Practical No. 2. INTRODUCTION TO SELENIUM AND LOCATORS (ID, CSS SELECTOR, PATH) IN SELENIUM.

Date: ________________

Aim:

To understand selenium tools and study various locators in Selenium to locate different web elements.

Theory:

What is Selenium?

Selenium is a free (open source) automated testing suite for web applications across different browsers and platforms. It is quite similar to HP Quick Test Pro (QTP now UFT) only that Selenium focuses on automating web-based applications. Testing done using Selenium tool is usually referred as Selenium Testing.

Selenium is not just a single tool but a suite of software's, each catering to different testing needs of an organization. It has four components.

- Selenium Integrated Development Environment (IDE)
- Selenium Remote Control (RC)
- WebDriver
- Selenium Grid

What is WebDriver?

Selenium Webdriver is an open-source collection of APIs which is used for testing web applications.

The Selenium Webdriver tool is used for automating web application testing to verify that it works as expected or not. It mainly supports browsers like Firefox, Chrome, Safari and Internet Explorer.

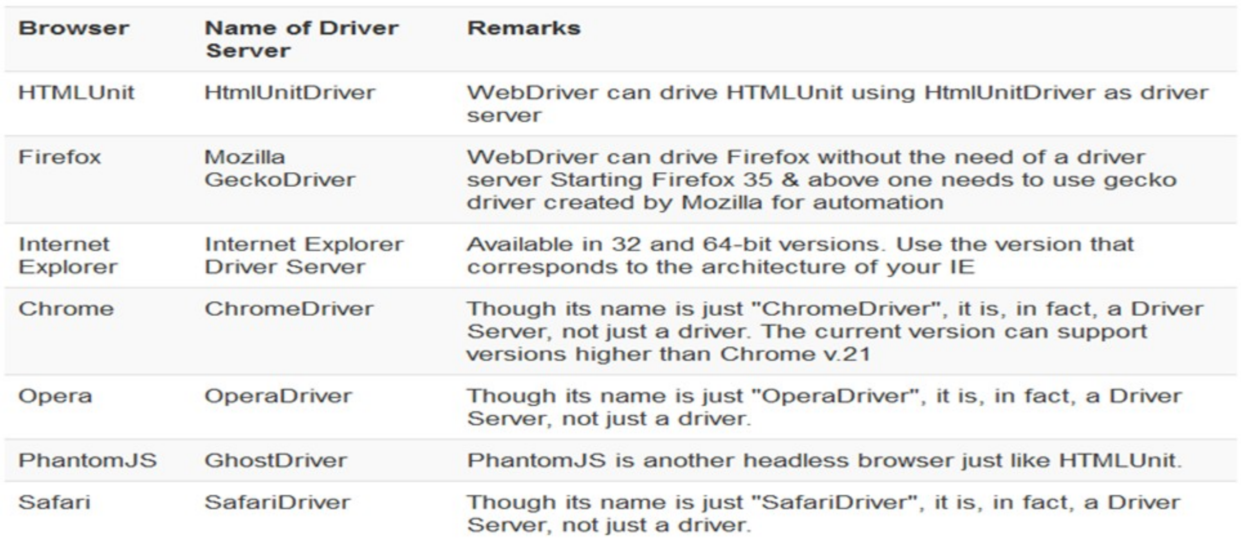
It also permits you to execute cross-browser testing. WebDriver also enables you to use a programming language in creating your test scripts (not possible in Selenium IDE).

Following programming languages are supported by WebDriver

- Java
- .Net
- PHP
- Python
- Perl
- Ruby

It controls the browser from the OS level. All you need are your programming language's IDE (which contains your Selenium commands) and a browser.

Below is the list of driver servers and the corresponding browsers that use them.



Browser	Name of Driver Server	Remarks
HTMLUnit	HtmlUnitDriver	WebDriver can drive HTMLUnit using HtmlUnitDriver as driver server
Firefox	Mozilla GeckoDriver	WebDriver can drive Firefox without the need of a driver server Starting Firefox 35 & above one needs to use gecko driver created by Mozilla for automation
Internet Explorer	Internet Explorer Driver Server	Available in 32 and 64-bit versions. Use the version that corresponds to the architecture of your IE
Chrome	ChromeDriver	Though its name is just "ChromeDriver", it is, in fact, a Driver Server, not just a driver. The current version can support versions higher than Chrome v.21
Opera	OperaDriver	Though its name is just "OperaDriver", it is, in fact, a Driver Server, not just a driver.
PhantomJS	GhostDriver	PhantomJS is another headless browser just like HTMLUnit.
Safari	SafariDriver	Though its name is just "SafariDriver", it is, in fact, a Driver Server, not just a driver.

First Selenium Webdriver Script

Let us try to create a WebDriver script that would: Open www.google.com in firefox browser
Steps:

1. Open Eclipse IDE
2. Open SeleniumWDLab project which is already configured with selenium jar files. If not create a new project by following instruction given previously.
3. Right Click on src and create new package WDIntro.
4. Right click on WDIntro and create new java class first.java.
5. Copy Paste the code given in next slide in first.java file
6. Run the file as java application.

```
 package WDIntro;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
public class first {
        public static void main(String[] args) {
         System.setProperty("webdriver.gecko.driver","D:\\selenium_drivers\\geckodriver.exe");
                WebDriver driver = new FirefoxDriver();
                String baseURL="http://google.com";
                driver.get(baseURL);
        }
}
```

Importing Packages

To get started, you need to import following two packages:

1. **org.openqa.selenium.*–** contains the WebDriver class needed to instantiate a new browser loaded with a specific driver
2. **org.openqa.selenium.firefox.FirefoxDriver** – contains the FirefoxDriver class needed to instantiate a Firefox-specific driver onto the browser instantiated by the WebDriver class

If your test needs more complicated actions such as accessing another class, taking browser screenshots, or manipulating external files, definitely you will need to import more packages.

Instantiating objects and variables

Normally, this is how a driver object is instantiated.

```
        WebDriver driver = new FirefoxDriver();
```

A FirefoxDriver class with no parameters means that the default Firefox profile will be launched by our Java program.
The default Firefox profile is similar to launching Firefox in safe mode (no extensions are loaded).
For convenience, we saved the Base URL and the expected title as variables.
WebDriver's get() method is used to launch a new browser session and directs it to the URL that you specify as its parameter.

```
        driver.get(baseURL);
```

Selenium Locators

Locator is a command that tells Selenium IDE which GUI elements (say Text Box, Buttons, Check Boxes etc) its needs to operate on. Identification of correct GUI elements is a prerequisite to creating an automation script.

But accurate identification of GUI elements is more difficult than it sounds. Sometimes, you end up working with incorrect GUI elements or no elements at all! Hence, Selenium provides a number of Locators to precisely locate a GUI element

- ID
- Name
- Link Text
- CSS Selector
  - Tag and ID
  - Tag and class
  - Tag and attribute
  - Tag, class, and attribute
  - Inner text
- DOM (Document Object Model)
  - getElementById
  - getElementsByName
  - dom:name
  - dom: index
  - XPath

Locating by Name

Every element on a web page has many attributes. **Name** is one among them.

```
WebElement searchBox = driver.findElement(By.name("q"));
```

Here name is one of the many attributes of the button, and its value is q

Locating by ID

On a web page, each element is uniquely identified by an **ID**, if provided. An ID can be assigned manually by the developer of the web application or, most of the times, left to be dynamically generated by the server where the web application is hosted, and this ID can change over a period of time.

```
WebElement emailbox = driver.findElement(By.id("email"));
```

Here id is one of the many attributes of the email textbox, and its value is “email”

Entering Values in Input Boxes

To enter text into the Text Fields and Password Fields, **sendKeys()** is the method available on the WebElement in Selenium.

```
WebElement emailbox = driver.findElement(By.id("email"));
emailbox.sendKeys("abcd@gmail.com");
WebElement pwdbox = driver.findElement(By.id("passwd"));
pwdbox.sendKeys("abcdefghlkjl");
```

Deleting Values in Input Boxes

The **clear()** method is used to delete the text in an input box.

This method does not need a parameter.

The code snippet below will clear out the text from the Email or Password fields

```
emailbox.clear();
pwdbox.clear();
```

Locating by Link Text

Accessing links using their exact link text is done through the **By.linkText()** method.

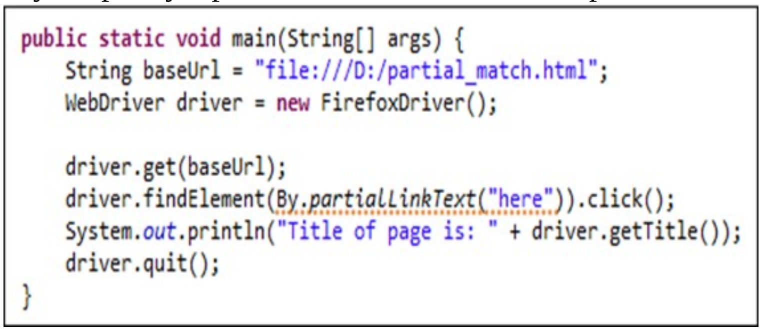
However, if there are two links that have the very same link text, this method will only access the first one.

```
public class LocatingByLink {
	public static void main(String[] args) {
	// TODO Auto-generated method stub
	System.setProperty("webdriver.gecko.driver","D:\\selenium_drivers\\geckodriver.exe");
	WebDriver driver = new FirefoxDriver();
```

```
            driver.get("http://google.com");
            WebElement imglink = driver.findElement(By.linkText("Images"));
            imglink.click();
            }
}
```

Locaating by partialLinkText

Accessing links using a portion of their link text is done using the **By.partialLinkText()** method. If you specify a partial link text that has multiple matches, only the first match will be accessed.



```
public static void main(String[] args) {
    String baseUrl = "file:///D:/partial_match.html";
    WebDriver driver = new FirefoxDriver();

    driver.get(baseUrl);
    driver.findElement(By.partialLinkText("here")).click();
    System.out.println("Title of page is: " + driver.getTitle());
    driver.quit();
}
```

Locating by Tag Name

Locating an element by tag name is slightly different from name and id locating mechanisms. The reason being it can return zero or more results. For example, on a Google Search page, if you search for an element with the tag name button, it will result in three WebElements because there are three buttons present on the search page.

So, it is always advisable to use the findElements() method rather than the findElement() method when trying to locate elements using tag names.

```
 public class LocatingByTagname1 {
public static void main(String[] args) {
System.setProperty("webdriver.gecko.driver","D:\\selenium_drivers\\geckodriver.exe");
            WebDriver driver = new FirefoxDriver();
            driver.get("http://demo.guru99.com/test/facebook.html");
            List <WebElement> list = driver.findElements(By.tagName("input"));
            for(int i = 0; i < list.size(); i++)
                        {
                        System.out.println(list.get(i).getAttribute("name"));
                        }
            }
}
```

Locating by Class Name

On web page elements can also be located by classname. we can use the **By.className** locating mechanism by passing the class attribute value to it.

```
public class LocatingByClassname {
public static void main(String[] args) {
            System.setProperty("webdriver.gecko.driver","D:\\selenium_drivers\\geckodriver.exe");
            WebDriver driver = new FirefoxDriver();
            driver.get("http://demo.guru99.com/test/facebook.html");
            List <WebElement> list = driver.findElements(By.className("inputtext"));
            System.out.println(list.size());
                        for(int i = 0; i < list.size(); i++)
                        {
                                    System.out.println(list.get(i).getAttribute("name"));
                        }
            }
}
```

