Syllabus:

1. Why Automation
2. Selenium Set Up
   1. Java, Maven,Firefox & Extensions,
3. Locators
4. Waits
5. Operations on web Elements
   1. Sendkeys, Click and dropdown selection
6. Locating elements using Text with Xpath
7. findElement&findElements
   1. List
8. Actions Class
   1. Multiple Actions
9. File Upload and download
   1. AutoIt
10. Alert & popup handling
11. Framework – Page Object Model

Websites to Practice:

automationpractice.com  
http://www.seleniumframework.com/Practiceform/  
<https://www.makemytrip.com/>

<https://www.reduceimages.com/>

<http://toolsqa.com/automation-practice-form/>

Automation Testing?

Why Automation Testing?

What is Selenium

Set Up Process.

Install JDK<http://www.oracle.com/technetwork/java/javase/downloads/index.html>  
Setup Environment Variables

Install Eclipse<https://www.eclipse.org/downloads/>

Install Maven <https://maven.apache.org/download.cgi>   
Setup Env variables

Install FireFox<https://filehippo.com/download_firefox/>  
Version must be old, 41 looks good

Add below AdOn to firefox  
Firebug, FirePath, Xpath Checker(Not available), Nightly Tester Tools

As Xpath Checker is no longer available, need to install from external file.  
Then to ignore compatibility check, need to install Nightly Tester Tools

Geckodriver, Chromedriver (Just download)<https://www.seleniumhq.org/download/>

Purpose of Gecko / ChromeDriver

**Gecko Driver** is the link between your tests in Selenium and the Firefox browser. GeckoDriver is a proxy for using W3C WebDriver-compatible clients to interact with Gecko-based browsers i.e. Mozilla Firefox in this case. As Selenium 3 will not have any native implementation of FF, we have to direct all the driver commands through Gecko Driver. Gecko Driver is an executable file that you need to have in one of the system path before starting your tests. Firefox browser implements the WebDriver protocol using an executable called **GeckoDriver.exe**. This executable starts a server on your system. All your tests communicate to this server to run your tests. It translates calls into the Marionette automation protocol by acting as a proxy between the local and remote ends

Locators in Selenium:

**id** : WebElement elementname=driver.findElement(By.id("username"));-

**Name**  : WebElement elementname = driver.findElement(By.name("login"));

**Xpath**  :

xpath=//\*[@id=’username’]

xpath=//input[@id=’username’]

xpath=//form[@name=’loginForm’]/input[1]

xpath=//\*[@name=’loginForm’]/input[1]

//a[text()='Contact us']

WebElement elementname = driver.findElement(By. xpath("//input[@id=’username’]"));

Starts with

**“//input[starts-with(text(),’Hel’)]”**

**Contains:**

**“//input[Contains(text(),’ell’)]”**

**Linktext** : WebElement element=driver.findElement(By.linkText("Contact us")).click();

**Partial Link Text:** driver.findElement(By.partialLinkText("Contact")).click();

**Tag Name**  : driver.findElement(**By**.tagName("select")) - Not popular. Try with select (dropdown)

**Class name**  : WebElement elementname =driver.findElement(By.className(“sample”)); --

**DOM** :

* getElementById
* getElementsByName
* dom:name
* dom: index

**CSS**:

When we don't have an option to choose Id or Name, we should prefer using CSS locators as the best alternative.  
CSS is "Cascading Style Sheets" and it is defined to display HTML in structured and colorful styles are applied to webpage.

Selectors are patterns that match against elements in a tree, and as such form one of several technologies that can be used to select nodes in an XML document.

* CSS has more Advantage than Xpath
* CSS is much faster and simpler than the Xpath.

1. **ID -: #idValue**  
   <button id="submitButton1" type="button" class="btn">Submit</button>  
    Answer: # submitButton1
2. **Class :.classvalue**  
   <button id="submitButton1" type="button" class="btn">Submit</button>  
   Answer : .btn
3. Tag:tagname  
   <input id="fname" type="text" name="firstName" class="textbox">  
   Answer: input
4. **Attribute & Value:[attribute=’value’]**  
   <input id="fname" type="text" name="firstName" class="textbox">  
   Answer: [name='firstName']
5. **Tag and ID :  tag#id**  
   <input id="fname" type="text" name="firstName" class="textbox">  
    Answer :input#fname
6. **Tag and class : tag.class**  
   <input id="fname" type="text" name="firstName" class="textbox">  
   Answer: input.textbox
7. **Tag and attribute :tag[attributeName='attributeValue']**  
   <input id="fname" type="text" name="firstName" class="textbox">  
   Answer :  input[name='firstName']
8. Tag, class, and attribute : css=tag.classvalue[attribute=value]
9. Inner text

Operation on Elements:

Sendkeys – To send values to textbox  
Click - Click on button, links etc

**Selecting Values on DropDown**

WebElement eleDrp= driver.findElement(By.*id*("days"));

Select osel =**new** Select(eleDrp);

osel.selectByValue("12");

By.Id and ById

Both of these methods are from the RemoteWebDriver class.  
  
findElement(By.id()) requires you to created your own By.id instance.  
findElementById(String) is a helper function that will generate the By.id  
instance for you.  
  
It comes down to providing you with the flexibility to choose what you want  
to keep track of in your tests/framework.  
Do you want to track locators as Strings?  
Do you want to track locators as By objects?

Tables:

findElement&findElements

Finding & Checking all CheckBoxes.

List<WebElement>lstCheck=driver.findElements(By.*xpath*("//input[@type='checkbox']"));

**for**(**int**i=0;i<lstCheck.size();i++)

{

lstCheck.get(i).click();

}

Actions Class

Actions builder =**new** Actions(driver);

builder.moveToElement(btnCreateAccuount).perform();

btnCreateAccuount.click();

File Upload:

This can be achieved by locating upload button and sending the filepath using sendkeys.

WebElement btnUpload= driver.findElement(By.id("ctl00\_Main\_ctl01\_fuImage"));

btnUpload.sendKeys("F:\\CSS1\\css\\Image.jpg");

File Download.

This will pop up a Save dialog, which cant be handled by selenium.  
So we have to use thirdparty tools like AutoIt, Wget, Sikuli, AutoHotKey

After clicking on the download button just run AutoIT Script.  
Download AutoIT & Install.  
Write below script for Just Save Cancel Popup in AutoIT and Compile which will create an exe file

WinWait("[CLASS:#MozillaDialogClass]","",5)  
 Send("{LEFT}")  
 Sleep(10)  
 Send("{ENTER}")

Wait for 5 seconds to appear download and save dialog.   
Used class property of download dialog.  
*Wait 5 seconds for the Dialog to appear.*

WinWait (“title" [, "text" [, timeout = 0]])

|  |  |
| --- | --- |
| title | The title/hWnd/class of the window to check. See [Title special definition](https://www.autoitscript.com/autoit3/docs/intro/windowsadvanced.htm). |
| text | **[optional]** The text of the window to check. Default is an empty string. See [Text special definition](https://www.autoitscript.com/autoit3/docs/intro/windowsbasic.htm#specialtext). |
| timeout | **[optional]** Timeout in seconds if the window does not exist. Default is 0 (no timeout). |

Call the .exe file from script  
Runtime.*getRuntime*().exec("F:\\CSS1\\css\\download.exe");

Switching to Child window: