**Practical:-1**

**1. Introduction to Selenium, Installation and Setup.**

**Selenium** is a free (open-source) automated testing framework used to validate web applications across different browsers and platforms. You can use multiple programming languages like Java, C#, Python etc.. to create Selenium Test Scripts. Testing done using the Selenium testing tool is usually referred to as Selenium Testing.

**Step:-1** Install Selenium jar on web.{ selenium-java-3.11.0.jar download }

**Step:-2** Install Google Chrome Latest version Web Drivers.

**Step:-3** Use Eclipse IDE

**Step:-1** Create Java Project

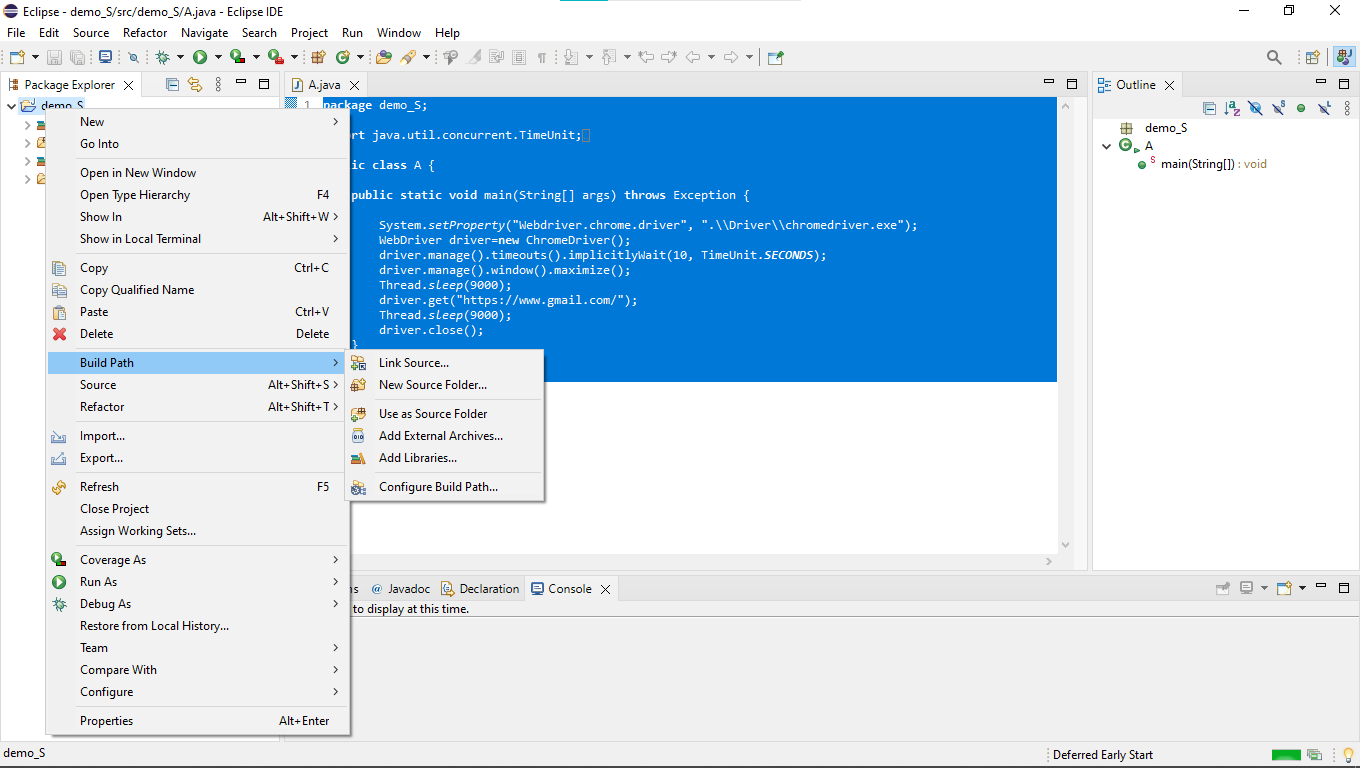
**Step:-2** Create Driver Folder

**Step:-3**  Copy Downloaded Chrome Driver in Driver Folder

**Step:-4** Set Chrome Driver Path in Environment Variables

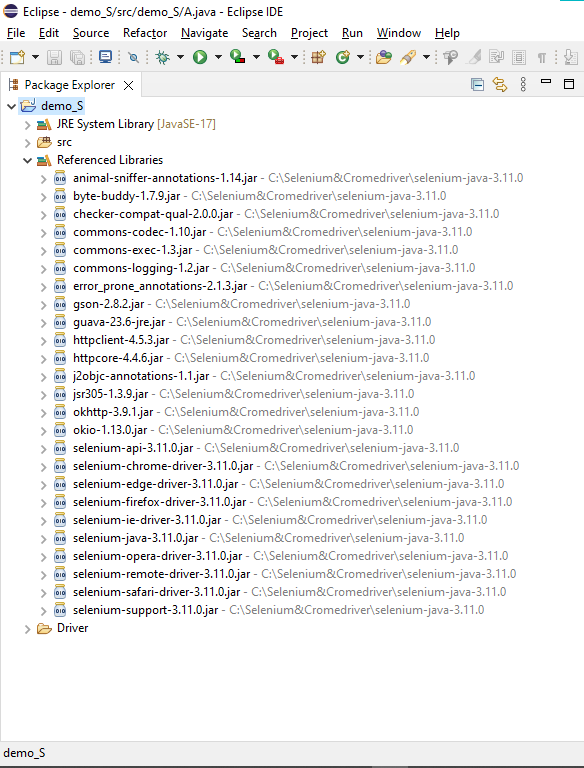
**Step:-5**  Right click and Select Build Path And Then Select Add External

Select To Build External.

