Introduction to Selenium

Manual Testing

Performing the testing without using any tools is known as Manual Testing. Here a tester / person will do the testing.

Automation Testing

Performing the testing with the help of tools is known as Automation Testing. Here instead of a person, a tool will perform the testing. The tool used for replacing the person is nothing but a Test Automation tool.

How can a tool perform Testing ?

Testers will do some coding and put the developed code into the Automation tool. Automation tool will perform testing with the help of the code developed by the testers.

Selenium

Selenium in simple terms is a Test Automation Tool. i.e. Instead of a person performing testing manually, Selenium tool will automatically perform the testing with the help of developed code.

Type of Applications that Selenium can automate

Selenium can only automate Web Applications (i.e. The applications which run on the browser).

Example: https://www.msn.com

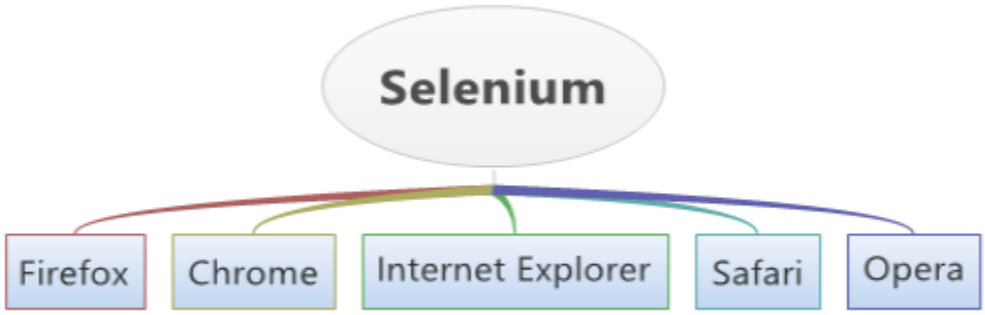
Selenium cannot automate Desktop Applications or Mobile Applications (i.e. The applications which dont run on the browsers).

Example: Skype, Adobe PDF Reader and many more.

Selenium is a free tool

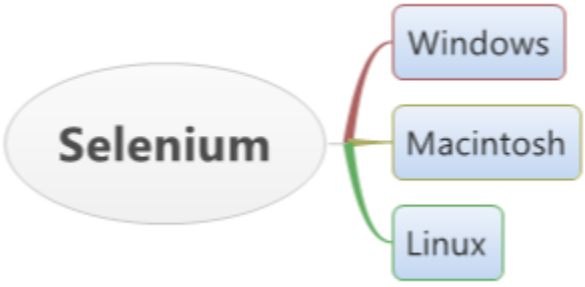
Selenium is a free tool. (i.e. You don't need to pay any license amount for using Selenium tool.)

Different Browsers supported by Selenium

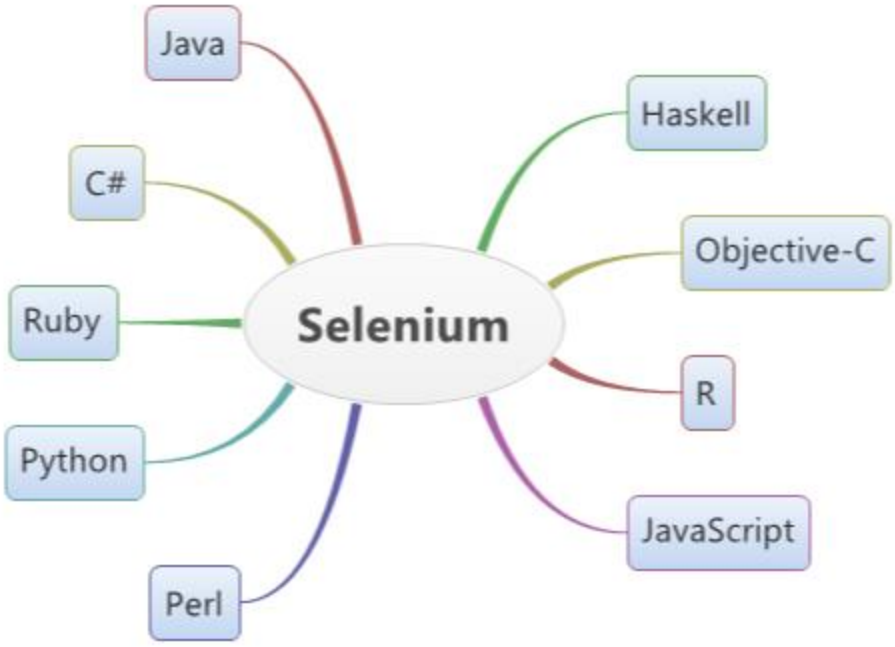


Note: Edge Browser is also added to this list.

Different Operating Systems supported by Selenium

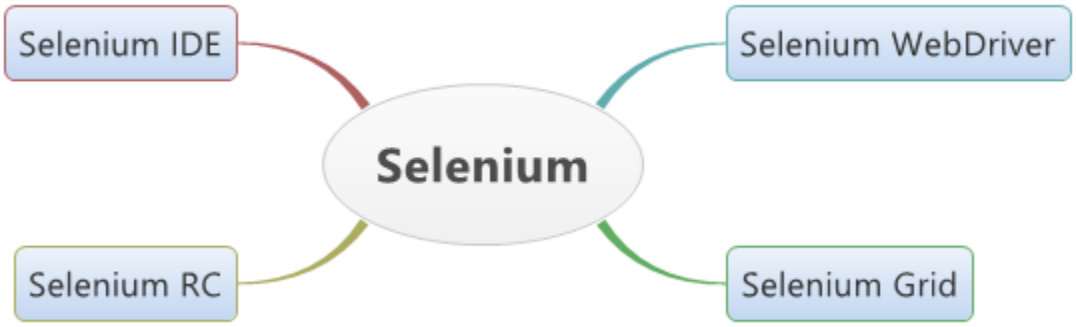


Different Programming Languages supported by Selenium



Different Components in Selenium

Selenium is not a single tool. Instead it is a combination of tools like Selenium IDE, Selenium RC, Selenium WebDriver and Selenium Grid. Out of all these tools, Selenium RC is outdated and can be ignored completely.



Different versions of Selenium

So far, Selenium got released into the market in three different versions i.e. Selenium 1, Selenium 2 and Selenium 3. Selenium 1 is the older version and Selenium 3 is the latest version.

As per todays date, the Alpha Versions of Selenium 4 are released into the market.

Pre-requisites required for learning Selenium

In order to understand Selenium in a better way, we need to know the below pre-requisites first:

- Locators
- Core Java

I will be explaining the above pre-requisites on the need basis.

Official Website of Selenium

- Selenium.dev

Locators

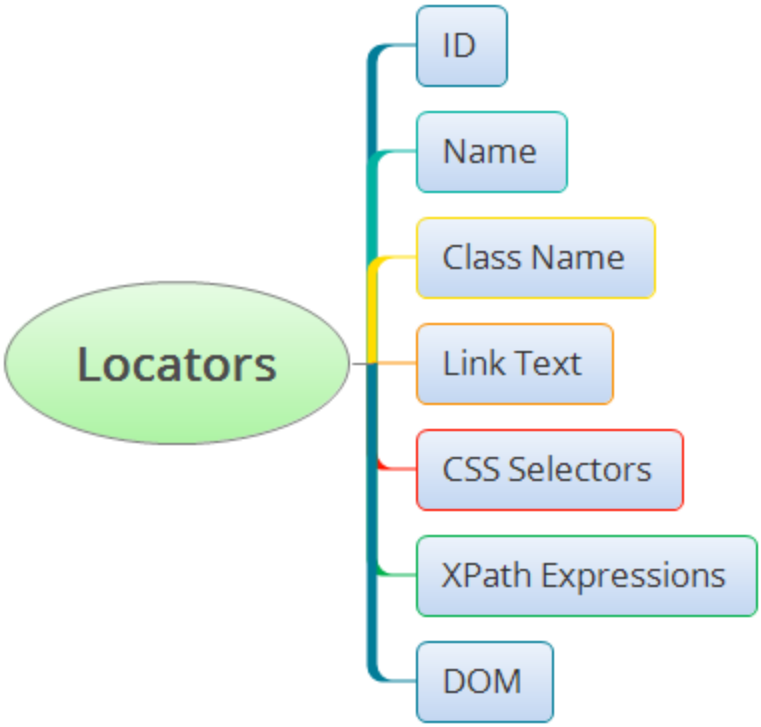
Introduction to Locators

- Locators help Selenium in finding the UI elements on Web pages.

- We can use these locators in our test automation code to locate the UI elements on the Web pages.
- Demonstrate the usage of Locators in locating the UI Elements on the Web Pages using Selenium IDE

Different types of Locators

The below are the different types of locators which can be used for locating the UI elements on the Web Pages:



HTML Basics for Locators

Having knowledge of below HTML basics will help you in understanding the locators better:

1. Take any simple web page
   say http://compendiumdev.co.uk/selenium/basic_web_page.html and inspect it to find its html code
2. Let's decode the above to create our own web page similar to above
   1. Title
   2. Paragraphs
3. Now, lets learn the HTML code for displaying different types of UI elements:
   1. Adding Rulers using <hr/>
   2. Adding Hyperlink (Requires href attribute with value and target attribute with value '_blank')
   3. Adding a Button (Using button tags and id attribute)
4. Inspect different UI elements on Omayo to identify the HTML code behind it
   1. Radio options having name attributes
   2. Drop down option having class attribute
   3. Image having src attribute
   4. Text Area field having some random attributes

Demonstrating Different types of Locators using Selenium IDE

Install Selenium IDE in any supported browser.

With the help of Selenium IDE's Target Text box field, demonstrate all the below different types of locators:

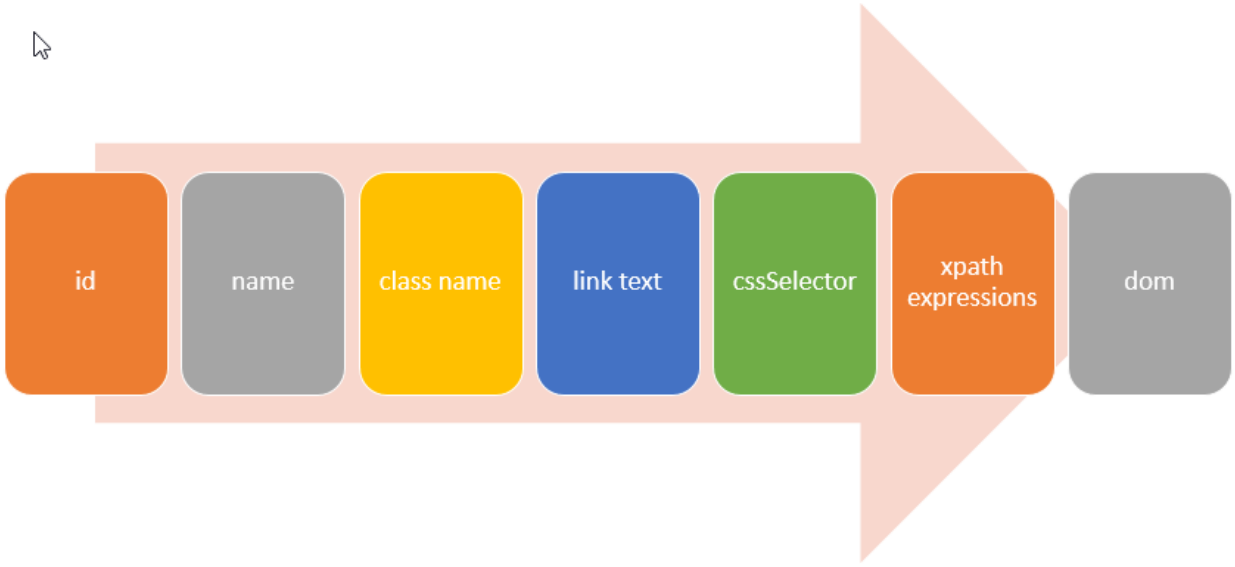
Note: class and dom locators won't work in Selenium IDE.