Locating by CSS Selectors

Locating an element on the web page can be challenging, especially in the current scenarios, where every other web page is dynamically programmed and creates/renders the web elements on a need

basis dynamically. It makes finding a unique static attribute for a dynamic element quite tricky. As most of the time, these elements don’t have consistent attribute values. Hence directly using locators like id, name, link, partial link, etc. will not be possible. But if we see, locating the correct web element is the pre-requisite of creating any UI based automation test script, specifically a Selenium automation script.
CSS is powerful enough to identify most of the web element present on the web page. It can also identify elements that don’t have constant attribute values. The CSS Selectors mainly use the character sequence pattern, which identifies the web elements based on their HTML structure.
Locating an element using CSS selector may seem a little difficult than using attributes like id, name, link, etc. but it’s one of the most efficient strategies to locate dynamic elements that don’t have consistent HTML attributes.
Syntax:
tagName[attributename=attributeValue]
Example 1: input[id=email]
Example 2: input[name=email][type=text]

How to create a CSS Selector using ID attribute?
We can use the ID in CSS Selector to identify and locate a web element. In CSS Hash(#) character refers to Id attribute of the web element.
On the demo page "https://demoqa.com/automation-practice-form", we will be trying to find the “first name ” textbox. If we inspect the element, we can see that the HTML tag of the element is “input”, and the value of the “id ” attribute is “firstName”.

How to create a CSS Selector using class attribute?
The class attribute of the HTML tags can also identify the elements on a Web Page. In CSS Dot (.) character refers to Class attribute of the web element.
Suppose we take an example, where we will look at the “Current Address” Textarea element on the page https://demoqa.com/automation-practice-form.
If we look at the HTML tag, we can see that the element has a class attribute, which we can use with a CSS Selector to identify the element.
Sometimes there are multiple classes used for the single element. For example,

Below are the examples to work with classes. If you observe, we have combined multiple classes to work. As the class is not unique like ID, we may require to join two classes and find the accurate element.
Example 1: css=.primary-btn
Example 2: css=.btn.primary-btn
Example 3: css=.submit.primary-btn

How to create a CSS Selector using other attributes?
Apart from the id and class attributes, all other attributes present within the HTML tag of the element can also be used to locate web elements using the CSS Selectors. Let’s take an example in the below screenshot, as we can see that the HTML tag ("textarea") has several different attributes like placeholder, rows, col, id, and class. For this example, we will use the “placeholder ” attribute.
So, by following the generic syntax for the CSS Selector, we can quickly come up with the expression for the text area, i.e., “Current Address ” element:
textarea[placeholder='Current Address']

How to combine multiple CSS selectors?
Let's see how we can combine multiple attributes of the web element to create a CSS Selector for the element:
Combine the ID and other Attributes of the web element to create a CSS Selector
In the given element, the HTML structure contains a textarea tag, id, and placeholder attribute.
We will use these together to create a CSS Selector statement that can easily recognize that element. So, the CSS Selector element for the above elements will be:

textarea#currentAddress[placeholder='Current Address']

- We started with the HTML tag, i.e., textarea
- Then we used the symbol for ID, i.e. "#"
- Then we provided the value of the id attribute.
- In the end, inside the square bracket, we provided the placeholder attribute and its value.

Combine the Class and other Attributes of the web element to create a CSS Selector

Using the same example as above, we have an HTML structure containing textarea tag, class, and placeholder attribute. Subsequently, we can use them together to create a CSS Selector for locating the web element, as shown below:

textarea.form-control[placeholder='Current Address']

We started with the HTML tag, i.e., textarea
Then we used the symbol for class, i.e., '.' Or dot
Then we provided the value of the class attribute.
In the end, inside the square bracket, we provided the placeholder attribute and its value.

How to use a CSS selector for locating dynamic web elements?

With the ever-growing new technologies of web development, multiple times, the web pages are created dynamically. So, there will be various scenarios when certain web elements appear on the web page based on specific conditions/action only, and there is no direct locator available for those elements.

For locating such dynamic web elements, Selenium provides various strategies, such as:

- Locating the element using Parent/Child hierarchy
- Locating the element using text Strings

Locating the element using Parent/Child hierarchy

Selenium provides the capability to locate an element in the HTML DOM using its relativity with other HTML elements.

It provides two strategies to locate the web elements related to other HTML elements:

- One element is the direct parent/child of another element.
- One element exists in the hierarchy of another element.

One element is the direct parent/child of another element.

CSS Selectors allow you to select an element by using the locator of the parent element and then moving to the child element. The CSS Selector for locating the child element can be syntactically represented as follows:

Parent_locator > child_locator

For example inspect textarea element. We have a "textarea " HTML tag which is the child tag of "div".

we can identify its parent HTML tag, then we can use it to access the child tag. Let's create the CSS selector for locating the textarea element:

div>textarea[placeholder='Current Address']

Here we have first used the locator for a parent then "> " followed by the child locator.

Similarly, this can be extended to the sub child also by adding another "> " followed by another locator.

Locate a web element when the element exists in the hierarchy

Similar to the child and sub-child, we can also use a CSS Selector to select the nth-child of an HTML tag. It is quite useful in recognizing list elements or in scenarios where a parent has multiple child elements with non-consistent attributes.

The syntax for locating the nth-child will be:

Parent CSS locator > Child HTML tag : nth-of-type(index)

Selecting nth-child using CSS Selector, for this, we will be using the following site link: https://www.demoqa.com/select-menu.

Say, we want to find the 2nd child element of the "ul" then the CSS Selector expression for the same will be:

select#oldSelectMenu>option:nth-of-type(2)

Here we started with the parent CSS Selector tag, followed by “>” which is followed by the HTML tag of the child. The child HTML tag is then followed by id symbol -“ : ”, which is followed by “nth-of-type(index)” where bracket accepts the index of the required element.