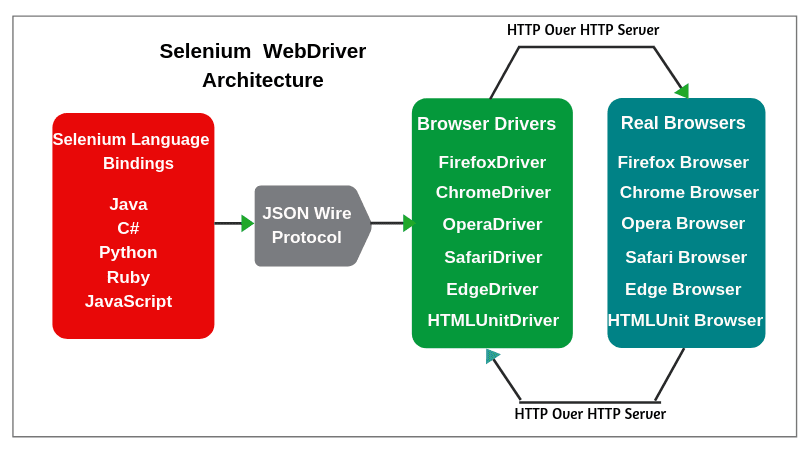


**Step:-6** Then Select Selenium Jar Files And Apply

**Step:-7**  We Can see Builded Jar File In Our Project.

****

**Architecture:-**



**Practical:-2**

**2. Selenium WebDriver Commands**

**• BrowserCommands**

**• NavigationCommands**

**• WebElementCommands**

**• FindElement and FindElementsCommand**

**• CheckBox & Radio ButtonOperations**

**• DropDown & Multiple SelectOperations**

**• Handle Dynamic Web Tables in SeleniumWebdriver.**

**• BrowserCommands**

### 1. Get Command

In WebDriver, this method loads a new web page in the existing browser window. It accepts String as parameter and returns void.

syntax:-

get(String arg0) : **void**

Example:-

### 2. Get Title Command

In WebDriver, this method fetches the title of the current web page. It accepts no parameter and returns a String.

syntax:-

getTitle(): String

Example:-

driver.getTitle();

     // Or can be written as

 String Title = driver.getTitle();

### 3. Get Current URL Command

### In WebDriver, this method fetches the string representing the Current URL of the current web page. It accepts nothing as parameter and returns a String value.

syntax:-

getCurrentUrl(): String

Example:-

driver.getCurrentUrl();

    //Or can be written as

    String CurrentUrl = driver.getCurrentUrl();

### 4. Get Page Source Command

### In WebDriver, this method returns the source code of the current web page loaded on the current browser. It accepts nothing as parameter and returns a String value.

syntax:-

getPageSource(): String

Example:-

driver.getPageSource();

    //Or can be written as

    String PageSource = driver.getPageSource();

### 5. Close Command

### This method terminates the current browser window operating by WebDriver at the current time. If the current window is the only window operating by WebDriver, it terminates the browser as well. This method accepts nothing as parameter and returns void.

syntax:-

driver.close();

Example:-

driver.close();

### 6. Quit Command

### This method terminates all windows operating by WebDriver. It terminates all tabs as well as the browser itself. It accepts nothing as parameter and returns void.

syntax:-

quit(): **void**

Example:-

driver.quit();

**• NavigationCommands**

### 1. Navigate To Command

### In WebDriver, this method loads a new web page in the existing browser window. It accepts String as parameter and returns void.

syntax:-

to(String arg0) : **void**

Example:-

driver.navigate().to("www.javatpoint.com");

### 2. Forward Command

In WebDriver, this method enables the web browser to click on the **forward** button in the existing browser window. It neither accepts anything nor returns anything.

syntax:-

to(String arg0) : **void**

Example:-

driver.navigate().forward();

### 3. Back Command

### In WebDriver, this method enables the web browser to click on the ****back**** button in the existing browser window. It neither accepts anything nor returns anything.

syntax:-

back() : **void**

Example:-

driver.navigate().back();

### 4. Refresh Command

### In WebDriver, this method refresh/reloads the current web page in the existing browser window. It neither accepts anything nor returns anything.

syntax:-

refresh() : **void**

Example:-

driver.navigate().refresh();

**• WebElementCommands**

### 1. Clear Command

syntax:-

clear() : **void**

Example:-

WebElement element = driver.findElement(By.id("UserName"));

element.clear();

//Or can be written as

driver.findElement(By.id("UserName")).clear();

### 2. Sendkeys Command

syntax:-

sendKeys(CharSequence? KeysToSend) : **void**

Example:-

WebElement element = driver.findElement(By.id("UserName"));

element.sendKeys("JavaTpoint");

//Or can be written as

driver.findElement(By.id("UserName")).sendKeys("JavaTpoint");

### 3. Click Command

syntax:-

click() : **void**

Example:-

WebElement element = driver.findElement(By.linkText("javaTpoint"));

element.click();

//Or can be written as

driver.findElement(By.linkText("javaTpoint")).click();

### 4. IsDisplayed Command

syntax:-

isDisplayed() : **boolean**

Example:-

WebElement element = driver.findElement(By.id("UserName"));

**boolean** status = element.isDisplayed();

//Or can be written as

**boolean** staus = driver.findElement(By.id("UserName")).isDisplayed();

### 5. IsEnabled Command

syntax:-

isEnabled() : **boolean**

Example:-

WebElement element = driver.findElement(By.id("UserName"));

**boolean** status = element.isEnabled();

//Or can be written as

**boolean** staus = driver.findElement(By.id("UserName")).isEnabled();

//Or can be used as

WebElement element = driver.findElement(By.id("userName"));

**boolean** status = element.isEnabled();

// Check that if the Text field is enabled, if yes enter value

**if**(status){

element.sendKeys("javaTpoint");

}

### 6. IsSelected Command

syntax:-

isSelected() : **boolean**

Example:-

WebElement element = driver.findElement(By.id("Sex-Male"));