- **id** locator - Example: 'Button2' button in omayo blog - Syntax: id=but2
- **name** locator - Example: 'Locate using name attribute' text box field in omayo blog - Syntax: name=textboxn
- **class** locator - Example: text box "locate using class" in omayo blog - Syntax: class=classone
    - I will demonstrate using this locator during Selenium WebDriver sessions.
- **link** locator - Example: link 'compendiumdev' in omayo blog - Syntax: link=compendiumdev
- **css** locator - Example: 'Button2' button in omayo blog - Syntax: css=#but2
- **xpath** locator - Example: 'Button2' button in omayo blog - Syntax: xpath=//*[@id='but2']
- **dom** locator - Example: 'Button2' button in omayo blog - Syntax: dom=document.getElementById("but2")

Locators Priority

Though there are different types of locators available for locating the UI elements on the Web Pages, we need to use any one of them based on their priority.

The below is the priority order in which we need to select and use the locators :



- 'id' locator needs to be given the first priority. i.e. If the same UI element can be located with the help of different locators like id, name and so on, we need to choose id locator locating it.
- Similarly second priority goes for 'name' locator. i.e. If the UI element cannot be located by id locator, then we will prefer to choose 'name' locator as second priority.
- Third priority goes for 'class' locator.
- Forth priority goes for 'link text' locator.
- Fifth priority goes for 'cssSelector' locator.
- Sixth priority goes for 'xpath expressions' locator.
- Last priority goes for 'dom' locator.

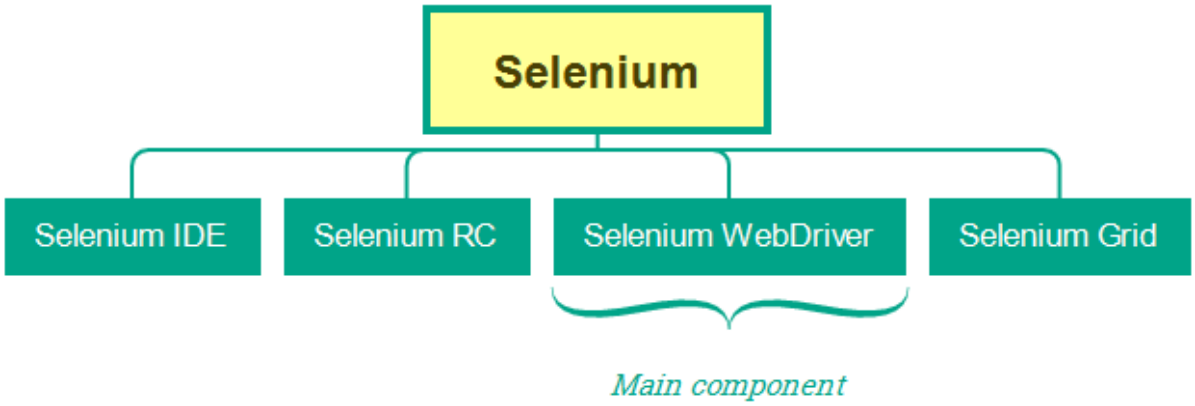
ChroPath for Auto-generating the XPath Expressions and CSS Selectors

1. Installing ChroPath in any supported browser
2. Inspect any element and go to ChroPath tab to auto-generate the XPath Expressions and CSS Selectors
3. Confirm its accuracy using Selenium IDE

WebDriver (Introduction, Downloading, Configuring and WebDriver API Commands - Part 1)

What is Selenium WebDriver?

Selenium WebDriver is the main component of Selenium and requires Core Java knowledge for using it.



Downloading and configuring Selenium WebDriver in Java Projects

Follow the below steps for downloading and configuring the Selenium WebDriver in a Java Project:

1. Create a new Java Project
2. Create a new Class say Demo having main() method
3. Create an object for ChromeDriver and assign to the variable of WebDriver interface
4. Create a new folder say 'library' inside the Project workspace
5. Go to mvnrepository and find the latest stable version of Selenium
6. Download latest stable version of Selenium from Selenium.dev > Previous releases
7. Extract the folder and Paste the available jar files into the library folder and configure the Project
8. Create a new folder 'drivers' and copy the compatible chrome driver
9. Setting the Chrome Driver -
   ```
   System.setProperty ("webdriver.chrome.driver" ,"D:\\SeleniumTrainingWorkspace\\Demo\\drivers\\chromedriver.exe" );
   ```
10. Run the code to see the Chrome Browser launching.
11. How to run the code on other browsers will be explained in the upcoming sessions

Selenium WebDriver API Commands

Selenium WebDriver is an API, which contains a set of predefined library of commands / methods.

- Google Search "Selenium Java WebDriver API" and find the predefined methods of WebDriver interface

Lets start using the Predefined methods of Selenium WebDriver.

- **get()** - Used to open the specified URL's web page - Demonstrate here
  - **http://** needs to be provided, otherwise the specified URL wont get opened
- **manage().window().maximize() -** Used to maximize the current web page - Demonstrate here
- **findElement(), By class and its predefined methods, WebElement interface and its predefined methods**
  - findElement() is a predefined method of WebDriver and it needs to be used with By Class and its predefined methods for locating and performing the operations on UI elements
  - View the 'By' Class and its predefined methods in Selenium API Pages - The purpose of the By class and its predefined methods is to locate the UI elements with the help of the locators
  - **id() - By.id("confirm")**
  - **name() - By.name("q")**
  - **className() - By.className("classone")**
  - **linkText() - By.linkText("compendiumdev")**
  - **partialLinkText() - By.partialLinkText("compendium")**
  - **cssSelector() - By.cssSelector("#confirm")**
  - **xpath() - By.xpath("//input[@id='confirm']")**
  - Store the UI element located by the findElement() to reference variable of **WebElement** interface
- **click()** - Used to perform click operation on different UI elements like Button, link, checkbox option and radio option etc
- **sendKeys() -** Used to enter text into the text fields like text box, text area, password fields etc. - Demonstrate here
- **clear() -** Used to clear the text available in the text box or text area fields - Demonstrate here
- **getText()** - Used to retrieve the elements text (i.e. The text between the starting and ending tags of HTML elements) - Demonstrate here

WebDriver API Commands - Part 2 and Executing Scripts on different browsers

Selenium WebDriver API Commands (Continued)

- **getTitle() -** Used to retrieve the title of the current web page - Demonstrate here
- **getCurrentUrl() -** Used to retrieve the URL of the current web page - Demonstrate here
- **close() -** Used to close the current Browser window - Demonstrate here
- **quit()** - Used to close all the Browser windows (i.e. All the browser windows, including child windows will be closed) - Demonstrate here
- **getAttribute() -** Used to retrieve the value stored in the specified attribute value of the html element - Demonstrate here
- **isDisplayed() -** Used to find out whether the element is displayed on the page (i.e. available on the page) before performing operations on it - Demonstrate here
- **isEnabled() -** Used to find out whether the element is enabled or disabled before performing operation on it - Demonstrate here
- **isSelected() -** Used to find out whether the radio options and checkbox options are selected or not. - Demonstrate here
- **navigate() -** Used to perform operations like navigate back to previous page, navigate forward again or refreshing the current. - Demonstrate here
- **getPageSource()** - Used to retrieve all the source code of the current page and return in the form of String - Demonstrate here
- **submit()** - Used to submit a form - Example: Search text field and Search Button on Omayo
- **getTagName()** - Used to get the html tag of the provided element - Example: find tag name of Search text field
- **getCSSValue()** - `driver.findElement(By.id("home")).getCssValue("line-height");`
- **getSize()** - Used to get the height and width of the given element - `Dimension d = driver.findElement(By.id("but2")).getSize(); - d.height and d.width`
- **getLocation()** - Used to get the x and y coordinate position of the given element - `Point p = driver.findElement(By.id("but2")).getLocation(); - p.x and p.y`
- **fullScreen()** - Used to display the web page in full screen mode - `driver.manage().window().fullscreen();`
- **findElements()** - Used to find and return more than one Web Element
    - **By.tagName()** - Print all the hyperlink texts on the web page

Executing Scripts on other browsers

- Firefox Browser - Using firefox compatible gecko driver
    - Firefox Browser Version - Selenium Version - GeckoDriver Version
- Internet Explorer Browser - IEDriverServer same as WebDriver version should be used
    - Download the same version of IEDriver, matching with Selenium WebDriver version
    - IE Settings > Toolbar > Internet Options > Security > Make sure all checkboxes are not selected

- Edge Browser - Using Edge Driver version same as Edge Browser version
- Execute the sample Selenium script on the specified browser
    - This is the reason why we are assigning the objects of different Classes to the variable of a Parent Interface

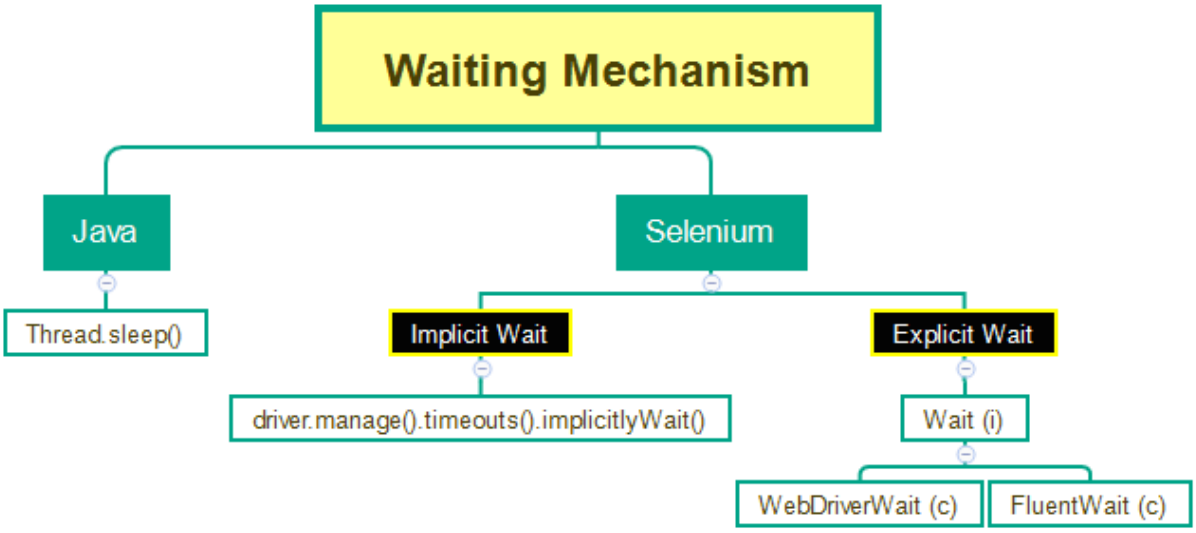
WebDriver - Part 3 (Handling Windows, Alerts, Drop-downs, Multi-Selection Box fields and Waiting Mechanism)

Handling multiple windows

- WebDriver interface has two predefined methods **getWindowHandles()** and **switchTo()** , which helps us in handling multiple browser windows.
- **getWindowHandles() -** Used to retrieve all the ids of the currently opened windows (Windows can be popup-windows, windows opened in new tabs etc) - Demonstrate here
- **switchTo() -** Used to switch between different windows when multiple windows are opened by using the ids of the currently opened windows which are returned by getWindowHandles() - Demonstrate here
    - Syntax: switchTo().window("Retrieved Window ID");
- Writing Programming logic for handling more windows

Waiting mechanism in Selenium

Waiting mechanism in Selenium and Java can be categorized as below:



- Demonstrate a program which don't use waiting mechanism to understand the importance of waiting mechanism in Selenium - Demonstrate here
- Using **Thread.sleep()** in Java to overcome the waiting problems - Demonstrate here
- **Implicit Wait** - Instead of halting the program till the specified time is reached, Implicit wait will wait for all the web elements dynamically (i.e. Global wait) - Demonstrate here
- **Explicit Wait -** Instead of waiting for all the statements in the program, Explicit wait will wait only for the specific web element - Demonstrate here
    - WebDriverWait wait = new WebDriverWait(driver, 5);
    - WebElement element = wait.until(ExpectedConditions.visibilityOfElementLocated(By.linkText("Facebook")));
    - element.click();
    - Also demonstrate 'ElementToBeClickable' - Demonstrate here
- **Fluent Wait** - Use `Duration.ofSeconds(30) in the deprecated methods`
    - `Copy the FluentWait code from Selenium API Documentation`
    - `Import Function from google.common.base`
    - `Import NoSuchElementException from selenium`

Handling Alerts

- Handle Alerts using the **Alerts** Interface - Demonstrate here
    - Click on the button to display alert
    - Switch to the displayed alert using **switchTo().alert()** of WebDriver interface
    - Read the text on the alert using **getText()** method of Alert interface
    - Accept the alert using **accept()** method of Alert interface
    - After accepting the alert, switch to the main window using **switchTo().defaultContent()**
    - Close the browser window

Handling Drop-down and Multi-selection box fields

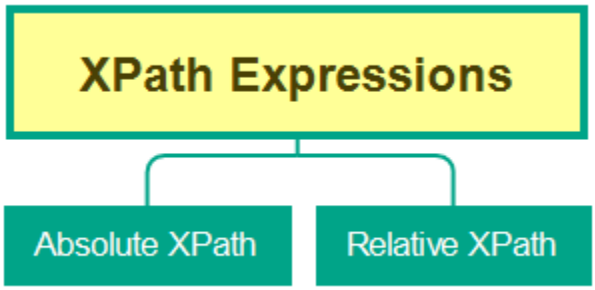
- **Select** class is the predefined class in Selenium WebDriver and it has predefined methods for performing various operations on Drop-down and Multi-selection box fields
- Using **selectByVisibleText**() with a drop-down field to select any value is a dropdown - Demonstrate here
- Using **selectByVisibleText**() with a Multi-selection box field to select multiple values - Demonstrate here
- Using **deselectByVisibleText()** with a Multi-selection box field to deselect multiple values - Demonstrate here

XPath Expressions

Out of all the locators, XPath Expressions are the powerful locators and can be able to locate any type of UI element.

Types of XPath Expressions:

XPath expressions can be classified into the below two types:



- Absolute XPath
- Relative XPath

Absolute XPath

Using XPath Expressions, we can navigate through the HTML code and locate the desired element.

Absolute XPath tries to locates the element from the root. i.e. complete path.

The below examples will help us in understanding the Absolute XPath Expressions:

Demo site :: http://compendiumdev.co.uk/selenium/basic_web_page.html

- / - locates the entire HTML document
- /html - locates the complete HTML code
- /html/head - locates the head portion of HTML code
- /html/head/title - locates the title portion of head section
- /html/body - locates the body portion of HTML code
- /html/body/p - locates all the p tags in the body portion
- /html/body/p[1] - locates the first p tag
- /html/body/p[2] - locates the second p tag
- All p tags having id 'para1' - /html/body/p[@id='para1']
- All p tags having id 'para2' - /html/body/p[@id='para2']
- All p tags having class 'main' - /html/body/p[@class='main']
- All p tags having class 'sub' - /html/body/p[@class='sub']
- All p tags having id as 'para1' and class as 'main' - /html/body/p[@id='para1'][@class='main']

Using ChroPath for generating Absolute XPath

- Inspect 'Button2' button on the www.omayo.blogspot.com using auto-generate using ChroPath

Disadvantages of using Absolute XPath

- Generate Absolute XPath for 'Button2' button on the www.omayo.blogspot.com
- Change the location of the button on the www.omayo.blogspot.com
- Generate Absolute XPath again and compare the Absolute XPaths

Relative XPath

Unlike Absolute XPath, Relative XPath tries to locate the element directly instead of locating from root.

The below examples will help us in understanding the Relative XPath Expressions:

Demo site :: http://compendiumdev.co.uk/selenium/basic_web_page.html

- Generally Relative XPath Expression starts with '//''
- //html - locates the complete HTML code
- //head - locates the head portion of HTML code directly
- //body - locates the body portion of HTML code directly
- //title - locate the title portion of HTML code directly
- //p - locates all the Paragraphs on the page
- //p[1] - locates the first paragraph
- //p[2] - locates the second paragraph
- //p[@id='para1'] - locates the paragraph having the id attribute value as 'para1'
- //p[@id='para2'] - locates the paragraph having the id attribute value as 'para2'
- //p[@class='main'] - locates the paragraph having the class attribute value as 'main'
- //p[@class='sub'] - locates the paragraph having the class attribute value as 'sub'
- //p[@id='para1'][@class='main'] - locates all the p tags having id as 'para1' and class as 'main'

Using ChroPath for generating Relative XPath

- Inspect 'Button2' button on the www.omayo.blogspot.com and auto-generate using ChroPath

Advantages of using Relative XPath

- Generate Relative XPath for 'Button2' button on the www.omayo.blogspot.com
- Change the location of the button on the www.omayo.blogspot.com
- Generate Relative XPath again and compare the Relative XPaths

More Examples on XPath:

- Demonstrate at http://compendiumdev.co.uk/selenium/basic_web_page.html
  - All p tags having ids as 'para1' | 'para2' - //p[@id='para1'] | //p[@id='para2']
  - All p tags having ids as 'para1' or 'para2' - //p[@id='para1' or @class='sub']
- Demonstrate at http://omayo.blogspot.in/
  - All the input tags inside the HTML page - //input
    - Observe that matching nodes are 34
    - Observe that all the matching nodes are highlighted
  - Finding the first input tag inside the HTML page - (//input)[1]
  - Finding the eighth input tag inside the HTML page - (//input)[8]
  - Finding the last input tag inside the HTML page - (//input)[34]
  - Finding the input tags having name attribute - //input[@name]
  - Finding the input tags using its attribute name and value - //input[@value='orange']
  - Finding the input tags using multiple attribute names and values - //input[@name='color'][@value='blue']
  - Finding the input tags having checked attribute - //input[@checked]
  - All the image tags inside the HTML page - //img
  - Finding an image element using its attribute values - //img[@height="200px"]
  - Finding an drop down field having class 'combobox' and also an hyper link having value 'link2' - //select[@class='combobox'] | //a[@value='link2']
  - Finding hyper link having id='link1' and also the hyper link having value='link2' - //a[@id='link1' or @value='link2']
  - Finding 'button' tags having id 'but2' - //button[@id='but2']
  - Finding any tags having id 'but2' - //*[@id='but2']
  - Finding 'button' tags having any attribute value as 'but2' - //button[@*='but2']
  - Finding 'button' tags having id attribute with any value - //button[@id

XPath Expressions - Part 2

  - Finding 'input' tags having name attribute as 'gender' - //input[@name='gender']
  - Finding the first 'input' tags having name attribute as 'gender' - //input[@name='gender'][1]
  - Finding the second 'input' tags having name attribute as 'gender' - //input[@name='gender'][2]
  - Finding any tags having name attribute as 'gender' - //*[@name='gender']
  - Finding 'input' tags having any attribute value as 'gender' - //input[@*='gender']
  - Finding any tags having any attribute value as 'gender' - //*[@*='gender']
  - Finding 'input' tags having name attribute value as anything - //input[@name]
  - Finding any tags having any attribute value as anything - //*[@*]havi
  - Finding any elements having id attribute value as 'radio1' and name attribute value as 'gender' - //*[@id='radio1'][@name='gender']
  - Finding the first 'input' tags having name attribute as 'gender' and is the first element - //input[@name='gender'][1]
  - Finding the second 'input' tags having name attribute as 'gender' and is the second element - //input[@name='gender'][2]

- Finding any elements having id attribute value as 'radio1' or name attribute value as 'gender' - //*[@id='radio1' or @name='gender']
- XPath Expressions - Part 2 - Demonstrate at http://omayo.blogspot.in/
  - Find all the hyper links in the page - //a
  - Find all the hyper links having URL 'http://www.Selenium143.blogspot.com' - //a[@href='http://www.Selenium143.blogspot.com']
  - Find the first hyper link having URL 'http://www.Selenium143.blogspot.com' - (//a[@href='http://www.Selenium143.blogspot.com'])[1]
  - Find the third hyper link having URL 'http://www.Selenium143.blogspot.com' - (//a[@href='http://www.Selenium143.blogspot.com'])[3]
  - Difference between (//a[@href='http://www.Selenium143.blogspot.com'])[3] and //a[@href='http://www.Selenium143.blogspot.com'][3]
    - Second XPath searches for the third element at tag level
    - First XPath searches for the third element at page level
  - Find first child of 'html' tag - //html/*[1]
  - Find second child of 'html' tag - //html/*[2]
  - Find first child of 'body' tag - //body/*[1]
  - Find second child of 'body' tag - //body/*[2]

XPath Functions

- XPath functions: Part1
  - **text()** - Demonstrate at http://omayo.blogspot.in/
    - Find the p tags having the exact text 'PracticeAutomationHere' - //p[text()='PracticeAutomationHere']
    - Use . instead of text() - Find the p tag having the exact text 'PracticeAutomationHere' - //p[.='PracticeAutomationHere']
  - **contains()** - Demonstrate at http://omayo.blogspot.in/
    - Purpose:
      - It is used when the value of any attribute changes dynamically.
      - Has the ability to find the elements with partial text
      - If part of the attribute value is changing dynamically i.e. id='123main123' to id='456main456', we can use //tagName[contains(@id,'main')] to locate such dynamically changing attribute values.
    - Find the input tag having the text 'ra' inside its value attribute text - //input[contains(@value,'ra')]
    - Find the p tag containing the text 'Automation' - //p[contains(text(),'Automation')]
    - Find the p tag containing the text 'Automation' using.- //p[contains(.,'Automation')]
  - **starts-with()**
    - Purpose:

            - It is used when the value of any attribute changes dynamically.
            - Has the ability to find the elements with partial text i.e. initial partial text
            - If part of the attribute value is changing dynamically i.e. id='main123' to id='main456', we can use //tagName[starts-with(@id,'main')] to locate such dynamically changing attribute values.
        - Find the input tag having the value attribute text starting with letter 'o' - //input[starts-with(@value,'o')]
        - Find the p tag starting with text 'Practice' - //*[starts-with(text(),'Practice')]
        - Find the p tag starting with text 'Practice' using . - //*[starts-with(.,'Practice')]
- XPath functions: Part2 (Demonstrate at http://compendiumdev.co.uk/selenium/basic_web_page.html)
    - Find the first child of 'body' tag - //body/*[1]
    - **last()** - Find the last child of 'body' tag - //body/*[last()]
    - Find the first 'p' tag - //p[1]
    - last() - Find the last 'p' tag - //p[last()]
    - Find the last but one 'p' tag - //p[last()-1]
    - Locate the last but 2 input tag - (//input)[last()-2] (Demonstrate at http://omayo.blogspot.in/)
    - Find second 'p' tag having class 'sub' - //p[2][@class='sub']
    - Find the last 'p' tag having class 'sub' - //p[last()][@class='sub']
    - Find the last but one 'p' tag having class 'main' - //p[last()-1][@class='main']
- XPath functions: Part3 (Demonstrate at http://compendiumdev.co.uk/selenium/basic_web_page.html)
    - **position()** - Find the first 'p' tag - //p[position()=1]
    - position() - Find the second 'p' tag - //p[position()=2]
    - position() - Find the 8th input tag - (//input)[position()=8] (Demonstrate at http://omayo.blogspot.in/)

XPath Axes

- XPath AXES: (Demonstrate at http://omayo.blogspot.in/)
    - Purpose:
        - If you want to locate an element which doesn't have id/name/class etc., with the help of XPath Axes we can locate such elements not having id/name/class with the help of id/name/class attributes of ancestor/descendant tags.
    - following
        - Purpose: Selects everything in the document after the closing tag of the current node

        - Find all the 'body' tags after the 'head' tag - //head/following::body
        - Find all the 'div' tags after //body/div[1]/div - //body/div[1]/div/following::div
        - Find the first 'div' after //body/div[1]/div - //body/div[1]/div/following::div[1]
        - Find all the 'input' tags after //body/div[1] - //body/div[1]/following::input
- preceding
    - Purpose: Selects all nodes that appear before the current node in the document, except ancestors nodes
    - Find all the 'head' tags before the 'body' tag - //body/preceding::head
    - Find all the 'div' tags before //body/div[4] - //body/div[4]/preceding::div
- following-sibling
    - Purpose: Selects all siblings after the current node
    - Find all the 'div' tag siblings after //body/div[1] - //body/div[1]/following-sibling::div
    - Find all the 'p' tag siblings after //body/p[1] - //body/p[1]/following-sibling::p (Demonstrate at http://compendiumdev.co.uk/selenium/basic_web_page.html)
- preceding-sibling
    - Purpose: Selects all siblings before the current node
    - Find all the 'div' tag siblings before //body/div[4] - //body/div[4]/preceding-sibling::div
    - Find all the 'p' tag siblings before //body/p[2] - //body/p[2]/preceding-sibling::p (Demonstrate at http://compendiumdev.co.uk/selenium/basic_web_page.html)
- parent
    - Purpose: Selects the parent of the current node
    - Find the parent of 'head' tag - //head/parent::html
    - Find the parent of 'body' tag - //body/parent::html
    - Find the parent of 'title' tag - //title/parent::head
    - Find the parent of first 'div' tag inside 'body' tag i.e. //div[1] - //div[1]/parent::body
- child
    - Purpose: Selects all children of the current node
    - Find one of the child tag say 'head' of 'html' tag - //html/child::head
    - Find one of the child tag say 'body' of 'html' tag - //html/child::body
    - Find one of the child tag say 'title' of 'head' tag - //head/child::title
    - Find one of the child tag say first 'div' tag of 'body' tag - //body/child::div[1]
- ancestor
    - Purpose: Selects all ancestors (parent, grandparent, etc.) of the current node
    - Find the ancestor 'html' tag for 'title' tag - //title/ancestor::html

        - Find the ancestor 'html' tag for 'head' tag - //head/ancestor::html
        - Find the ancestor 'html' tag for 'body' tag - //body/ancestor::html
    - descendant
        - Purpose: Selects all descendants (children, grandchildren, etc.) of the current node
        - Find the descendant 'title' tag for 'html' tag - //html/descendant::title
        - Find the descendant 'head' tag for 'html' tag - //html/descendant::head
        - Find the descendant 'body' tag for 'html' tag - //html/descendant::body

Miscellenious

- //ParentXpath//ChildXPath//GrandChildXPath
    - Child XPath will be searched in the parent XPath located section or area

CSS Selectors - Cheat-sheet

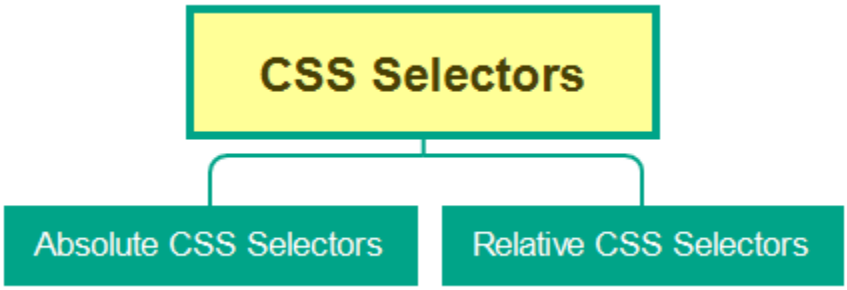
While locating the UI elements, CSS Selectors needs to be used as a priority over XPath Expressions.

The below are the reasons, why the CSS Selectors needs to be preferred over XPath Expressions:

- When compared to XPath Expressions, CSS Selectors locate the UI elements faster.
- Selenium may not be able to locate few UI elements using XPath Expressions, while executing the Automation scripts on Internet Explorer Browser.

Types of CSS Selectors:

CSS Selectors can be classified into the below two types:



Absolute CSS Selectors

Absolute CSS Selectors tries to locates the element from the root. i.e. complete path.

The below examples will help us in understanding the Absolute CSS Selectors:

Demo site :: http://compendiumdev.co.uk/selenium/basic_web_page.html

- html - locates the complete HTML code
- html > head - locates the head portion of HTML code
- html > head > title - locates the title portion of head section
- html > body - locates the body portion of HTML code
- html > body > p - locates all the p tags in the body portion
- html > body > p[id='para1'] - Locates p tag having id as 'para1'
- html > body > p[class='sub'] - Locate p tag having class as 'sub'
- html > body > p#para1 - Locates p tag having id as 'para1'
- html > body > p.sub - Locates p tag having class as 'sub'
- html > body > p[id='para1'][class='main'] - Locates p tag having id as 'para1' and class as 'main'

Note: ChroPath cannot auto-generate absolute css selectors

Relative CSS Selectors

Relative CSS Selectors locates the elements directly, instead of locating from root.

The below examples will help us in understanding the Relative CSS Selectors:

Demo site :: http://compendiumdev.co.uk/selenium/basic_web_page.html

- html - locates the html tag
- head - locates the head portion of HTML code
- title - locates the title portion of head section
- body - locates the body portion of HTML code
- p - locates all the p tags in the body portion
- p[id='para1'] - Locates p tag having id as 'para1'
- p[class='sub'] - Locates p tag having class as 'sub'
- p[id='para1'][class='main'] - Locates p tag having id as 'para1' and class as 'main'
- p#para1 - Locates p tag having id as 'para1'
- p.sub - Locates p tag having class as 'sub'

Using ChroPath for generating Relative CSS Selectors

- Using ChroPath

Relative CSS Selectors (More examples)

- Locating different elements using Relative CSS Selectors (Demonstrate at http://compendiumdev.co.uk/selenium/basic_web_page.html)
    - HTML page - html
    - HTML Head - head
    - HTML Title - title
    - HTML Body - body
    - p tags - p
    - p tags inside body - body p
    - p tags inside html - html p
    - Locate p tag having id 'para2' - p[id='para2']
    - Locate p tag having class 'main' - p[class='main']
    - Locate elements having id 'para1' - [id='para1']
    - Locate elements having class 'sub' - [class='sub']
    - Using # for locating elements by ids
        - p tag having id 'para1' - p#para1
        - p tag having id 'para2' - p#para2
        - Locate elements having id 'para2' - #para2
    - Using . for locating elements by class
        - p tag having class 'main' - p.main

        - p tag having class 'sub' - p.sub
        - Locate elements having class 'main' - .main
    - (Demonstrate at http://omayo.blogspot.in/)
        - Locate input tag having value='blue' - input[value='blue']
        - Locate elements having value='blue' - [value='blue']
        - Locate all the input tags - input
        - Locate all the elements having 'value' as attribute - [value]
        - Locate all the elements having 'id' as attribute - [id]
        - Locate all the elements having 'name' as attribute - [name]
        - Locate all the elements having 'href' as attribute - [href]
        - Locate all the elements having 'src' as attribute - [src]
        - Locate all the img tags having 'src' as attribute - img[src]
    - (Demonstrate at http://compendiumdev.co.uk/selenium/basic_web_page.html)
        - Locate all the p tags having 'id' as attribute - p[id]
        - Locate all the elements having 'id' as attribute - [id]
        - Locate all the p tags having 'class' as attribute - p[class]
        - Locate all the elements having 'class' as attribute - [class]
    - (Demonstrate at http://compendiumdev.co.uk/selenium/basic_web_page.html)
        - *:first-child
            - Locate the first child inside body tag - body > *:first-child
            - Locate the first child inside head tag - head > *:first-child
            - Locate the first child inside html tag - html > *:first-child
            - Locate the first p tag - p:first-child
            - Locate the first p tag having id 'para1' - p[id='para1']:first-child
        - *:last-child
            - Locate the last child inside body tag - body > *:last-child
            - Locate the last child inside head tag - head > *:last-child
            - Locate the last child inside html tag - html > *:last-child
            - Locate the last p tag - p:last-child
            - Locate the last p tag having id 'para2' - p[id='para2']:last-child
        - *:nth-child
            - Locate the second child inside the html tag - html > *:nth-child(2)
            - Locate the first child inside the html tag - html > *:nth-child(1)
            - Locate the first child inside the body tag - body > *:nth-child(1)
            - Locate the second child inside the body tag - body > *:nth-child(2)
            - Locate the second p child inside the body tag - body > p:nth-child(2)
            - Locate the second child inside the body tag - p:nth-child(2)
            - Locate the second child having id 'para2' - p[id='para2']:nth-child(2)
            - Locate the second child having p tag and who's ancestor is html tag - html p:nth-child(2)
    - (Demonstrate at http://omayo.blogspot.in/)
        - textarea[id='ta1'] , button[id='but2'] - Works as | operator in xpath
        - * - All the elements will get highlighted
        - head > * - All the elements under head tag will get highlighted
        - body > * - All the elements under body tag will get highlighted
    - (Demonstrate at http://compendiumdev.co.uk/selenium/basic_web_page.html)

        - p[class^='ma'] - starts with
        - p[class$='ub'] - ends with
        - p[class*='ai'] - contains
    - (Demonstrate at http://compendiumdev.co.uk/selenium/basic_web_page.html)
        - p[id='para1'][class='main'] - works as and operator
        - p:not([id='para1'])
        - p:not([id='para1'])[class='sub']
        - p:not([id='para1']):not([class='main'])
    - (Demonstrate at http://compendiumdev.co.uk/selenium/basic_web_page.html)
        - Following Sibling having tag p - p[id='para1']+p
        - Following sibling having any tag - head+*
        - Demonstrate at http://book.theautomatedtester.co.uk/chapter2
            - Following sibling having link tag - title + link
    - (Demonstrate at http://omayo.blogspot.in/)
        - Locate disabled elements - *:disabled
        - Locate enabled elements - *:enabled
        - Locate selected checkbox or radio options or drop down field options etc - *:checked (Need not be default selected)

Handling Frames

- Demonstrate the problem statement
    - 'NoSuchElementException' will be displayed on trying to find the web element which is displayed in an iframe
    - Enter text into a text field inside the iframe page
- Frame is a web page which is embedded in another web page, and is used to display multiple pages inside a single web page.
    - Developers can also embedded a document to be scrolled inside a frame
- In HTML, <iframe> is the tag used by the Web Developers to display any Frame on the Page.
- View the iframes in www.omayo.blogspot.com page - Right click on the frames and observe that 'This Frame' option will be displayed
- Switch to the required frame and perform operations (View code here)
    - First switch to a frame and enter text into text field inside frame - Using **switchTo().frame(WebElementOfFrame)**
    - Switch back to the main page using **switchTo().defaultContent()** and type text into the 'Search' text box field
- Finding the number of frames available on the page