So this way, we can locate any of the HTML elements in the hierarchy.

How to locate a web element using text strings? E.g locate username text element

using the starting text

The Symbol for representing the starting text of a string is: ‘^’

Using this symbol in the CSS Selector, the expression for locating the web element will be:

input[id^='userN']

using the Ending text

The Symbol for representing the ending text of a string is: ‘$’

Using this symbol in the CSS Selector, the expression for locating the web element will be:

input[id$='ame']

using the contains text

The Symbol for representing the contains the text: ‘’*

Using the same symbol in CSS Selector, the expression for the above elements will be:

input[id*='erNa']

Locating Elements using Xpath

XPath in Selenium is an XML path used for navigation through the HTML structure of the page. It is a syntax or language for finding any element on a web page using XML path expression. XPath can be used for both HTML and XML documents to find the location of any element on a webpage using HTML DOM structure.

The basic format of XPath in selenium is explained below with screen shot.

Types of X-path

There are two types of XPath:

- Absolute XPath
- Relative Xpath

You can practise the following XPath exercise on this http://demo.guru99.com/test/selenium-xpath.html.

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.

Below is the example of an absolute xpath expression of the element

/html/body/div[2]/div[1]/div/h4[1]/b/html[1]/body[1]/div[2]/div[1]/div[1]/h4[1]/b[1]

Relative Xpath:

Relative Xpath starts from the middle of HTML DOM structure. It starts with double forward slash (//).

It can search elements anywhere on the webpage, means no need to write a long xpath and you can start from the middle of HTML DOM structure. Relative Xpath is always preferred as it is not a complete path from the root element. Below is the example of a relative XPath expression of the same element

//div[@class='featured-box cloumnsize1']//h4[1]//b[1]

What are XPath axes.

XPath axes search different nodes in XML document from current context node. XPath Axes are the methods used to find dynamic elements, which otherwise not possible by normal XPath method having no ID , Classname, Name, etc. Axes methods are used to find those elements, which dynamically change on refresh or any other operations. There are few axes methods commonly used in Selenium Webdriver like child, parent, ancestor, sibling, preceding, self, etc.

Basic XPath:

XPath expression select nodes or list of nodes on the basis of attributes like ID , Name, Classname, etc. from the XML document as illustrated below.

```
Xpath=//input[@name='uid’]
```

Some more basic xpath expressions:

```
Xpath=//input[@type='text']
Xpath=//label[@id='message23']
Xpath=//input[@value='RESET']
Xpath=//*[@class='barone']
Xpath=//a[@href='http://demo.guru99.com/']
Xpath= //img[@src='//cdn.guru99.com/images/home/java.png']
```

Contains():
Contains() is a method used in XPath expression. It is used when the value of any attribute changes dynamically, for example, login information. The contain feature has an ability to find the element with partial text as shown in below XPath example. In this example, we tried to identify the element by just using partial text value of the attribute. In the below XPath expression partial value ‘sub’ is used in place of submit button. It can be observed that the element is found successfully.
Complete value of ‘Type’ is ‘submit’ but using only partial value ‘sub’.

```
Xpath=//*[contains(@type,'sub')]
```

Using OR & AND:
In OR expression, two conditions are used, whether 1st condition OR 2nd condition should be true. It is also applicable if any one condition is true or maybe both. Means any one condition should be true to find the element.
In the below XPath expression, it identifies the elements whose single or both conditions are true.

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

Highlighting both elements as “LOGIN ” element having attribute ‘type’ and “RESET” element having attribute ‘name’.

In AND expression, two conditions are used, both conditions should be true to find the element.
It fails to find element if any one condition is false.

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

In below expression, highlighting ‘LOGIN’ element as it having both attribute ‘type’ and ‘name’.

Xpath Starts-with
XPath starts-with() is a function used for finding the web element whose attribute value gets changed on refresh or by other dynamic operations on the webpage.
In this method, the starting text of the attribute is matched to find the element whose attribute value changes dynamically.
You can also find elements whose attribute value is static (not changes).
For example -: Suppose the ID of particular element changes dynamically like:

```
Id=” message12″
Id=” message345″
Id=” message8769″
```

and so on.. but the initial text is same. In this case, we use Start-with expression.

XPath Text() Function
The XPath text() function is a built-in function of selenium webdriver which is used to locate elements based on text of a web element.
It helps to find the exact text elements and it locates the elements within the set of text nodes.
The elements to be located should be in string form.
In this expression, with text function, we find the element with exact text match as shown below. In our case, we find the element with text “UserID”.

```
Xpath=//td[text()='UserID']
```

XPath axes methods:

These XPath axes methods are used to find the complex or dynamic elements. Below we will see some of these methods. For illustrating these XPath axes method, we will use the Guru99 bank demo site.

a) Following:

Selects all elements in the document of the current node( ) [UserID input box is the current node] as shown in the below screen.

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

There are 3 “input” nodes matching by using “following” axis- password, login and reset button.

If you want to focus on any particular element then you can use the below XPath method:

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

You can change the XPath according to the requirement by putting [1],[2]…………and so on.

With the input as ‘1’, the below screen shot finds the particular node that is ‘Password’ input box element.

b) Ancestor:

The ancestor axis selects all ancestors element (grandparent, parent, etc.) of the current node as shown in the below screen. In the below expression, we are finding ancestors element of the current node(“ENTERPRISE TESTING” node).

Xpath= //*[text()='Enterprise Testing']//ancestor::div

There are 3 “div” nodes matching by using “ancestor” axis. If you want to focus on any particular element then you can use the below XPath, where you change the number 1, 2 as per your requirement:

Xpath=//*[text()='Enterprise Testing']//ancestor::div[1]

You can change the XPath according to the requirement by putting [1], [2]……and so on.

c) Child:

Selects all children elements of the current node (Java) as shown in the below screen.