**boolean** status = element.isSelected();

//Or can be written as

**boolean** staus = driver.findElement(By.id("Sex-Male")).isSelected();

### 7. Submit Command

syntax:-

submit() : **void**

Example:-

WebElement element = driver.findElement(By.id("SubmitButton"));

element.submit();

//Or can be written as

driver.findElement(By.id("SubmitButton")).submit();

### 8. GetText Command

syntax:-

getText() : String

Example:-

WebElement element = driver.findElement(By.xpath("anyLink"));

String linkText = element.getText();

### 9. GetTagName Command

syntax:-

getTagName() : String

Example:-

WebElement element = driver.findElement(By.id("SubmitButton"));

String tagName = element.getTagName();

 //Or can be written as

String tagName = driver.findElement(By.id("SubmitButton")).getTagName();

### 10. getCssValue Command

syntax:-

getCssvalue() : String

Example:-

### 11. getAttribute Command

syntax:-

getAttribute(String Name) : String

Example:-

WebElement element = driver.findElement(By.id("SubmitButton"));

 String attValue = element.getAttribute("id"); //This will return "SubmitButton"

### 12. getSize Command

syntax:-

getSize() : Dimension

Example:-

WebElement element = driver.findElement(By.id("SubmitButton"));

Dimension dimensions = element.getSize();

System.out.println("Height :" + dimensions.height + "Width : "+ dimensions.width);

### 13. getLocation Command

syntax:-

getLocation() : Point

Example:-

WebElement element = driver.findElement(By.id("SubmitButton"));

Point point = element.getLocation();

System.out.println("X cordinate : " + point.x + "Y cordinate: " + point.y);

**• FindElement and FindElementsCommand**

|  |  |
| --- | --- |
| **findElement** | **findElements** |
| Returns the first matching web element if multiple web elements are discovered by the locator | Returns a list of multiple matching web elements |
| Throws **NoSuchElementException** if the element is not found | Returns an empty list if no matching element is found |
| Detects a unique web element | Returns a collection of matching elements |

1. ID
2. Name
3. Class Name
4. Tag Name
5. Link Text
6. XPath

### 1. Find by ID

ID is uniquely defined for each element and is the most common way to locate elements using ID Locator. If a website has dynamically generated ids, then this strategy cannot be used to uniquely find an element. However, it will return the first web element which matches the locator.

elementID = driver.findElement(By.id("tagid"))

### 2. Find by Name

This is similar to Find By ID except the driver will locate an element by “name” attribute instead of “id”.

elementName = driver.findElement(By.name("field-keywords"))

### 3. Find By LinkText

LinkText is helpful to find links in a webpage. It is the most efficient way of finding web elements containing links.

elementLinktext = driver.findElement(By.linkText(“Returns”))

### 4. Find By CSS Selector

[CSS Selector](https://www.browserstack.com/guide/css-selectors-in-selenium) is used to provide style rules for web pages and also can be used to identify one or more web elements.

elementcss= driver.findElement(By.cssSelector('div.nav-search-input'))

### 5. Find By XPath

[XPath](https://www.browserstack.com/guide/xpath-in-selenium) is a technique in Selenium to navigate through the HTML structure of a page. XPath enables testers to navigate through the XML structure of any document and can be used on both HTML and XML documents.

elementxpath = driver.findElement(By.xpath("//input[@id=’twotabsearchtextbox’]"))

**• Check Box & Radio Button Operations**

**• DropDown & Multiple SelectOperations**

The **Select**class is a Webdriver class which provides the implementation of the HTML SELECT tag. It exposes several “Select By” and “Deselect By” type methods. We use these methods to select or deselect in the drop down list or multi select object. The **Select**class is the part of the selenium package.

**Types of Select Methods:**

i. selectByVisibleText Method  
ii. selectByIndex Method  
iii. selectByValue Method

**i. selectByVisibleText Method**

It works based on the ‘visible text‘ provided by us.

syntax:-

dropdown.selectedByVisibleText(“Autometion testing”);

**ii. selectByIndex Method**

It works based on the ‘index value‘ provided by us.

Syntax:-

dropdown.selectByIndex(Index);

**iii. selectByValue Method**

It works based on the ‘value‘ provided by us.

syntax:-

dropdown.selectByValue(Value);

**Types of DeSelect Methods:**

i. deselectByVisibleText Method  
ii. deselectByIndex Method  
iii. deselectByValue Method  
iv. deselectAll Method

i. deselectByVisibleText Method

It works based on the ‘visible text‘ which we provide

syntax:-

dropdown.deselectByIndex();

ii. deselectByIndex Method

It works based on the ‘index value’ which we provide

syntax:-

dropdown.deselectByVisibleText();

iii. deselectByValue Method

It works based on the ‘value‘ provided by us.

syntax:-

dropdown.deselectByValue();

iv. deselectAll Method

It is to deselect all the selected options at once.

syntax:-

dropdown.deselectAll();

**• Handle Dynamic Web Tables in SeleniumWebdriver.**

Web Tables are like normal tables where the data is presented in a structured form using rows and columns. The only difference is that they are displayed on the web with the help of HTML code.

**<table>** is the HTML tag that is used to define a web table. While **<th>** is used for defining the header of the table, **<tr>** and **<td>** tags are used for defining rows and columns respectively for the web table.

<table>

<tr>

<th>First Name</th>

<th>Last Name</th>

<th>Age</th>

</tr>

<tr>

<td>Jill</td>

<td>Ann</td>

<td>24</td>

</tr>

<tr>

<td>Eve</td>

<td>Anderson</td>

<td>34</td>

</tr>

</table>

## **Types of Web Tables**

**1. Static Web Tables**

These tables have fixed data that remains unchanged throughout. Due to the static nature of their content, they are called Static web tables.

**2. Dynamic Web Tables**

These tables have data that changes over time, and hence the number of rows and columns might also change depending upon the data shifts. Due to the dynamic nature of their content, they are called Dynamic web tables.