    - System.out.println(driver.findElements(By.tagName("iframe")).size());
- We can switch to the frames using id locator or name locator also
    - driver.switchTo().frame("idvalue");
    - driver.switchTo().frame("namevalue");

Handling Light-box

- Unlike alerts, frames or windows, we need not switch to Lightbox for performing operations.
    - http://omayo.blogspot.com/p/lightbox.html
- Light boxes are part of the same HTML web page only.
- Demonstrate a program which handles the light box - Demonstrate here
    - Hence it is not required to switch to the lightbox for performing operations on it.
- Real time examples for Light-box
    - https://www.flipkart.com/

Actions Class

- **Actions** is a predefined Class of Selenium WebDriver
- Actions class contain various predefined methods which can simulate Mouse and Keyboard Events
- The below are the different methods of Actions class which we can use in automation for handing Mouse and keyboard actions:
    - **moveToElement(), click(), perform() and build() methods**
        - Demonstrate moving the mouse to Blogs menu, followed by Selenium143 menu option and clicking it using mouse - Demonstrate here
        - Optimizing the above program using build().perform() - Demonstrate here
        - Dont huddle the mouse while handing the mouse actions using Actions class
    - **dragAndDropBy()**
        - Demonstrate dragging and dropping the startButton horizontal to the right - Demonstrate here
            - `Application URL: http://omayo.blogspot.com/p/page3.html`
        - Demonstrate dragging and dropping the startButton horizontal to the left - Demonstrate here
    - **contextClick()**
        - Demonstrate right clicking on Search Box field - Demonstrate here
    - **doubleClick()**

        - Demonstrate double clicking on double click text in the omayo blog - Demonstrate here
    - **dragAndDrop()**
        - Demonstrate dragging and dropping an element from a location to a different location - Demonstrate here
        - Application URL: http://dhtmlgoodies.com/scripts/drag-drop-custom/demo-drag-drop-3.html
    - **keyDown() and keyUp() methods**
        - Demonstrating opening a link in new tab (Compendium Link on Omayo) - Demonstrate here
    - **sendKeys()**
        - Demonstrate typing username, then press tab key, entering password and then pressing tab key and pressing enter key - Demonstrate here
        - Login functionality available at the end of the omayo blog page

Keys Class

- Login using Enter key on http://tutorialsninja.com/demo/index.php?route=account/login
    - Demonstrate here
- Use Keys.chord for pressing multiple keys together
    - Enter text into the text area field and clearing it using Ctrl + z keys - Demonstrate here

Taking Screenshots in Selenium

- We can take screenshots in Selenium using the predefined Interface and methods in Selenium.
- Demonstrate taking Screenshots in Selenium - Demonstrate here
    - Copy the below two lines of code, where ever we need to take screenshots and modify them accordingly:
        - File screenshotFile = ((TakesScreenshot) driver).getScreenshotAs(OutputType.FILE);
        - FileUtils.copyFile(screenshotFile, new File("screenshots\\screenshot1.jpg"));
    - Note: We have to download the Jar files for importing FileUtils
        - Google search for FileUtils common.io package

Handling Auto-suggestive Dropdowns

- We can type text into Auto-suggestive drop-down fields in-order to filter the drop-down results

- Demonstrating selecting an option from the Auto-Suggestive Dropdowns - Demonstrate here

Handling Calendar

- Demonstrating a program for handling a calendar - Demonstrate here

Handling Table

- Create a table using the HTML code
- XPath or CSS
    - Inspect the table using ChroPath and locate table heading, table body, table headings and table data etc.
    - Locating the table data in first row
    - Locating the table data in second row
    - Locating the second column data in third row
    - Locating all column data of third column
- Program to print the table headings and all the above other details
- Program to find the place of the given Person name, when we know the column number of names and place - Demonstrate here
- Program to find the number of rows in a table
- Program to find the number of columns in a table

Maven

- Download and configure Maven
    - download maven
    - MAVEN_HOME
    - Path - maven bin path
    - mvn --version to confirm
- The purpose of Maven is to automatically download the Jar files with the dependency tags provided in pom.xml file
- Creating a Maven Project
- maven-archetype-quickstart in Eclipse IDE
- Project name into Artifact id
- Create a Class with main method
- Provide the dependency tags for **Selenium Java** from mvnrepository.com in pom.xml file

WebDriverManager

- Provide the dependency tags for **WebDriverManager** from mvnrepository.com in pom.xml file
- Purpose of WebDriverManager

Automating an End to End Scenario

- Program for creating an automation script for an end to end scenario - Demonstrate here

Debugging Process

- Demonstrate how to debug our Automation scripts to find where exactly it is going wrong

TestNG

TestNG plays a major role in Test Automation Frameworks Development.

- **Installing TestNG in Eclipse IDE**
  - Checking that TestNG is not available in Eclipse IDE
  - Go to Testng.org > Eclipse > Plug-in Installation > Use the latest URL referred at this location
  - Check whether TestNG is installed in Eclipse IDE
- **Configure the Project with TestNG JAR file**
  - Create a new Maven Project and execute a sample Selenium Script
  - Configure the Project with TestNG (By adding Dependency tags)
- **TestNG Annotations**
  - TestNG is an API similar to Java and Selenium WebDriver
  - TestNG has a huge list of Annotations and below are few annotations we need to learn for Selenium:
    - Check the annotations list in TestNG API
      - @Test
      - @BeforeMethod
      - @AfterMethod
      - @BeforeTest
      - @AfterTest
      - @BeforeSuite
      - @AfterSuite
      - @BeforeClass
      - @AfterClass

        - And many more
- **@Test**
    - The purpose of this annotation is to represent the methods inside Java class as Tests.
    - This annotation replaces main() method in traditional Java programs - Demonstrate
    - Run as 'TestNG Test' and view the execution results in both Eclipse IDE Console and TestNG Results tab
    - TestNG Reports
        - Under test-output folder > index.html
        - Refresh the Project and view the TestNG report generated at 'test-output' > old > index.html
        - Passing a Test
        - **Failing @Test annotated methods**
            - If any @Test annotated method fails, it will be displayed as failed test in Eclipse IDE Console, TestNG results tab and default TestNG reports
            - Demonstrate a program in which the @Test annotated method is failing - Demonstrate here
        - **Skipping a test in TestNG**
            - **throw**
                - If we want to manually throw any exception based on some condition, we have to use **throw** - Demonstrate here
                    - Syntax: **throw new Exception();**
                    - Exception is the predefined class of Java
            - If we want to manually skip any test method in TestNG, we have to use throw - Demonstrate here
                - Syntax: **throw new SkipException();**
                - SkipException is the predefined class of TestNG
        - **TestNG Assertions**
            - We perform testing to verify whether a particular test is passed or failed.
            - In the similar way, TestNG provides a predefined Class '**Assert**' and its predefined methods assertEquals(), assertNotEquals(), assertTrue(), assertFalse() and fail() to verify whether @Test annotated methods are passed or failed.
            - **assertEquals()**

        - Demonstrate a program which uses assertEquals() to verify a failing test - Demonstrate here
        - Demonstrate a program which uses assertEquals() to verify a passing test - Demonstrate here
    - **assertTrue()**
        - Demonstrate a program which uses assertTrue() to verify a failing test - Demonstrate here
        - Demonstrate a program which uses assertTrue() to verify a passing test - Demonstrate here
    - **assertFalse()**
        - Demonstrate a program which uses assertFalse() to verify a failing test - Demonstrate here
        - Demonstrate a program which uses assertFalse() to verify a passing test - Demonstrate here
    - **fail()**
        - Demonstrate a program which fails a test directly - Demonstrate here
- Demonstrate a program which has multiple **@Test** testNG annotated tests - Demonstrate here
    - A single class can have multiple tests,
        - Create multiple Tests and execute
        - Tests will be executed in alphabetical order
- Demonstrate a program which executes the @Test annotated methods according to their priority - Demonstrate here
    - **priority** attribute of @Test annotation is used to prioritize the tests
        - @Test(priority=1)
    - Check the priority attribute in TestNG AP

- **@BeforeMethod**
  - Methods annotated with @BeforeMethod annotation will be executed before the @Test annotated methods.
  - Used with the methods having the code, which is required to be executed before executing the code in test methods.
    - Example: **Opening Browser and Application** before actually performing any tests on them.
  - Demonstrate a program which uses @BeforeMethod and a single @Test method - Demonstrate here
  - Demonstrate a program which uses @BeforeMethod and multiple @Test methods - Demonstrate here
- @**AfterMethod**
  - Methods annotated with @AfterMethod annotation will be executed after the @Test annotated methods.
  - Used with the methods having the code, which is required to be executed after executing the code in test methods.
    - Example: Closing the Application/Browser after the tests are performed on them.
  - Demonstrate a program which uses @AfterMethod and a single @Test method - Demonstrate here
  - Demonstrate a program which uses @AfterMethod and multiple @Test methods - Demonstrate here
- **@BeforeClass**
  - Methods annotated with @BeforeClass annotation will be executed before executing the Class
  - Used with the methods having the code, which is required to be executed before executing the Class code
    - Example: Instead of opening application for each and every test, if we want to open the application only once before all the tests in a Class are executed.
  - Demonstrate a program which uses @BeforeClass along with other annotated methods - Demonstrate here
- **@AfterClass**
  - Methods annotated with @AfterClass annotation will be executed after executing the Class

        - Used with the methods having the code, which is required to be executed after the executing the Class code
            - Example: Instead of closing application for each and every test, if we want to close the application only once after all the tests inside the Class got executed.
        - Demonstrate a program which uses @AfterTest along with other annotated methods - Demonstrate here
- **Executing the Java class files in batch using TestNG.xml**
    - Instead of running the Classes individually, we can use testng.xml file
    - First create multiple classes (Say ClassOne, ClassTwo,Class Three, ClassFour etc) having @Test methods and execute the classes individually
    - Create a testng.xml file in the Project and execute all the classes at a go using testng.xml file
        - Right click on the Project > Select TestNG > Convert To TestNG
    - Suite name in testng file - suite tag - say payments
    - Test module name in testng file - test tag - say netbanking
    - Create multiple test modules in testng.xml file
    - We can specify a group of classes under classes and class tags
- @BeforeTest
- @AfterTest
- @BeforeSuite
- @AfterSuite
- Commenting in testng.xml file
- Excluding a method from execution by changing the testng.xml file
    - <class name="demopack.DemoMavenProject.Demo"> <methods> <exclude name="demoMethodTwo"/> </methods> </class>
- Or, we can include a specific method that only needs to executed by changing the testng.xml file
    - <class name="demopack.DemoMavenProject.Demo"> <methods> <include name="demoMethodTwo"/> </methods> </class>
- Exclude or include with regular expressions
    - <class name="demopack.DemoMavenProject.Demo"> <methods> <include name="demo.*"/> </methods> </class>
- Executing the tests at package level
    - Under test tags, specify the package name - <test> <packages> <package name="xyz"/></packages> </test>
- Groups
    - include a group - view here
    - exclude a group (Simply change include to exclude in the above example here)
    - @Test(groups={"smoke"})
- More attributes
    - dependsOnMethods - before executing the test, the dependent tests will be executed - View here
    - enabled=false to stop it from executing- View here

- timeOut=5000 - View here

TestNG (Part 3)

- Testng.xml Parameterization
  - We should not hard-code the data
  - Suite level parameterization - View here
  - Test level parameterization - View here
  - Multiple Parameters can be passed too
  - Run the testng.xml file only
  - Running from individual classes will result in errors
- Parameterization for Passing multiple Data to the specified test - View here
  - @DataProvider
  - (dataProvider="methodName")
- TestNG Listeners
  - Listeners listen to our code and perform necessary actions like taking screenshots, failing tests etc.
  - Create any class with name Listeners or any and implement the ITestListener interface of TestNG
  - Add the required methods of ITestListener interface - View here
  - Add the Listeners tags below the suite tag in testng.xml file - View here
- TestNG - Parallel Execution of Tests at test level and suite level
  - parallel="methods" thread-count="2" (Add this in the test tag)
  - parallel="methods" thread-count="2" (Add this in the suite tag)

Cross Browser Testing using Selenium Grid

We can distribute our tests to run on different machines having different browser using Selenium Grid.