Xpath= //*[@id='java_technologies']//child::li

There are 71 “li” nodes matching by using “child” axis. If you want to focus on any particular element then you can use the below xpath:

Xpath=//*[@id='java_technologies']//child::li[1]

You can change the xpath according to the requirement by putting [1],[2]… & so on.

d) Preceding:

Select all nodes that come before the current node as shown in the below screen. In the below expression, it identifies all the input elements before “LOGIN” button that is Userid and password input element.

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

There are 2 “input” nodes matching by using “preceding” axis. If you want to focus on any particular element then you can use the below XPath:

Xpath=//*[@type='submit']//preceding::input[1]

You can change the xpath according to the requirement by putting [1],[2]…………and so on.

e) Following-sibling:

Select the following siblings of the context node. Siblings are at the same level of the current node as shown in the below screen. It will find the element after the current node.

xpath=//*[@type='submit']//following-sibling::input

One input nodes matching by using “following-sibling” axis.

f) Parent:

Selects the parent of the current node as shown in the below screen.

Xpath=//*[@id='g-utility']//parent::div

There are 65 “div” nodes matching by using “parent” axis. If you want to focus on any particular element then you can use the below XPath:

Xpath=//*[@id='g-utility']//parent::div[1]

You can change the XPath according to the requirement by putting [1],[2]…………and so on.

g) Descendant:

Selects the descendants of the current node as shown in the below screen.

In the below expression, it identifies all the element descendants to current element ('Main body surround' frame element) which means down under the node (child node , grandchild node, etc.).

```
Xpath=//*[@id='g-utility']//descendant::a
```

There are 12 "link" nodes matching by using "descendant" axis. If you want to focus on any particular element then you can use the below XPath:

```
Xpath=//*[@id='g-utility']//descendant::a[1]
```

You can change the XPath according to the requirement by putting [1],[2]……and so on.

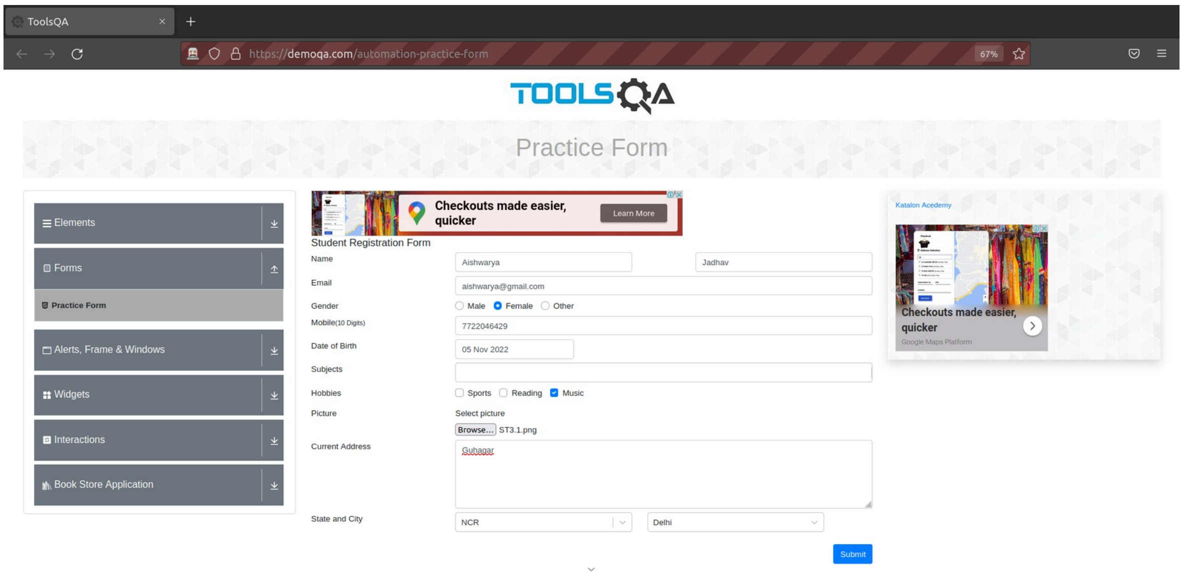
Implementation:

1. **Write a selenium script to use css locators to find elements on the following wepage https://demoqa.com/automation-practice-form.**

```
 package webdriver_scripts;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;

public class Exercise1 {

	public static void main(String[] args) {
		System.setProperty("webdriver.gecko.driver", "/home/aishu/Apps/geckodriver");
		WebDriver driver=new FirefoxDriver();
		driver.get("https://demoqa.com/automation-practice-form");

		WebElement
firstName=driver.findElement(By.cssSelector("input[id='firstName']"));
		firstName.sendKeys("Aishwarya");

		WebElement
lastName=driver.findElement(By.cssSelector("input[id='lastName']"));
		lastName.sendKeys("Jadhav");

		WebElement mail=driver.findElement(By.cssSelector("input[id='userEmail']"));
		mail.sendKeys("aishwarya@gmail.com");

		WebElement    gender=driver.findElement(By.cssSelector("label[for='gender-radio-
2']"));
		gender.click();

		WebElement
mob=driver.findElement(By.cssSelector("div>input[id='userNumber']"));
		mob.sendKeys("7722046429");

		WebElement
sbject=driver.findElement(By.cssSelector("input[id='subjectsInput']"));
		sbject.sendKeys("Software Testing");

		WebElement           hobby=driver.findElement(By.cssSelector("label[for='hobbies-
checkbox-3']"));
		hobby.click();

	WebElement uploadElement = driver.findElement(By.id("uploadPicture"));
	uploadElement.sendKeys("//home/aishu/Downloads/Aish.jpg");
```

```
                    WebElement
currentAddress=driver.findElement(By.cssSelector("textarea[class='form-control']"));
                    currentAddress.sendKeys("Guhagr");


          }

}
```

Output:



ToolsQA
https://demoqa.com/automation-practice-form

Practice Form

Student Registration Form

Name: Aishwarya | Jadhav
Email: aishwarya@gmail.com
Gender: Male [✓ Female] Other
Mobile(10 Digits): 7722046429
Date of Birth: 05 Nov 2022
Subjects:
Hobbies: Sports Reading [✓ Music]
Picture: Select picture — Browse... ST3.1.png
Current Address: Guhagar
State and City: NCR | Delhi

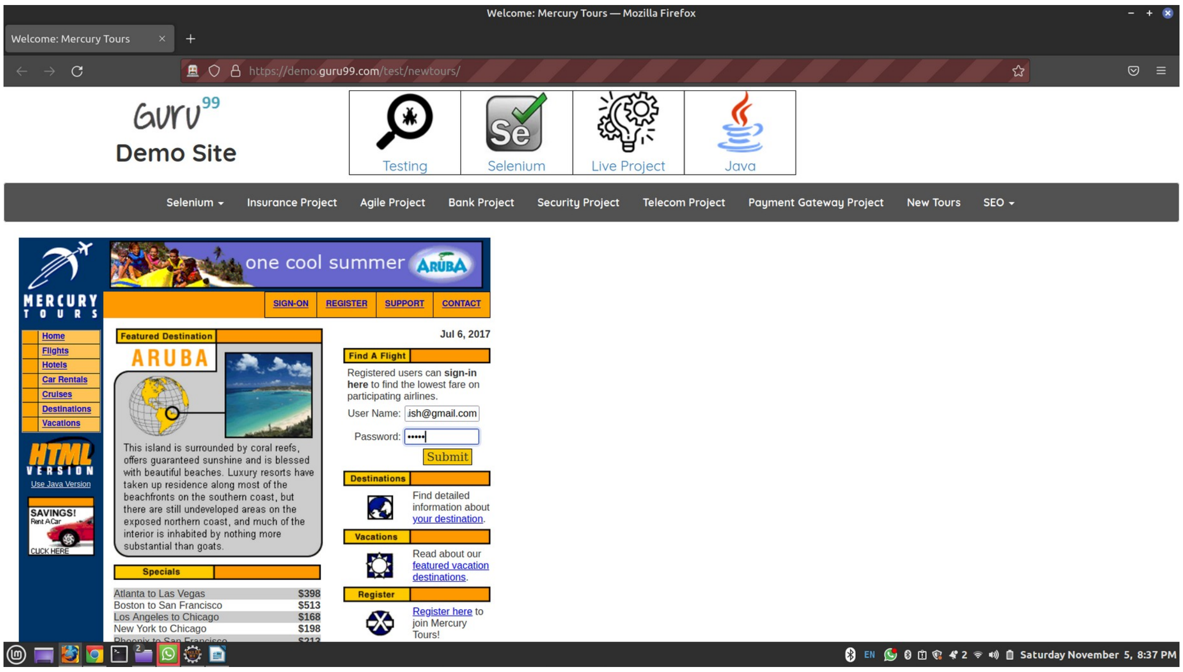
Submit

2. **Write a selenium script to find elements on http://demo.guru99.com/test/newtours/ using id,name, link, partial link locators**.

```
 package webdriver_scripts;

 Import org.openqa.selenium.By;
 import org.openqa.selenium.WebDriver;
 import org.openqa.selenium.WebElement;
 import org.openqa.selenium.firefox.FirefoxDriver;

 public class Exercise2 {

 public static void main(String[] args) {
          System.setProperty("webdriver.gecko.driver", "/home/aishu/Apps/geckodriver");
          WebDriver driver=new FirefoxDriver();
          driver.get("https://demo.guru99.com/test/newtours/");

          WebElement user=driver.findElement(By.cssSelector("input[name='userName']"));
          user.sendKeys("aish@gmail.com");

          WebElement pass=driver.findElement(By.cssSelector("input[name='password']"));
          pass.sendKeys("12345");

          WebElement register_link=driver.findElement(By.linkText("REGISTER"));
          System.out.println(register_link.getText());
```

```
                WebElement signon_link=driver.findElement(By.linkText("SIGN-ON"));
                System.out.println(signon_link.getText());

                WebElement contact_link=driver.findElement(By.linkText("CONTACT"));
                System.out.println(contact_link.getText());

                WebElement support_link=driver.findElement(By.linkText("SUPPORT"));
                System.out.println(support_link.getText());



        }

}
```

Output:



This island is surrounded by coral reefs, offers guaranteed sunshine and is blessed with beautiful beaches. Luxury resorts have taken up residence along most of the beachfronts on the southern coast, but there are still undeveloped areas on the exposed northern coast, and much of the interior is inhabited by nothing more substantial than goats.

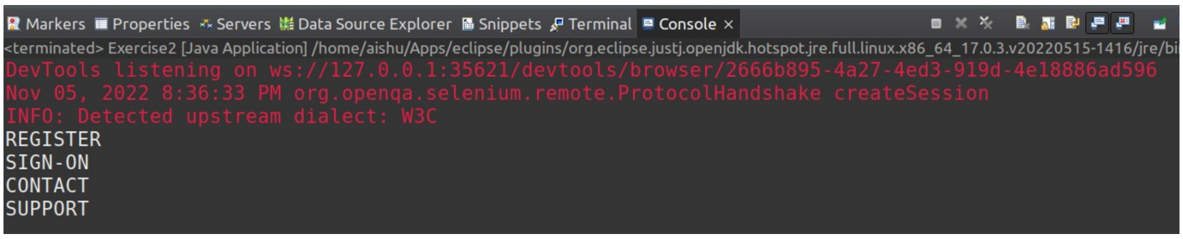
Registered users can **sign-in here** to find the lowest fare on participating airlines.