## **Finding XPath Selected Element in Dynamic Web Table**

To find the XPath of a UI element in Firefox, right-click on the desired element, go to “**Inspect Element**” to open the inspector which will help identify its [XPath](https://www.browserstack.com/guide/xpath-in-selenium).

## **Finding the number of rows and columns of Dynamic Web Table in Selenium**

//Finding number of Rows

List<WebElement> rowsNumber = driver.findElements(By.xpath(“//\*[@id="content-8b4988b9-2ec9-4e77-9b4d-9c2994bd9e8a"]/div/div/table[1]/tbody/tr[1]/td[1]”));

int rowCount = rowsNumber.size();

System.out.println("No of rows in this table : " + rowCount);

**Output: No of rows in this table: 8**

//Finding number of Columns

List<WebElement> columnsNumber = driver.findElements(By.xpath("//\*[@id="content-8b4988b9-2ec9-4e77-9b4d-9c2994bd9e8a"]/div/div/table[1]/thead/tr/th[1]"));

int columnCount = columnsNumber.size();

System.out.println("No of columns in this table : " + columnCount);

**Output: No of columns in this table: 9**

**Practical:-3**

**3. Navigate back/forwards, get, refresh**

**• I\_ loading a page in current window / New window**

**• Move back and forward**

**• Refresh Page**

**• I\_ loading a page in current window / New window**

**• Move back and forward**

back method is used to go one step back in browser history.

**Syntax –**

driver.back()

**Example –**

driver.get("https://www.google.com/")

driver.back()

forward method is used to go one step forward in browser history.

**Syntax –**

driver.forward()

**Example –**

driver.get("https://keyu04.github.io/")

driver.forward()

**• Refresh Page**

This is the inbuilt method for performing page refresh operation provided by Selenium web driver. This command is the most commonly used command across test automation for performing a page refresh operation. Refresh command can be used in a simple way as mentioned below.

Example:-

driver.get("https://keyu04.github.io/keyurhalpati.github.io/");

driver.navigate().refresh();

**Practical:-4**

**4. Interrogation:**

**i. get windowtitle**

**ii. currenturl**

**iii. Pagesource**

**i. get windowtitle**

The method **getTitle()** is used to obtain the present page title and then we can get the result in the console.

**Syntax –**

driver.getTitle ()

**Example –**

System.*setProperty*("webdriver.chrome.driver",".\\Driver\\chromedriver.exe");

WebDriver driver=**new** ChromeDriver();

driver.get("https://keyu04.github.io/keyurhalpati.github.io/");

driver.manage().timeouts().implicitlyWait(50000000,TimeUnit.***SECONDS***);

System.***out***.print("Tab Title "+driver.getTitle());

driver.close();

**ii. currenturl**

We can obtain the URL of the current page with Selenium webdriver. This is achieved with the help of getCurrentUrl() method. It fetches the URL of the opened application. This method accepts no parameters and strings the URL in the form of String.

**Syntax –**

driver.getCurrentUrl ()

**Example –**

System.*setProperty*("webdriver.chrome.driver",".\\Driver\\chromedriver.exe");

WebDriver driver=**new** ChromeDriver();

driver.get("https://keyu04.github.io/keyurhalpati.github.io/");

driver.manage().timeouts().implicitlyWait(500, TimeUnit.***SECONDS***);

System.***out***.print("Current Url id "+driver.getCurrentUrl());

Driver.close();

**iii. Pagesource**

We can get page source as it is in browser using Selenium webdriver using the getPageSource method. It allows us to obtain the code of the page source.

**Syntax –**

driver.getPageSource ()

**Example –**

System.*setProperty*("webdriver.chrome.driver", ".\\Driver\\chromedriver.exe");

WebDriver driver=**new** ChromeDriver();

driver.get("https://keyu04.github.io/keyurhalpati.github.io/");

driver.manage().timeouts().implicitlyWait(5, TimeUnit.***SECONDS***);

System.***out***.print("Source Code"+driver.getPageSource());

driver.close();

**Practical:-5**

**5. Locating web elements by**

**Id, ClassName, LinkText, PartialLinkText, Name, TagName, CssSelector ,XPath**

**ID**

ID’s are unique for each element so it is a common way to locate elements using ID Locator. It is the most common fastest and safest way to detect an element. It is recommended for website developers to avoid using non-unique Ids or dynamically generated Ids however some MVC frameworks like – ADF can lead to pages with dynamically generated ids.

**Syntax –**

driver.findElement(By.id(<locator\_value>))

**Example –**

System.setProperty("webdriver.chrome.driver",".\\Driver\\chromedriver.exe");

WebDriver driver=new ChromeDriver();

driver.get("https://www.python.org");

WebElement search=driver.findElement(By.id("downloads"));

search.click();

driver.close();

**ClassName**

This method finds elements based on the value of the *CLASS* attribute. More applicable for locating multiple elements which has a similar css class defined against them.

**Syntax –**

driver.findElements(By.**className**(<locator\_value>)) ;

**Example –**

System.*setProperty*("webdriver.chrome.driver",".\\Driver\\chromedriver.exe");

WebDriver driver=**new** ChromeDriver();

driver.manage().timeouts().implicitlyWait(5000, TimeUnit.***SECONDS***);

driver.get("https://www.python.org");

List<WebElement> links = driver.findElements(By.*className*("search-field"));

System.***out***.print("text"+links);

**LinkText**

With this method, one can find elements of “a” tags (Link) with the link names or having matching partial link names. This strategy is only applicable in finding element(s) of type anchor tags which contain a text value.

**Syntax –**

driver.findElement(By.**linkText**(<link\_text>)) ;

**Example –**

System.*setProperty*("webdriver.chrome.driver",".\\Driver\\chromedriver.exe"); WebDriver driver=**new** ChromeDriver();

driver.manage().timeouts().implicitlyWait(5000, TimeUnit.***SECONDS***);

driver.get("https://www.python.org");

List<WebElement> links = driver.findElements(By.*linkText*("Menu"));

System.***out***.print("text"+links);

**PartialLinkText**

With this method, one can find elements of “a” tags (Link) with the link names or having matching partial link names. This strategy is only applicable in finding element(s) of type anchor tags which contain a text value.

**Syntax –**

driver.findElement(By.**partialLinkText**(<link\_text>)) ;

**Example –**

System.*setProperty*("webdriver.chrome.driver",".\\Driver\\chromedriver.exe");

WebDriver driver=**new** ChromeDriver();

driver.manage().timeouts().implicitlyWait(5000, TimeUnit.***SECONDS***);

driver.get("https://www.python.org");

List<WebElement> links = driver.findElements(By.*partialLinkText*("Menu"));

System.***out***.print("text"+links);

**Name**

This method is similar to Find By Id except the driver will try to locate an element by “name” attribute instead of “id” attribute.

**Syntax –**

driver.findElement(By.name("name"));

**Example –**

System.*setProperty*("webdriver.chrome.driver",".\\Driver\\chromedriver.exe");

WebDriver driver=**new** ChromeDriver();

driver.manage().timeouts().implicitlyWait(5000, TimeUnit.***SECONDS***);

driver.get("https://www.python.org");

List<WebElement> links = driver.findElements(By.*name*("q"));

System.***out***.print("text"+links);

**TagName**

This method finds elements based on the HTML tag name of the element. This is not widely used and used as the last resort if the particular web element can’t be detected by Id/name/link/className/XPATH/CSS.

**Syntax –**

driver.findElement(By.**tagName**(<locator\_value>)) ;

**Example –**

System.*setProperty*("webdriver.chrome.driver",".\\Driver\\chromedriver.exe");

WebDriver driver=**new** ChromeDriver();

driver.manage().timeouts().implicitlyWait(5000, TimeUnit.***SECONDS***);

driver.get("https://www.python.org");

WebElement links = driver.findElement(By.*tagName*("body")); System.***out***.print("text"+links);

**CssSelector**

For websites generating dynamic Ids like ADF based applications or websites which are built on latest javascript frameworks like – React js which may not generate any Ids or names can’t use locator by Id/Name strategy to find elements. Instead, we have to use either CSS selector or XPath selectors.

**Syntax –**

driver.findElement(By.*cssSelector* (<locator\_value>)) ;

**Example –**

System.*setProperty*("webdriver.chrome.driver",".\\Driver\\chromedriver.exe");

WebDriver driver=**new** ChromeDriver();

driver.manage().timeouts().implicitlyWait(5000, TimeUnit.***SECONDS***);

driver.get("https://www.python.org");

WebElement links = driver.findElement(By.*cssSelector*("input#id-search-field"));

System.***out***.print("text"+links);

**XPath**

XPATH is more readable and the learning curve is less steep since it uses standard XML query syntaxes, however, CSS selectors though have simpler syntax support but are not standard like XPATH and other documentation support, unlike XPATH.

**Syntax –**

driver.findElement(By.*xpath*(//tagname[@attribute='value']

)

**Example –**

System.*setProperty*("webdriver.chrome.driver",".\\Driver\\chromedriver.exe");

WebDriver driver=**new** ChromeDriver();

driver.manage().timeouts().implicitlyWait(5000, TimeUnit.***SECONDS***);

driver.get("https://www.python.org");

WebElement links = driver.findElement(By.*xpath*("//input[@id='id-search-field']"));

System.***out***.print("text"+links);