- TestNG - Allows parallel execution, where as Selenium Grid - Allows distribution of tests on different machines and their browsers
  - Limitations of TestNG
  - TestNG + Selenium Grid saves time
- Understanding Hub and Nodes
- Steps for configuring Selenium Grid

- Download Selenium Standalone Server
- Register Hub by following below steps:
  - Open command prompt from C drive
  - Copy Selenium Standalone Server to C drive
  - Run java -jar selenium-server-standalone-3.141.59.jar -role hub
  - Get the ip address and port number from the command prompt and append /grid/console as shown below:
    - http://192.168.0.106:4444/grid/console
- Register nodes to the above Hub
  - In Node machine, make sure Java is already installed and configured (check java -version)
  - Download Selenium Standalone Server Jar here too
  - Download chromedriver.exe
  - Copy Selenium Standalone Server and chromedriver.exe to C drive
  - Run java -Dwebdriver.chrome.driver="C:\chromedriver.exe" -jar selenium-server-standalone-3.141.59.jar -role webdriver -hub http://192.168.0.106:4444/grid/register -port 5555
    - We can run on any port
  - Hub command prompt - will show that a node got registered to the hub
  - Refresh in Hub - http://192.168.0.106:4444/grid/console
- Create a Project
  - Configure it with selenium standalone server jar file
  - Create a class with main method
  - Create an object for DesiredCapabilities and other code - View here
  - Execute the Test to see whether the Test is running on the node machine

Properties (Global Parameters)

1. Create a Maven Project
2. Execute a selenium script with hard-coded values to login into the application
3. Creating the data.properties file
4. Create an object for the Properties
5. load() for loading the properties file
6. getProperty() for retrieving the data from the properties file
7. store() for updating the data in the properties file
8. View code here

SelectorsHub

1. Creating XPath Expressions and CSS Selectors manually
2. SelectorsHub will help you

Maven (Continued)

- Create a new Maven Project
- Create any class ending with the name 'Test' say 'OneTest' and write sample Selenium code - view here
- Make sure Maven Surefire plugin is inside the pom.xml file
    - Add the latest version of Maven Surefire Plugin
- Go to project director and perform the below commands
    - Maven Build Life Cycle Phases
        - Clean
        - Compile
        - Test
        - And many
    - mvn clean - Cleans the project and removes build errors if any
    - mvn compile - compile build phase after maven will identify an compiler error in the code without eclipse ide
    - mvn test - performs clean, compile and test
        - Maven Surefire Reports
        - Also, verifies windows > users > username > .m2 > repository > org for required jars mentioned in pom.xml file
        - If not available, it will download the jars from the mvnrepository
- Executing using testng.xml file
    - Add the "Maven Surefire Plug-in for TestNG configuration" into the surefire plug-in
    - mvn test (Running complete things mentioned in testng.xml)
    - mvn -Dtest=ClassTwoTest test (Only specified class tests will be executed)
    - Creating Profiles in pom.xml file - View here
        - mvn clean test -P Smoke

Jenkins

- Automating the initiation of the automation scripts
- Using Jenkins we can schedule and trigger the scripts at early hours if needed
- In Real time, Dev-ops team will setup Jenkins for you in a centralized Server machine.
    - download Jenkins > Generic Java Package (war)
    - Open command prompt from the place where war file is available and hit the below command
        - java -jar jenkins.war
        - You will see a password
        - Then open localhost:8080 in chrome browser
        - After a while, it will ask for password, provide the password
- We can run maven commands in Jenkins
    - Manage Jenkins > Global Tool Configurations
    - Add Java
        - JAVA_HOME

        - JDK Path in your machine
        - Don't opt for automatic installation
    - Add Maven
        - Opt for Install Automatically
        - Select the latest version of Maven while opting
    - And Save & Apply
- Create a new Job in Jenkins
    - Jenkins Home > New Item > Give Job name and select 'Freestlye' project
    - Copy the Project to the Jenkins folder (C > Users > Username > Jenkins)
    - Select Advanced in Jenkins > Select Use custom workspace checkbox
    - Directory - Give your Project Path
    - Build > Add build step > Invoke top-level Maven targets > Select Maven Version >type test command without mvn in the field and save
- Build Now to Run
- Click on the Results > Output Console
- Installing TestNG Results Plugin
    - Manage Jenkins > Plugin Manager > Available > TestNG > Install without restart
    - Job > Configure > Post Build Actions > Publish TestNG Results

Extent Reports

1. Selenium by default don't have any reporting features - View Code
2. Once we configure TestNG, we get TestNG reports under test-output > index.html - View code
3. When we run the Tests using Maven command - mvn test, we will get target > surefire-reports > index.html
4. For a good looking reports, we have to go for popular Extent Reports
    1. Create a Maven Project
    2. Configure the Project with Selenium, WebDriverManager, TestNG and ExtentReports
5. Create a Test method and write some sample automation code - View code
6. Create a BeforeMethod configuration method and do the below:
    1. Create an object for ExtentSparkReporter and provide the path in its constructor
        1. String path = System.getProperty("user.dir")+"\\reports\\index.html";
        2. ExtentSparkReporter reporter = new ExtentSparkReporter(path);
        3. reporter.config().setReportName("Omayo Test Results");
        4. reporter.config().setDocumentTitle("A Test Results");
        5. extent = new ExtentReports();
        6. extent.attachReporter(reporter);
        7. extent.setSystemInfo("Operating System","Windows 10");
        8. extent.setSystemInfo("Tested By","Arun Motoori");
    2. Make ExtentReports class global and write the below link before the test method:
        1. ExtentTest eTest = extent.createTest("Test One Started");
        2. WebDriverManager.chromedriver().setup();
        3. WebDriver driver = new ChromeDriver();

4. eTest.info("Chrome Browser Launnched");
5. driver.get("http://omayo.blogspot.com/");
6. eTest.info("Navigated to Application");
7. eTest.fail("Test got failed"); - Demonstrate separately
8. extent.flush();

POI API

1. We can use this API for retrieving the data from the Excel files to Java program
2. Create a Maven Project
3. Add dependency tags for poi and poi-ooxml
4. Create an excel file with dummy test data - View here
5. Finding the number of sheets - View here
6. Retrieving the data of the required Test say 'Register' and print it - view here
7. Do the below changes:
    1. Adding the above retrieved data of the required Test into an ArrayList, instead of printing
    2. Move the code in to a reusable method and remove hardcoding of testcase
    3. Return the ArrayList
    4. Access this method from a Test method in another class
    5. Also make this reusable method to access numeric values
    6. View here

Log4j2

1. Log4j is for logging
2. Logs are like running commentary
3. The purpose of logs is to know how the script got executed at later point of time and if it failed where exactly it gone wrong
    - We can know what got executed in the code and how it got executed.
    - We can know where exactly in the code, Exceptions / errors occurred.
4. Implement logging in Selenium Automation code using **System.out.println()** statements
    - Create a Java Project
    - Configure the Project with Selenium WebDriver
    - Create Selenium Automation code to visit Omayo blog, navigate to Compendium site, navigate back to Omayo blog, forward again to Compendium site and close the browser.
    - Write **System.out.println()** statements for logging - View here
5. Disadvantages of SOP logging

- SOP's are for simple logging, and cannot be used for advanced logging.
    - We cannot turn off the logs when required
    - Logs are captured in the console, instead of a separate file
    - No Time-stamp will be displayed in the required format (Developers may need this from testers)
    - We cannot provide earlier logs
    - Cannot differentiate between logs (Level of logs)
    - And many more

6. To resolve the above disadvantages, we have to use advanced logging - **Log4j logging**
7. Similar to Java, Selenium, TestNG and POI API's , Log4j is released into market as API by Apache guys
8. Implementing Log4j in Selenium Automation
    - Step1: Go to the downloads page of Log4j and download the zip file
    - Step2: Extract the Zip file and configure the Project with log4j-core ad log4j-api jars only
    - Step3: Write the below code
        - Logger logger = LogManager.getLogger(Demo.class.getName());
        - logger.debug for all the general logs
        - logger.info for successfull test
        - logger.error for failure test
        - Execute and observe that no logs will be displayed in the output console, **except error logs**
        - We need a configuration file for all the logs to work
    - Step4: Create log4j configuration file by following the below steps:
        - Search for 'Log4j configuration' and go to the required URL page and find any xml configuration files
            - There are two types of tags
                - Appenders (Information on where to log)
                - Loggers (Information on what to log)
        - Create resources folder under the project
            - Create an xml file with the name log4j2.xml
            - Paste the xml configuration things into the file
            - Change the root level to All (Log Levels - All < Trace < Debug < Info < Warn < Error < Fatal < Off)
            - Build the project and add the resources folder
    - Step5: Printing the logs to a file instead of console
        - Add RollingFile tags between Appenders tags - View here
            - Change the AppenderRef to File from Console
                - <AppenderRef ref="File" />
        - Create logs folder under Project and create prints.log file
            - Add the properties tags before the Appenders tag - View here
        - Run the Demo and observe that the logs got printed into the File instead of output console

Test Data from Database

1. Install the MySQL Server in your machine as explained at this Video - https://www.youtube.com/watch?v=NQXxFPyqmDg
2. Using MySQL Workbench, connect to MySQL and perform the below:
    1. Connect to a DB
        1. create table Employees(id int, name varchar(20),location varchar(20),experience int);
        2. describe Employees;
    2. Insert records into Table
        1. insert into Employees values(1,'Arun','Hyderabad',12);
        2. insert into Employees values(2,'Varun','Bangalore',9);
        3. insert into Employees values(3,'Tharun','Delhi',7);
    3. Select a record from the Table
        1. Select * from Employees where id=3;
3. Create a Maven Project
    1. Configure this Project with 'MySQL Java Connector' Jar file
    2. Write the Java program to print a single record data - View here
4. Use the same concept to read the Username and Password from the Table at Database and pass it into the Selenium Code
    1. Use http://tutorialsninja.com/demo/ for this Demonstration to Login

Page Object Model and Page Factory Design Patterns

1. Using Page Object Model and Page Factory design pattern, we can write our scripts in an effective way
    - For each page, we have to create Class and store objects of the page
    - When something changes, we have to update in the respective class, instead of in all test scripts
2. Advantages
    - Achieving lesser maintenance
    - Readability of Scripts
3. Create a Maven Project
    - Create a packages - objectrepository
    - Under objectrepository package, create class for Login page and write the below code:
        - WebDriver driver;
            - Configure Selenium WebDriver for the above statement
        - Create Objects for the required UI elements
            - By email = By.id("input-email");
            - By password = By.id("input-password");
            - By login = By.cssSelector("input[value='Login']");
    - Create a LoginTest class under 'ppack' package
        - Create a login() method - @Test annotated method under this class
        - Configuring TestNG for the above @Test annotation