Find detailed information about your destination.

Read about our featured vacation destinations.

Register here to join Mercury Tours!

Atlanta to Las Vegas	$398
Boston to San Francisco	$513
Los Angeles to Chicago	$168
New York to Chicago	$198

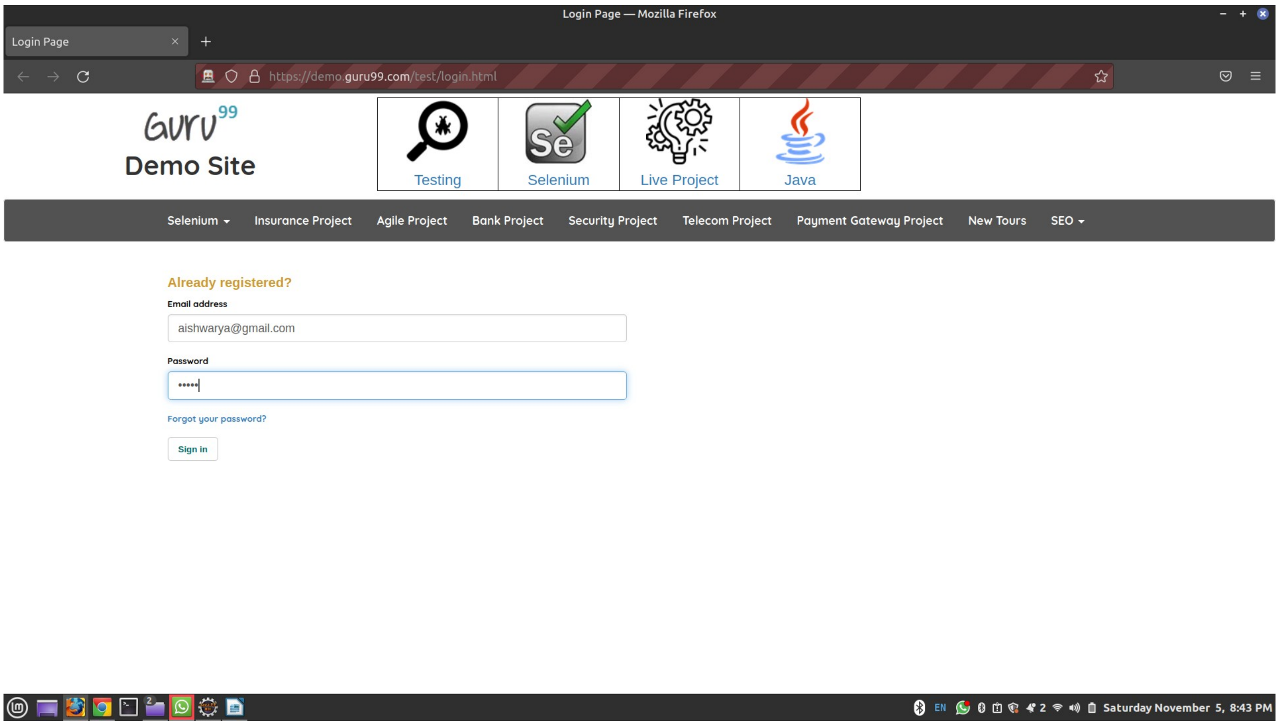


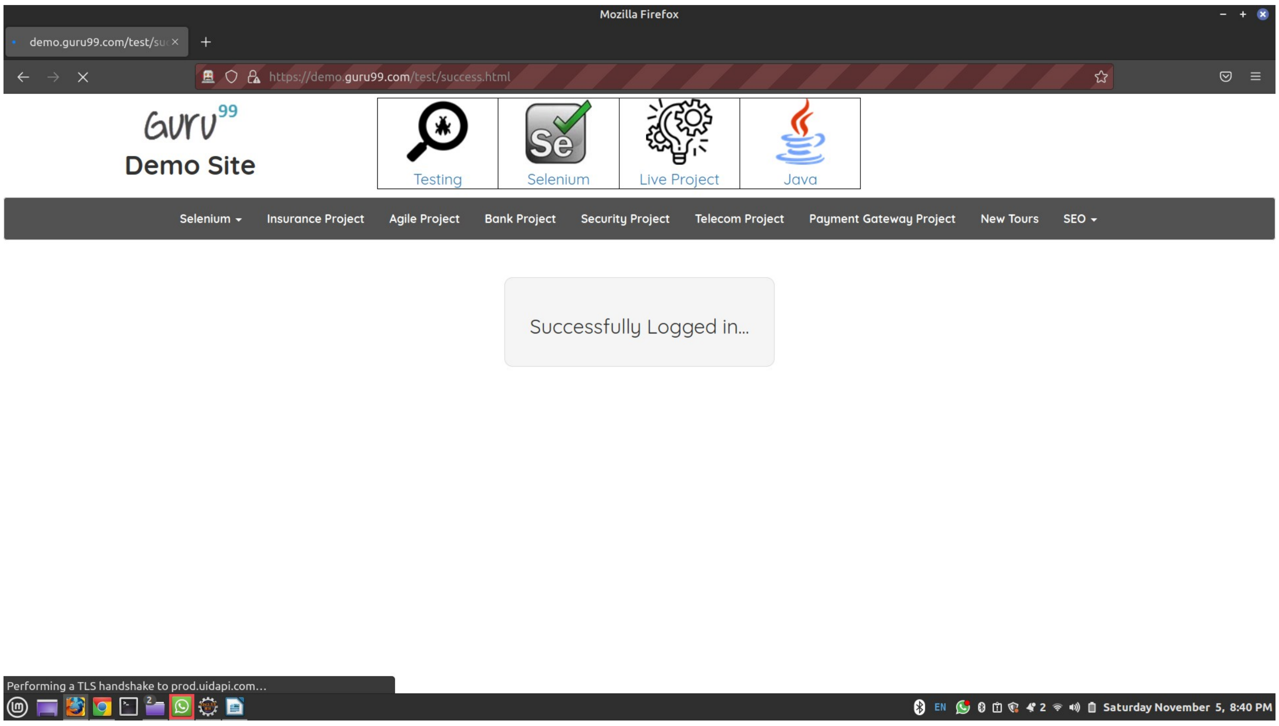
```
<terminated> Exercise2 [Java Application] /home/aishu/Apps/eclipse/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.linux.x86_64_17.0.3.v20220515-1416/jre/bi
DevTools listening on ws://127.0.0.1:35621/devtools/browser/2666b895-4a27-4ed3-919d-4e18886ad596
Nov 05, 2022 8:36:33 PM org.openqa.selenium.remote.ProtocolHandshake createSession
INFO: Detected upstream dialect: W3C
REGISTER
SIGN-ON
CONTACT
SUPPORT
```