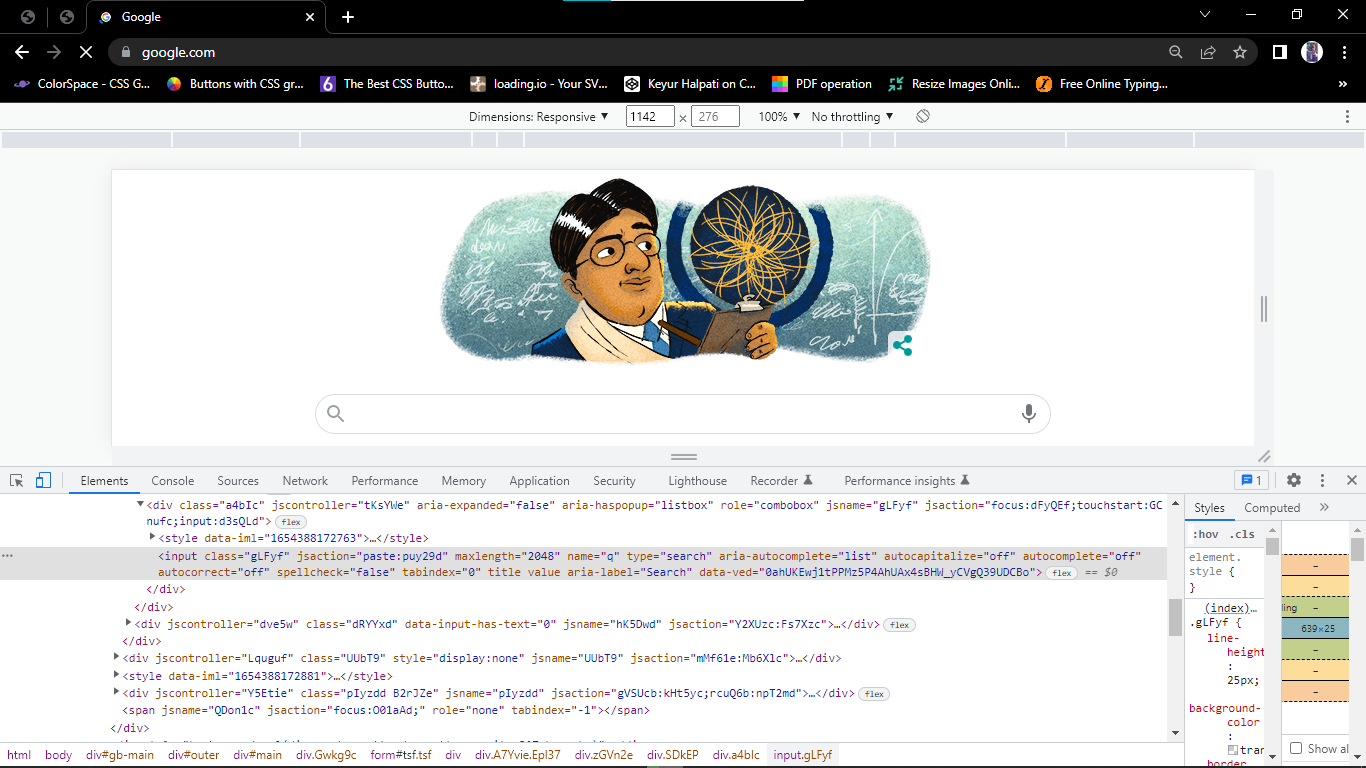
**Practical:-6**

**6. Inspecting elements in web browsers.**

**Step:-1** Write Scrip to open The specific URL.

**Step:-2** Then click or Press Shift+Ctrl+I or Right click and select Inspect Option.

**Step:-3** That open Inspect window



**Step:-4** To inspect particular Element on browser press Ctrl+P. it provide Textbox where we can write our code to find the particular element(we can use locator to find the elements.)

**Step:-5** that code we put in our script to perform automation.

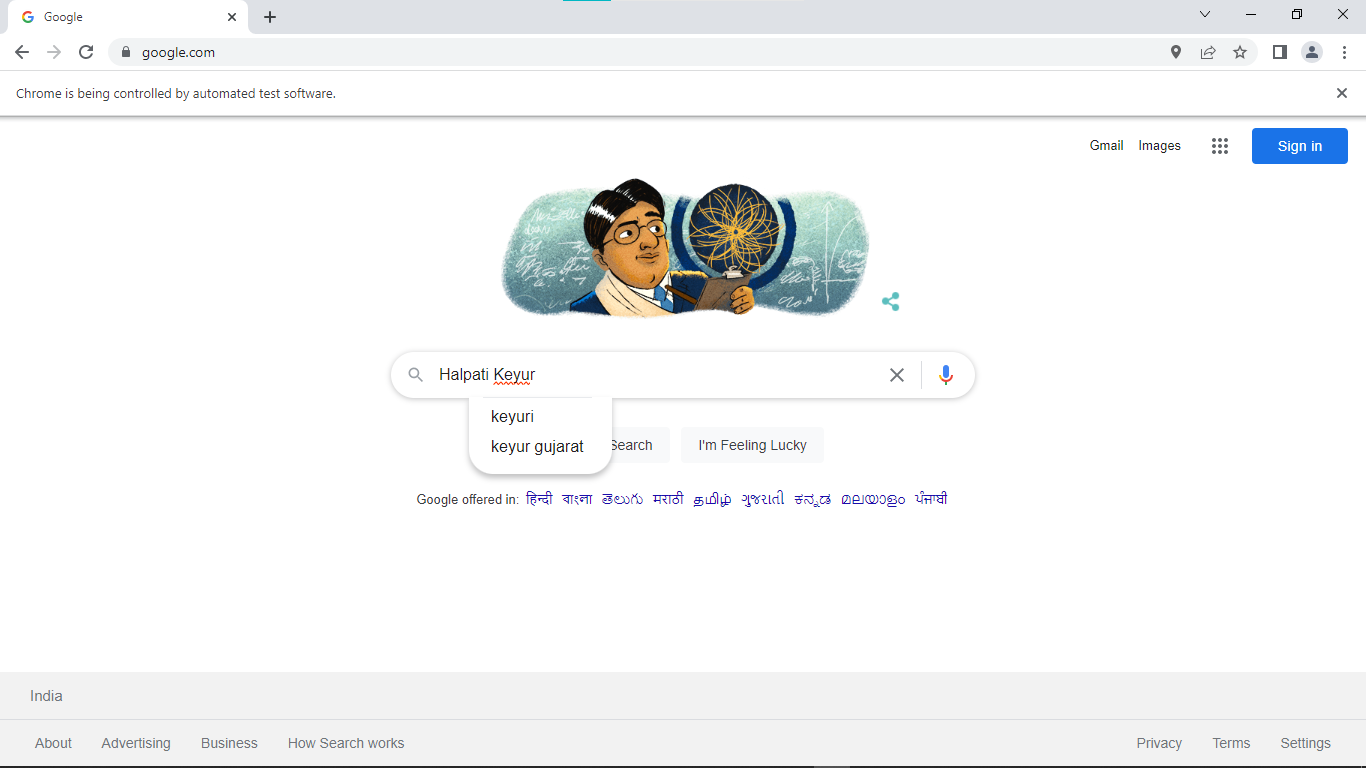
**Step:-6** find input text location using class locator and write some Text to conform We locate that inputbox.

**Ex:**

System.*setProperty*("webdriver.chrome.driver", ".\\Driver\\chromedriver.exe");

WebDriver driver=**new** ChromeDriver();

driver.get("https://www.google.com/"); driver.findElement(By.*className*("gLFyf")).sendKeys("Halpati Keyur");



**Practical:-7**

**7. Element interrogation**

This category includes commands help your automation to find user interface (UI) elements on your application's pages (screens) and retrieve the information from those elements.  
  
Most of these commands that start with 'get', 'assert' and 'verify' belong to this category. This category also includes commands that help you print the value of variables within your automated tests like standard output commands do in programming languages ('echo' in shell and batch scripts OR 'print' and 'println' in Java).

**Driver level:**These are web driver level interrogations like getting the title of the current page, getting the current URL in context, and getting the source of the current page.