        - Resolving the error with latest TestNG jar file configuration
    - Write Selenium code inside login() method
        - System.setProperty("webdriver.chrome.driver","D:\\SeleniumTraining Workspace\\PMDemoProj\\drivers\\chromedriver.exe");
        - WebDriver driver = new ChromeDriver();
        - driver.manage().window().maximize();
        - driver.manage().timeouts().implicitlyWait(10,TimeUnit.SECONDS);
        - driver.get("http://tutorialsninja.com/demo/index.php?route=account/login");
        - Create methods to return objects in the LoginPage class of object repository
        - Create object for Login class of objectrepository by passing driver in the constructor
            - Login login = new Login(driver);
        - Create a constructor in the Login class of object repository
        - Access the methods using the object reference of Login class
            - login.email().sendKeys("arun.selenium@gmail.com");
            - login.password().sendKeys("Second@123");
            - login.login().click();
  - Create a MyAccount class under objectrepository package
    - Copy all the things from Login class and modify according to needs - View code
    - Create an object for MyAccount and access the method using object reference
        - MyAccount account = new MyAccount(driver);
        - Assert.assertTrue(account.account().isDisplayed());
  - Execute the Test and see everything is working fine
4. Page Factory
  - Replace By locators in Login class with @FindBy - View here
  - Add this in the constructor
    - PageFactory.initElements(driver,this);
  - Repeat the same for MyAccount class
  - Execute the Test and see everything is working fine

AutoIt

1. Search 'Download AutoIt' and download it
2. Install AutoIt
3. Create a Project and write the Selenium code to click on the 'Choose File' button
4. Go to C Drive - Program Files (x86) - AutoIt - SciTE-Lite Editor
    1. Inspect options using Au3Info
        1. ControlFocus("Open","","Edit1")
        2. ControlSetText("Open","","Edit1","C:\Users\arunm\Desktop\ExcelTestData.xlsx")
        3. ControlClick("Open","","Button1")
5. Save the file and compile it to generate exe file

6. Write the remaining code:
    1. Thread.sleep(5000);
    2. Runtime.getRuntime().exec("D:\\SeleniumTrainingWorkspace\\AutoItDemoProj\\autoiffile\\FileUpload.exe");
7. Execute the Selenium Code

Cucumber and BDD (Part 1)

1. What is Cucumber?
    1. Cucumber is a framework which supports Behavior Driven Development (BDD).
2. What is a BDD?
    1. BDD stands for Behavior Driven Development
    2. Difference between Traditional Development and Behavior Driven Development
        - In **traditional development projects**, there will be a communication gap between Customers, Developers & Testers
            - Developers misunderstand the Business requirements provided by Customers and build wrong products - View here
            - BDD can be used to overcome this problem
        - In BDD, development is driven by the behavior of the required Software.
            - Development focuses on what to be developed instead of how to develop
            - Behavior of the required Software will be communicated in a plain English language using which the Business, Developers and Tester will be on the same page
            - Example (Amazon.com) - View here to find out how the behavior of the software can be communicated in a plain English language.
3. What is Cucumber ?
    1. Cucumber is a framework which supports the implementation of BDD in Selenium Automation by doing the below:
        - Allows us to create Feature Files in plain English language.
        - Understands the Gherkin language which is used for writing the scenarios inside Feature Files in plain English language.
        - Connects the steps inside the Feature Files with the automation code written in Selenium Java language and executes.
4. Install **Cucumber Eclipse Plug-in**
    1. Launch Eclipse IDE and select 'Help' Menu > 'Install New Software' option
    2. Click on 'Add' button and provide Name as 'Cucumber'
    3. Search for 'Cucumber Eclipse IDE' and find the github page for Cucumber and find the eclipse update site for Cucumber
    4. You will see the instructions for installing
    5. Select to install the 'Cucumber Eclipse Plugin'
    6. Alternative Plugin - Natural in Eclipse IDE Marketplace
5. Create Maven Project and Configure with Cucumber & Selenium

1. Create a new Maven Project by Right clicking on the Package Explorer and selecting 'New' > 'Other'
2. Filter Maven and select 'Maven Project' option
3. Select 'Create a simple project' option
4. Provide the artifact ID as Project Name and Group ID as Package name (Say CBProj and cbPack)
5. Google 'Cucumber Maven Dependency' and click on the link https://mvnrepository.com/artifact/info.cukes which got appeared in our Search Results
6. Copy the dependency tags for 1.2.5 versions of both Cucumber JVM: JUnit and Cucumber JVM : Java in between the dependencies tags of pom.xml file and save the Project
7. Google 'JUnit Maven Dependency' and click on the link https://mvnrepository.com/artifact/junit/junit which got appeared in our Search Results
8. Copy the dependency tag for 4.11 version of JUnit in between the dependencies tags of pom.xml file and save the Project
9. Google 'Selenium Maven Dependency' and click on the link https://mvnrepository.com/artifact/org.seleniumhq.selenium which got appeared in our Search Results
10. Copy the dependency tags for 2.53.1 of Selenium Java in between the dependencies tags of pom.xml file and save the Project
11. Make the project ready for execution in different browsers by downloading and pasting the required drivers under 'drivers' folder of the project

6. Creating Feature File
    1. Create a package say 'cbPack'
    2. Right click on the 'bcPack' and select 'New' > 'File'
    3. Provide the file name as 'search.feature'
    4. Clear all the stuff which got auto-generated in the feature file
    5. Create a Feature File with various keywords like **Feature**, **Scenario**, **Given**, **When**, **And**, **Then** and **But** keywords - View File Details Here
    6. Convert the Project to Cucumber Project
        - Right Click on Project > Configure > Convert to Cucumber Project
7. Create Step definitions class
    1. Right click on the 'bcPack' and select 'New' > 'Class'
    2. Provide step-definitions for all the steps that are provided in feature file - View Code here
    3. Google chrome plugin to auto-generate the step definitions for the given feature file
        - Tidy Gherkin plugin
8. Run the individual feature using Run As > Cucumber Feature

1. Create Runner class
    - Right click on the 'bcPack' and select 'New' > 'Class' < 'Runner'

- Provide @RunWith(Cucumber.class) on the top of the class - View code here
  - @RunWith annotation is from JUnit
- Run as 'Runner' Class with JUnit and see the console and JUnit results tab

2. Install Ansi-escape-console
   - Install this by dragging 'Install' option on https://marketplace.eclipse.org/content/ansi-escape-console into the Eclipse IDE market place and see the improvised output in console
3. Providing a description into a Feature File
   - Description is optional but recommended
   - Example - Copy the description from here and paste into the Feature File
4. Understanding **Scenario Outline**
   - Create a new feature file say 'omayologin.feature' under 'bcPack' package
   - Write a login scenario using the keywords 'Scenario Outline' and '**Examples**' as shown here
   - Create a step definition class say 'OmayoLogin'
   - Create step definitions for all the steps in feature file and write Selenium Code - View code here
   - Run all the Feature Files in the Project using Runner class and see the Passing results
   - Provide wrong expected login status and the scenario should fail on running
5. Executing individual Feature Files
   - Open the required feature file
   - Right click anywhere inside the Feature File and select 'Run As' > 'Cucumber Feature' option
   - Observe that only one Feature File will be executed
6. Using **Background**
   - Open the earlier created 'search.feature' and move the common step under the Background - Click here for updated scenarios

1. **Regular Expressions -** [a-zA-Z]{1,} **,** [^\"]* **and** .*
   - Demonstrate **[a-zA-Z]{1,}**
   - Right click on the 'bcPack' and create a new Feature File say 'softwaretester.feature' - Click here for scenarios
   - Implement step definitions for the above feature steps:
     - Optimize the created step definitions by parameterizing the tests
       - Provide circular brackets around the above expression
       - Represent the words in " " and update the above expression with \"
   - Demonstrate **[^\"]*** and **.*** in the same above example
2. Generating **HTML Report**
   - Provide the below statement in Runner Class, to generate an html report
     - @CucumberOptions(**plugin={"html:target/cucumber_html_report"}**)
3. **Organizing** the feature files , step definitions and Runner class
   - Create a new folder under the Project with name say "features" and move all the feature files created so far into this folder

- Create a new package under the 'src/test/java' with name say "stepdefinations" and move all the step defection Java classes under it
- Create a new package under the 'src/test/java' with name say "runner" and move the Runner.java class under it
- Run the Runner.java class and observe that it cant locate features files and won't be able to execute them.
- To overcome this problem, update the @CucumberOptions in the Runner.java Class with features and glue attributes as shown below:
    - @CucumberOptions(plugin={"html:target/cucumber_html_report"}, **features ="features"**, **glue="stepdefinations"**)

4. Cucumber **Tags**
    - Tags are used to group the Scenarios into different categories
    - Tags start with @
    - Provide @SmokeTest tag only for the first scenario in search.feature file and first scenario in softwaretester.feature file
    - Provide @All tag for all the scenarios in all the feature files
    - Now provide specific tags like @Books, @Baby, @GoodTester, @AverageTester, @BadTester and @OmayoLogin with respective scenarios in the feature file
    - Tags can be categorized as Default tags and Custom tags
    - Examples for Default tags which are used as standard tags are @Dev, @Ignore and @wip
    - Examples for Custom tags are @Books, @Baby, @GoodTester, @AverageTester, @BadTester and @OmayoLogin
    - Provide @Ignore tag for Average Tester scenario in softwaretester.feature file
    - Provide @Dev tag for Baby scenario in search.feature file
    - Provide @wip tag for Bad Tester scenario in softwaretester.feature file
    - Update the @CucumberOptions in the Runner.java Class with tags attributes as shown below, for only executing the @SmokeTest tagged scenarios:
        - @CucumberOptions(plugin={"html:target/cucumber_html_report"}, features="features", glue="stepdefinations", **tags={"@SmokeTest"}**)
    - Update the @CucumberOptions in the Runner.java Class with tags attributes as shown below, for executing all the scenarios:
        - @CucumberOptions(plugin={"html:target/cucumber_html_report"}, features="features", glue="stepDefination", **tags={"@All"}**)
    - Update the @CucumberOptions in the Runner.java Class with tags attributes as shown below, for executing a scenario which is specific to Books:
        - @CucumberOptions(plugin={"html:target/cucumber_html_report"}, features="features", glue="stepDefination", **tags={"@Books"}**)
    - Update the @CucumberOptions in the Runner.java Class with tags attributes as shown below, for executing the scenario specific to @Books or @Baby **(ORed case)**
        - @CucumberOptions(plugin={"html:target/cucumber_html_report"}, features="features", glue="stepDefination", **tags={"@Books,@Baby"}**)
    - Update the @CucumberOptions in the Runner.java Class with tags attributes as shown below, for executing the scenarios specified with both @SmokeTest or @Books **(ANDed case)**

        - @CucumberOptions(plugin={"html:target/cucumber_html_report"}, features="features", glue="stepDefination", **tags={"@SmokeTest","@Books"}**)
    - Update the @CucumberOptions in the Runner.java Class with tags attributes as shown below, for executing all the scenarios except @Dev, @wip and @Ignore tagged scenarios
        - @CucumberOptions(plugin={"html:target/cucumber_html_report"}, features="features", glue="stepDefination", **tags={"@All","~@Dev","~@wip","~@Ignore"}**)
5. Cucumber **Comments**
    - In feature files, comments can be provided any where by starting the statements with # symbol
    - Comments are for documentation purpose and won't be executed
    - Example: Go to 'search.feature' file and provide a comment stating #This is how Background keyword can be used in the feature files
6. Using **monochrome** attribute in Cucumber options of Runner class - Example: monochrome=true
    - Uninstall ANSI Escape plugin
    - Run the scripts and see how the results are displayed in console
    - Now add the attribute monochrome as true into the cucumber options
7. Using **pretty** in Cucumber options of Runner class - Example: plugin={"pretty"}
    - Install ANSI Escape plug-in
    - Remove monochrome
    - Prints the Gherkin source with additional colors and stack traces for errors in the Eclipse Output console.

