link
* where yyy denotes the visible text of the link on the page

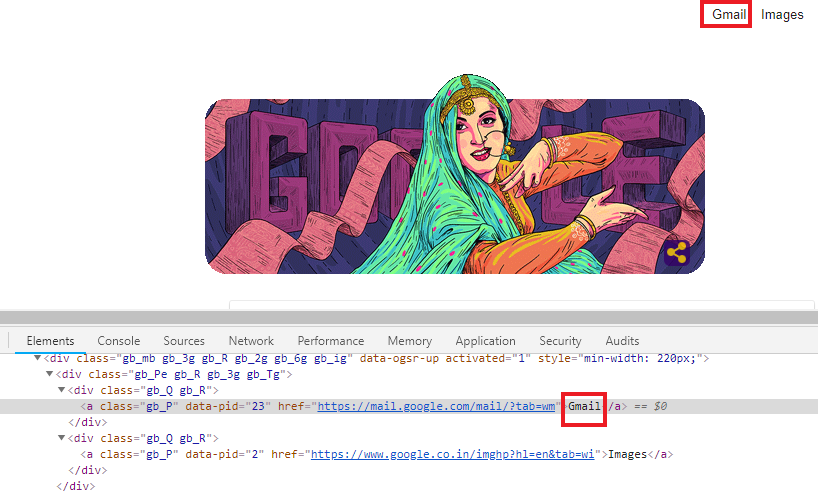
**Example:** If you want to click on link ‘Gmail’, we can use the actual visible text (i.e Gmail) of the link. So we can use **driver.findElement(By.linkText(“Gmail”).click()** to identify the link and to click on it.

6) **Using Partial Link Text:**

driver.find\_element\_by\_partial\_link\_text(“yyy”).click();

* where “partialLinkText” is the method name to find the element using the subtext of the link
* where yyy denotes the visible subtext of the link on the page

**Example:** If you want to click on link ‘Gmail’ using partial link text locator, we can use the sub text (i.e mail or mai etc.) of the link. So we can use **driver.findElement(By.partialLinkText(“mail”).click()** to identify the link and to click on it.



**Limitation**: Make sure the subtext is not the subtext of any other links on the same page

7) **Using XPath:**

XPath is a technique which is used for navigating between nodes in XML.The same technique can be used for HTML also as HTML is alike XML document (xHTML). Using XPath, we will target the node by traversing through HTML nodes.

driver.**find\_element\_by\_xpath**.(“yyy”).click()

* where “XPath” is the method name to find the element using XPath
* where yyy denotes the XPath (see the format below)

There are two types of Xpath: Absolute XPath, Relative XPath

**Absolute XPath:** Traversal starts from the root node(i.e.<html>) and ends at the node of the target element

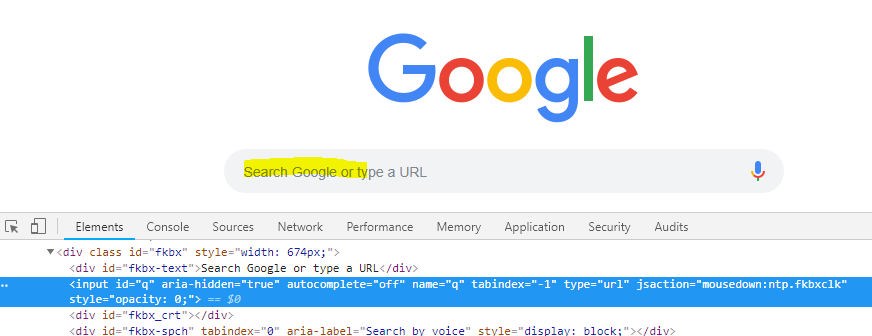
Format: /html/……/targetnode

Example: /html/body/div[1]/div[2]/div[1]/a

**Relative XPath:** Traversal starts from anywherefrom the html tree and ends at the node of the target element or it can also directly point to the target element.

Format: //tagname[@attribute=‘value’]

Example: //input[@id=‘q’] (using google search box scenario mentioned below) or you can also mention as //\*[@id=’q’] where \*denotes any node/tag



So, **driver.findElement(By.XPath(“//input[@id=‘q’]”)).click();** will identify the search box and click on it

**Tip**: You can also use pre-defined XPath which is automatically created by the browser (refer to the below screenshot) however user-defined XPath are the most reliable one and most recommended to use as we cannot be sure how long the pre-defined feature will be offered by the browser



8) **Using CSS Selector:**

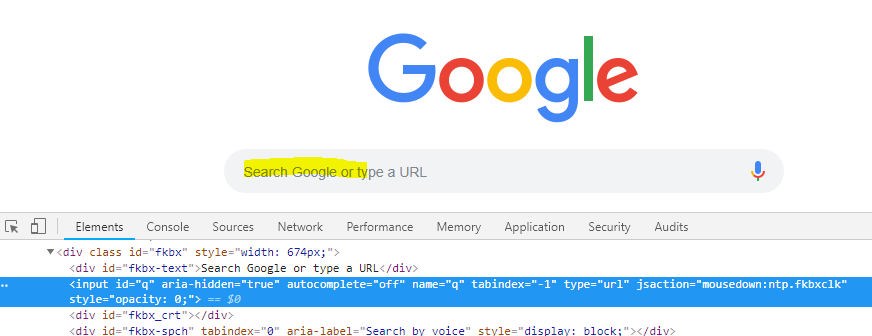
CSSSelector is also used for identifying the element using navigation technique like XPath.Selector is also constructed using HTML tags, attributes and their values.The concept lies the same however the only difference between both is based on the construction protocol

driver. *find\_elements\_by\_css\_selector.* (“yyy”).click()