**DOM level:**DOM stands for "Document Object Model". It defines the logical structure of HTML (and XML documents) and the way a page is accessed and manipulated. Everything in a web page such as text input field, button, radio button, checkbox, drop-down, plain text, text area, etc. is a web element in Web Driver.

Interrogation is to be done in two steps, and eureQa supports commands for each step:

1. First, your automated script has to find (also known as 'locate') the element which we want to interrogate. Finding or Locating the elements can be done by id, name, link text, partial link text, class attribute name, HTML tag name, CSS Selectors or XPath Selectors.
2. Then, your automated script can interrogate that element and retrieve information from it. 'Asserts' & 'Verify' series of commands belong to this category.

Assert and Verify are both 'Assertions' which check that the state of the application conforms to what is expected of you(test). Examples include “make sure the page title is X” and “verify that this checkbox is checked”. Effectively an “assert” will fail the test and abort the currently executing test, whereas a “verify” will fail the test and continue to run the currently executing test.

The table(s) below lists the commands in the ascending alphabetic order within each category along with a brief description of the purpose of the command. By clicking on the hyperlinked command

name in the table(s) below, you can review more detailed information about a specific command that includes the syntax for you to follow when using the commands.

|  |  |
| --- | --- |
| **Command** | **Purpose** |
| [**assertAlert**](https://eureqa.uservoice.com/knowledgebase/articles/1113961-assertalert-ui-automation-interrogation) | assertAlert will also verify text same as verifyAlert and take action on the alert box but if the alert text does not match then it will stop execution |
| [**assertAlertPresent**](https://eureqa.uservoice.com/knowledgebase/articles/1114039-assertalertpresent-ui-automation-interrogation) | This command checks for the presence of an alert and this function never throws an exception |
| [**assertAllFields**](https://eureqa.uservoice.com/knowledgebase/articles/1114534-assertallfields-ui-automation-interrogation) | If a given field has no ID, it will appear as "" in this array. It Returns the IDs of all input fields on the page. If a given field has no ID, it will appear as "" in this array. |
| [**assertAllLinks**](https://eureqa.uservoice.com/knowledgebase/articles/1114558-assertalllinks-ui-automation-interrogation) | If a given link has no ID, it will appear as "" in this array. Returns the IDs of all links on the page.If a given link has no ID, it will appear as "" in this array. |
| [**assertAttribute**](https://eureqa.uservoice.com/knowledgebase/articles/1114561-assertattribute-ui-automation-interrogation) | Gets the value of an element attribute. The value of the attribute may differ across browsers (this is the case for the 'style' attribute, for example). |
| [**assertBackgroundColor**](https://eureqa.uservoice.com/knowledgebase/articles/1114564-assertbackgroundcolor-ui-automation-interrogati) | Retrieves the CSS background-color attribute value applied to that element |
| [**assertBackgroundImage**](https://eureqa.uservoice.com/knowledgebase/articles/1114582-assertbackgroundimage-ui-automation-interrogati) | Retrieves the CSS background-image attribute value applied to that element. |
| [**assertChecked**](https://eureqa.uservoice.com/knowledgebase/articles/1114585-assertchecked-ui-automation-interrogation) | Gets whether a toggle-button (checkbox/radio) is checked. Fails if the specified element doesn't exist or isn't a toggle-button. |
| [**assertColor**](https://eureqa.uservoice.com/knowledgebase/articles/1114588-assertcolor-ui-automation-interrogation) | Retrieves the CSS color attribute value applied to that element |

**Practical:-8**

**8 Manipulation: Click, submit, shift-click, special actions, type text, clear text, list box selection and manipulation commands.**

**1.Click:**

A click is the most fundamental user action performed by anyone accessing the internet. It allows the user to navigate web pages or perform particular tasks by interacting with links, buttons, and other web elements.

needs to first click on the address bar to enter a URL in order to visit a particular website. Then, to login to an account, one needs to click on input fields like ID and Password to enter credentials. Then, they must click on Login. In a nutshell, the click operation is at the core of web browsing.

**Example:**

driver.get("https://www.browserstack.com/");

driver.findElement(By.id("signupModalButton")).click(); //using Selenium click button method

**2.submit:**

he click() method that we discussed above can be used to click on any links or buttons. Submit() is a better alternative to click() if the element to be clicked is a submit button. The submit button is inside the HTML ‘form’ tag and the type of button is ‘submit’(not ‘button’).

The submit() makes life easier by automatically finding the button and the method that can be appended to any other field like name or email address. In the case of click, we have to use findElement(By, by) method and specify the correct locators.

In some scenarios where the action is done through elements other than a button, submit() works and click() won’t.

**Syntax:**

driver.findElement(By.xpath("//input[@name='comments']")).submit();

**3.special action:-**

Actions class is an ability provided by Selenium for handling keyboard and mouse events. In [Selenium WebDriver](https://www.browserstack.com/guide/selenium-webdriver-tutorial), handling these events includes operations such as drag and drop, clicking on multiple elements with the control key, among others. These operations are performed using the advanced user interactions API. It mainly consists of *Actions*that are needed while performing these operations.

**Syntax:**

Actions action = new Actions(driver);

action.moveToElement(element).click().perform();

**Methods of Action Class:**

Action class is useful mainly for mouse and keyboard actions. In order to perform such actions, Selenium provides various methods.

 Mouse Actions in Selenium:

1. **doubleClick()**: Performs double click on the element
2. **clickAndHold()**: Performs long click on the mouse without releasing it
3. **dragAndDrop()**: Drags the element from one point and drops to another
4. **moveToElement()**: Shifts the mouse pointer to the center of the element
5. **contextClick()**: Performs right-click on the mouse

**4.list box selection:**

ListBox is an element where user can select/deselect one or more items from it.

To identify the ListBox on webpage, look for select tag and attribute should be ‘multiple’ to select multiple items and there will be option tag which contains each item in it.