3. Write a selenium script to login to http://demo.guru99.com/test/login.html. Use name and ID locator strategy to locate elements.(Paste your code here)

```
package webdriver_scripts;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;

public class Exercise3 {

public static void main(String[] args) {
        System.setProperty("webdriver.gecko.driver", "/home/aishu/Apps/geckodriver");
        WebDriver driver=new FirefoxDriver();
        driver.get("https://demo.guru99.com/test/login.html");

        WebElement email=driver.findElement(By.cssSelector("input[name='email']"));
        email.sendKeys("aishwarya@gmail.com");

        WebElement pass=driver.findElement(By.cssSelector("input[id='passwd']"));
        pass.sendKeys("12345");

        WebElement button=driver.findElement(By.name("SubmitLogin"));
        button.click();

}

}
```

Output:





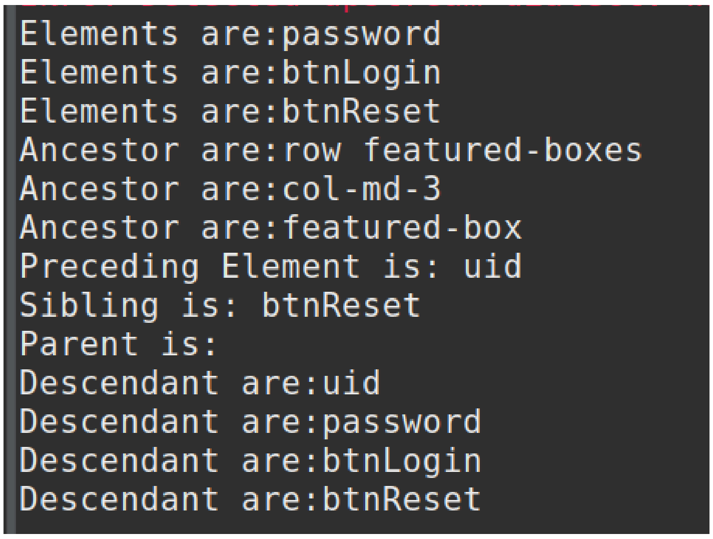
Successfully Logged in...

4. **Write a script to locate elements on https://demo.guru99.com/test/selenium-xpath.html using xpath.**

```
package webdriver;

package webdriver_scripts;

import java.util.List;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;

public class ByXpath {

public static void main(String[] args) {
        System.setProperty("webdriver.gecko.driver", "/home/aishu/Apps/geckodriver");
        WebDriver driver=new FirefoxDriver();
        driver.get("https://demo.guru99.com/test/selenium-xpath.html");

        List<WebElement>
in=driver.findElements(By.xpath("//*[@type='text']//following::input"));
        for (int i=0; i<in.size();i++){
            System.out.println("Elements are:" + in.get(i).getAttribute("name"));
        }

        List<WebElement> ancestor=driver.findElements(By.xpath("//*[text()='Enterprise
Testing']//ancestor::div"));
        for (int i=0; i<ancestor.size();i++){
            System.out.println("Ancestor are:" + ancestor.get(i).getAttribute("class"));
```

```
                    }

                    WebElement
preceding=driver.findElement(By.xpath("//*[@type='submit']//preceding::input[2]"));
                    System.out.println("Preceding Element is: "+preceding.getAttribute("name"));

                    WebElement
sibling=driver.findElement(By.xpath("//*[@type='submit']//following-sibling::input"));
                    System.out.println("Sibling is: "+sibling.getAttribute("name"));

                    WebElement
parent=driver.findElement(By.xpath("//*[@type='submit']//parent::td"));
                    System.out.println("Parent is: "+parent.getText());

                    List<WebElement>
descendant=driver.findElements(By.xpath("//*[@name='frmLogin']//descendant::input"));
                    for (int i=0; i<descendant.size();i++){
                         System.out.println("Descendant are:" +
descendant.get(i).getAttribute("name"));
                    }
```

Output:



```
Elements are:password
Elements are:btnLogin
Elements are:btnReset
Ancestor are:row featured-boxes
Ancestor are:col-md-3
Ancestor are:featured-box
Preceding Element is: uid
Sibling is: btnReset
Parent is:
Descendant are:uid
Descendant are:password
Descendant are:btnLogin
Descendant are:btnReset
```

Conclusion: Gained knowledge about selenium tools and learnt to locate various web elements using selenium locators.

After performing this Practical/lab, students are expected to answer following questions
Q.1. What is selenium webdriver?
Q.2. Which drivers are required for running test on chrome and firefox?
Q.3. What are different CSS locators?
Q.4. Which characters represent attributes – ID and Class?
Q.5. What are xpath? How to differentiate absolute xpath and relative xpath?