1. **Hooks**
    - Similar to TestNG annotations, Cucumber provides only two options known as Hooks
    - **@Before** and **@After** are the two Hooks options in Cucumber
        - Demonstrate Hooks by creating setup and tearDown methods for opening and closing the Browsers in selenium (Example: omayologin scenario)
        - Will be executed before and after all the scenarios in all the features files.
    - **Tagged Hooks**
        - Will be executed only before and after the scenarios in the feature files which are specified with specific tags
        - @Before("@login") and @After("@login") - Demonstrate here
2. **JUnit Assertions**: Create a feature file for Logging into http://tutorialsninja.com/demo and implement the steps
    - Create a feature file say login.feature - View
    - Create step definitions for the above feature file - view
        - Understanding Assertions - Example - Assert.assertNotNull("User has not logged in", element);
    - Navigating to step definition methods of the steps in feature files
        - Select the step in feature file, right click and select 'Find Step' option

3. **Integrating TestNG and Maven with Cucumber**
    - Generate a testng.xml file and modify like this - View
    - Add the dependency tags of 'Cucumber TestNG'
    - Extend the TestRunner class with AbstractTestNGCucumberTests
    - Run the testng.xml file
    - Run using Maven by adding 'TestNG Maven' configuration
4. Cucumber **Data Tables** for handling multiple data in a step
    - The purpose of using Data Tables is to fill long forms at a go
    - Create a sample feature file for registration in http://tutorialsninja.com/demo with the required details
    - Create a feature file say register.feature - View
    - Create step definitions for the above feature file - view
        - DataTable is the predefined class of Cucumber

- Create a Maven Quick-start Project
- Add the dependency tags for below:
    - WebDriverManger
    - Selenium Java
    - TestNG
    - Log4j (log4j-api and log4j-core)
    - Apache Commons IO
    - ExtentReports
- Create a Base class
    - Create resources package under 'src/main/java'
    - Create Base class under this package
    - Create a method for initializeBrowser inside the Base class - View here
    - Read the browser name from Properties file, instead of hardcoding it
        - create data.properties file under resources pack
        - Add this property browser=chrome to the file
        - Modify the initializeBrowser code accordingly - View here
- Create a Class for 'LoginTest' under 'src/test/java' under 'tests' package
    - Create a method say 'login' and annotate with @Test
    - Extend the LoginTest class with Base class
    - Write the code to initialize the browser and open the Application URL
    - Run the LoginTest to see, if this is executing so far.
    - Remove the URL hardcoding and read the URL from the properties file
    - Run the LoginTest to see, if this is executing after this change - View here
        - Add this property url= http://tutorialsninja.com/demo/ to the properties file
- Write the remaining code of LoginTest by creating Page Objects
    - Create a package under 'src/main/java' as 'pageobjects'
    - Create a class say 'LandingPage' under this package with page objects - View here
    - Create an object for LandingPage from LoginTest and perform required operations - View here
    - Run the LoginTest to see, if this is executing so far.

    - Create a class say 'LoginPage' under this package with page objects - View here
    - Create an object for LoginPage from LoginTest and perform required operations - View here
    - Run the LoginTest to see, if this is executing so far.
    - Create a class say 'AccountPage' under this package with page objects - View here
    - Create an object for AccountPage from LoginTest and perform required operations - View here
    - Run the LoginTest to see, if this is executing so far.
    - Remove the email and password hardcoding and read them from the properties file
    - Write @AfterMethod to close the browser - View here
- Parameterize the Test by passing multiple sets of data from DataProvider annotated method
    - Create a DataProvider annotated method and pass the data to the Test annotated method - View here
- Create @BeforeMethod
    - Organize the code for opening the Application - View here
- Create some dummy Test Classes with simple SOP statements
    - TestTwo
    - TestThree
    - TestFour
- Convert the Project to TestNG
    - Right click on the Project, Select TestNG > Convert to TestNG
    - Execute the Tests as a group using testng.xml file
- Integrate with Maven to run the Project from the command line
    - Search for Maven TestNG configuration and add the plugin for Maven Surefire plugin having configuration tags
    - Run from the command prompt - mvn test

- Configuring log4j for logs
    - Copy paste the log4j2.xml file here into the resources package - View here
    - Open the LoginTest and write the logs - View here
    - Add the tags to pom.xml file to specify where exactly log4j file is available in the project
        - Search for 'Maven filtering' in google
        - Add the tags under <build> tag
    - Run the code and observe that the logs will be printed in the logs file
- Enable Parallel execution using TestNG
    - Adding sample automation code in all the Test Classes and Run them as a group to see whether they are running one after the other
    - Modify the testng.xml file as below:
        - Separate individual classes to separate tags
        - Add parallel=tests
    - Run the code and observe that all the tests will run in parallel
- Taking screenshots for failing Tests
    - Create a listeners package under src/main/java

    - Create a Listeners class under it and make it implement ITestListener interface
        - Add TestNG library if unable to resolve the errors
    - Select 'Source' menu in Eclipse IDE < Override/Implement Methods and select all the check boxes of ITestListerner
    - Create reusable method for taking screenshots in Base class with two parameters - View here
    - Extend the Listeners class with Base class
    - Update the onTestFailure() method of Listeners class - View here
    - Add the Listeners tags in testng.xml file - View here
    - Make the driver of the Test Classes to global and public
    - Intentionally fail a test and run the testng.xml file
- Integrating ExtentReports to the framework
    - Create a package say 'utilities' under 'src/main/java'
    - Create a class under this package say 'ExtentReporter'
    - Create a method say 'getExtentReport' with this code - View here
    - Make the getExtentReport method static
    - Write extent report code into different Listeners methods (onTestStart, onTestSuccess, onTestFailure and onFinish) - View here
    - Remove parallel execution from testng.xml file and Run
    - Make ExtentReports thread-safe, by adding this code to Listeners class
        - Add parallel execution to testng.xml file
        - Add this line to the Listeners class
            - ThreadLocal<ExtentTest> extentTestThread = new ThreadLocal<ExtentTest>();
        - Add this line inside onTestStart
            - extentTestThread.set(extentTest);
        - Replace existing line with this line inside onTestSuccess
            - extentTestThread.get().log(Status.PASS,"Test Passed");
        - Replace existing line with this link inside onTestFailure
            - extentTestThread.get().fail(result.getThrowable());
    - Adding the screenshot to the ExtentReports
        - Make the takeScreenshot() method return the destination file path - View here
        - Update the onTestFailure method
            - String screenshotFilePath = takeScreenshot(testMethodName,driver);
            - extentTestThread.get().addScreenCaptureFromPath(screenshotFilePath, testMethodName);
    - Run the testng.xml file
- Applying Encapsulation in the framework
    - Make all the Page Objects as private - So far Project Download Here
- Integrating Cucumber BDD into this Framework
    - Add the dependencies for Cucumber
        - Cucumber-Java
        - Cucumber-testng
    - Verify whether the Cucumber Eclipse IDE plugin is installed
    - Create a package say 'features' under 'src/test/java'

- Create a feature file say 'Login.feature' under this package - View here
- Create a package say 'stepdefinitions' under 'src/test/java'
- Create a Class say 'Login.java' under this package
- Auto-generate the Step definitions using TinyGerkhin Chrome Plugin
- Move the automation code to respective step definitions methods - View here
- Create a runner package
- Create a Runner class under it - View here
  - Extend AbstractTestNGCucumberTests
- Generate a testng2.xml file and modify like this - View here
- Run the testng2.xml file
- Mention this testng2.xml file in maven surefire plugin of pom.xml and run using the Maven command (mvn test)

Git and GitHub

1. In real time, we work as a team, hence we need Git and GitHub
2. Create an account at GitHub, where we can host our code
3. Create a new repository say 'GitDemoRepo'
4. At a high level Git is used to move the code between local machines and GitHub Repo
5. Download git and install
6. Open Git Command Prompt and trigger the below commands:
    1. Create a local folder in your System for Git Repos
        1. Navigate to this folder using command prompt
    2. Initialize the git folder as local git repository
        1. git init
        2. .git will be created to confirm this
    3. Register yourself with Git
            1. git config --global user.name "Arun"
            2. git config --global user.email "arun.motoori@gmail.com"
    4. Clone the GitHub repo to the local repo
        1. git clone https://github.com/arunmotoori/GDR.git
    5. Go inside the cloned project folder
        1. cd GDR
    6. Copy all the Project files to the Local Git Repo folder
    7. We have to commit before pushing the code to GitHub
        1. There are two levels of commit
            1. staging and commit
        2. Adding all the project files to staging
            1. git add *
        3. Commiting the code
            1. git commit -m "first commit"
    8. Giving the address of GitHub where we need to push the local repository code

        1. git remote add origin git@github.com:arunmotoori/GDR.git
        2. We can find this command ready for us in the GitHub page
    9. Push the code to GitHub
        1. git push origin main
    10. Import this code into Eclipse IDE:
        1. Launch Eclipse IDE from a different workspace
        2. Import the Project from GitLocalRepo
        3. Do some changes to the LocalRepo code
    11. Check the changes, add to staging and commit
        1. git status
        2. git add src/test/java/tests/TestFour.java
        3. git status
        4. git commit -m "second commit comment"
        5. git status
        6. git push origin main
    12. Modify directly from GitHub - Assuming other person has changed code
        1. Get the latest code
            1. git pull origin main
    13. Create a new branch and switch to it
        1. git checkout -b sbranch (create branch and switch)
        2. Note: git checkout sbranch (Will only switch but not create)
        3. git branch (To check the current branch)
        4. Update some code and push to subbranch
    14. Switch to master branch
        1. git checkout main
        2. git pull origin master (Get the latest code from master)
        3. git status (Once everything clear we can go to next step)
    15. Merge the branch to main
        1. git merge devbranch
        2. Merges to the active branch

Jenkins (Continued)

1. Launch jenkins from command line using the below command
    1. keep jenkins.war in any folder
    2. Run the jenkins using the command - java -jar jenkins.war
2. Access localhost:8080 and login using the below credentials
    1. arunmotoori
    2. 12345
3. Install Maven Integration Plug-in in Jenkins
4. Create a new Maven Job in Jenkins
5. Select 'Git' under 'Source Code Management' and give the page URL path of GitHub Repo
6. Uncheck any selected checkbox options under 'Build Triggers'
7. Under post steps > select invoke top level Maven targets

8. Apply and Run the Job
9. Explain about build periodically

SauceLabs for Cloud Testing

1. Why Cloud?
2. Create a SauceLabs account
    1. arunmotoori
    2. Second@123
3. Create a new Maven Project
4. Configure with Selenium
5. Write the below code in main method
    1. WebDriver driver = new RemoteWebDriver(new java.net.URL(),caps);
    2. Search for 'SauceLabs Platform Configurator' and get the auto-generated capabilities code
    3. Paste the auto-generated code
    4. Search for 'saucelabs getting started with selenium website testing'
    5. And copy paste three lines of code directly inside the class
    6. Modify the username in the code
    7. Go to Account > User Settings and copy the Access Key and paste into the code
    8. Write some sample selenium code
    9. View the code here
    10. Execute the code and watch under SauceLabs > Automated > Test Results