**Syntax:**

WebElement val=driver.findElement(By.id("tools"));

**Practical:-9**

**9.Synchronization: Page load time out, implicit wait, explicit wait,ExpectedConditions class.**

**1. pageLoadTimeout(time,unit):**

 pageLoadTimeout(time,unit) to set the time for a page to load.

Sometimes due to server issues or network delays, a page might take more than usual time to load. This might throw an error in the program. In order to avoid this, we set a wait time and pageLoadTimeout() is one of such method. This will usually follow a get() command.

**Syntax:**

driver.manage().timeouts().pageLoadTimeout(500, SECONDS);

**Explanation:**

* Wait for 500 seconds for a page to load.

**2. implicit Wait:**

 implicitlyWait() to set a wait time before searching and locating a web element.

What happens if the Webdriver tries to locate an element before the webpage loads and the element appears? NoSuchElementExeption will be thrown. In order to avoid this, we can add a command to implicitly wait for a certain amount of time before locating the element.

**Syntax:**

driver.manage().timeouts().implicitlyWait(1000, TimeUnit.SECONDS);

**Explanation:**

* Implicitly wait for 1000 seconds before executing the next line in the code

**3.Explicit wait:**

By using the Explicit Wait command, the WebDriver is directed to wait until a certain condition occurs before proceeding with executing the code.In order to declare explicit wait, one has to use “ExpectedConditions”. The following Expected Conditions can be used in Explicit Wait.

**Example**:

WebDriverWait wait = new WebDriverWait(driver,30);

**4.Expected condtion class:**

A Custom ExpectedCondition is a class that consists of a constructor with the parameters of the expected condition. It implements the ExpectedCondition interface and overrides the apply method.

**Syntax:**

import org.openqa.selenium.support.ui.ExpectedConditions;

import org.openqa.selenium.support.ui.WebDriverWait;

**Practical:-10**

**10. Window handling: size, position, handles, switch to.**

**window handling in Selenium and**answer this question. Sometimes, we need to deal with a new window or multiple windows in our test automation projects and for these situations, we should use webdriver API’s built-in window methods.

**Size:**

The [size()](https://www.geeksforgeeks.org/list-size-method-in-java-with-examples/) method of List interface in Java is used to get the number of elements in this list. That is, this method returns the count of elements present in this list container.

So essentially, the variable **link** which is of type WebElement won't support the size() method. So you have to change the type of the variable **link** as List and populate the List using findElements() method and you can use the following solution:

List<WebElement> link = driver.findElements(By.tagName("a"));

System.out.println(link.size());

**– .getSize()** – It returns the size of the current window.

**– .setSize()** – It sets a new size to current window*.*

**Position:**

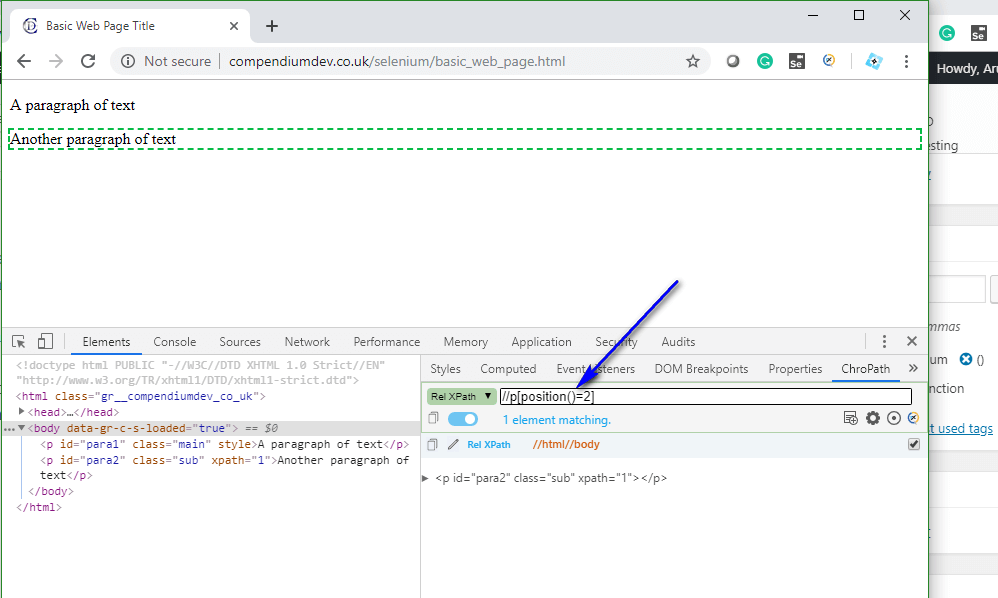
Let’s locate the second paragraph p tag using the**position()** XPath Function in Relative XPath Expression.

Relative XPath Expression: //p[**position()=2**]

Execute the above Relative XPath Expression in the ‘ChroPath’ and observe that the second paragraph is getting located as shown below:

**– .getPosition()** – It returns current position in term of x,y coordinates.

**– .setPosition()** – It moves the current window around.



**Handle:**

Every window has a unique handle. Handle means a specific identifier that represents its window. We can manage the windows with their handles. I want to explain what Selenium Webdriver provides us for window handles.

**– driver.getWindowHandle(**) – We can get the current window’s handle

**– driver.getWindowHandles()** – We can get all windows handles

**– driver.switchTo().window(String handle)** – We can switch to the target window by using its handle

**Practical:-11**

**11.Screenshot/capture**

A **Screenshot in Selenium Webdriver** is used for bug analysis. Selenium webdriver can automatically take screenshots during the execution. But if users need to capture a screenshot on their own, they need to use the TakeScreenshot method which notifies the WebDrive to take the screenshot and store it in Selenium.

**Step1:Convert web driver object to TakeScreenshot**

TakesScreenshot scrShot =((TakesScreenshot)webdriver);

**Step 2: Call getScreenshotAs method to create image file:**

File SrcFile=scrShot.getScreenshotAs(OutputType.FILE);