* where “cssSelector” is the method name to find the element using CSS
* where yyy denotes the CSS Selector pattern (see the format below)

Format:tagname[attribute=‘value’]

Example: input[id=’q’] (using google search box scenario mentioned below)

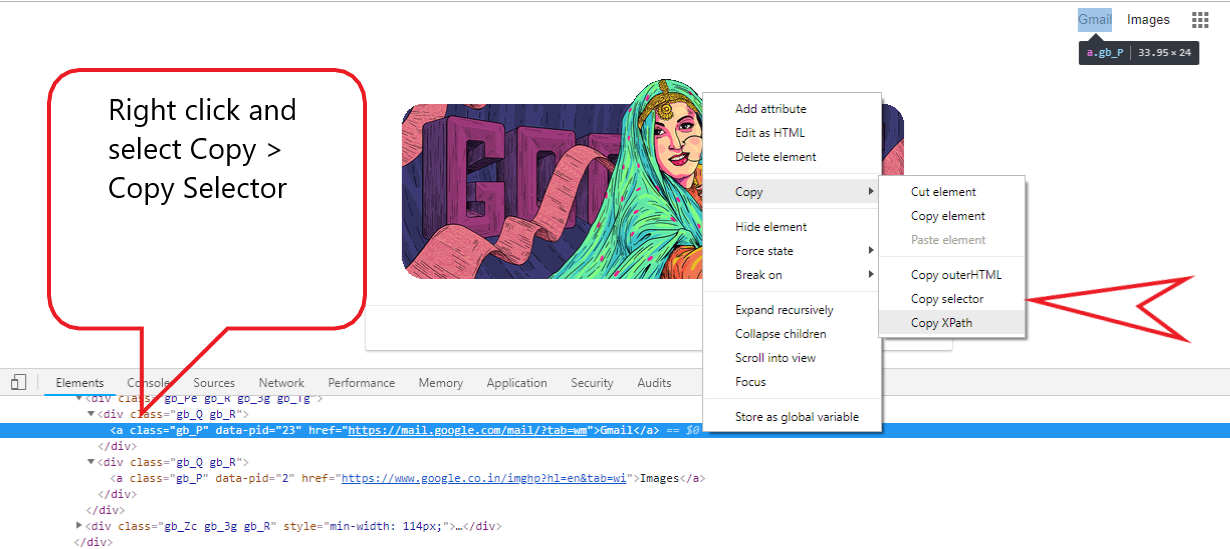


So, **driver.findElement(By.cssSelector(“input[id=‘q’]”)).click();** will identify the search box and click on it

Using ID and Class names, CSS can be created quickly by following the format below

* Format using ID: tagname # id => Ex: input # q
* Format using Class: tagname. Classname => Ex: span.hb2Smf

**Tip**: You can also use pre-defined selector which is automatically created by the browser (refer to the below screenshot) however user-defined selectoris the most reliable one and most recommended to use as we cannot be sure how long the pre-defined feature will be offered by the browser



**XPath & CSS - Derivation techniques - User Defined Approach:**

There are many ways to create XPath and CSS which is mentioned below

**Parent - Child traversal relationship traversal:**

* This can be used by navigating between parent and child nodes until we find the target node
* Tip: Use the unique/stable node as the pointer and navigate till you hit the target node
  + XPath: //\*[@attribute=‘value’]/……/…/ targetnode
  + CSS: tagname[attribute=‘value’] > …. > …… > targetnode

**Sub Text:**

* Sub text of the attribute’s value can be used to create XPath / CSS
  + XPath: //\*[contains(@attribute, ‘partial value’)]
  + CSS: tagname[attribute \*= ‘partial value’]

**Prefix:**

* Prefix of the attribute’s value can be used to create XPath / CSS
  + XPath: //\*[starts-with(@attribute, ’prefix value’)]
  + CSS: tagname[attribute ^= ‘prefix value’]

**Suffix:**

* Prefix of the attribute’s value can be used to create XPath / CSS
  + XPath: //\*[ends-with(@attribute, ’suffix value’)]
  + CSS: tagname[attribute $= ‘suffix value’]

**Visible Text:**

* Actual text or partial text of the element which is visible on the page can be used to create XPath. Note that this feature is not available in CSS
  + XPath: //\*[text()=‘actual text’]
  + XPath: //\*[contains(text()=‘partial text’)]

**AND operator:**

* Multiple values can be combined to create XPath / CSS
* It will find the element if both the values are TRUE
  + XPath: //\*[@attribute = ’value’ and @attribute = ’value’ ]
  + CSS: tagname[attribute1 = ‘value’] [attribute2 = ‘value’]

**OR operator:**

* Multiple values can be combined to create XPath / CSS
* It will find the element if one of the values are found to be TRUE
  + XPath: //\*[@attribute1 = ’value’ OR @attribute2 = ’value’ ]
  + CSS: tagname[attribute1 = ‘value’] , [attribute2 = ‘value’]