❖ Selenium:

- **Definition:** It is an open source automation tool which provide the functionality of automating the web based/mobile based application by using different programming language.
- Testing an application or software with the help of any automation tool and executing test case is called as automation.
- For web base application the most popular and easy to use open source tool is Selenium:
  - ***Java + Selenium === Automation tool.***

- **Advantages of Selenium:**
  - Selenium is an open source tool
  - Selenium supports various programming language to write the test scripts such as Java, ruby, python, c++ etc.
  - Selenium supports various operating system such as mac, Linux, Windows etc.
  - Selenium supports multiple browser like Mozilla Firefox, Google chrome, Opera, Safari etc.
  - Selenium supports parallel test case execution.
  - Selenium uses less hardware resources.

- **Disadvantages of Selenium:**
  - Selenium does not automate captcha and barcode, image and video and audio.
  - Selenium only works on the web based applications.
  - Difficult to use, takes more time to create test scripts.
  - Selenium depends on third party frameworks like *TestNG, Cucumber for the reporting.*
  - New features may not work properly.
  - Selenium has no built-in reporting facility.

- **Advantages of Automation Testing:**
  - **Reusability of test scripts:** We did not write the test cases multiple times for same functionality so reusability is possible.
  - **Project duration reduces:** So everything depends on delivery we can reduce the project duration by using agile methodology with automation.
  - **Compatibility testing is easy:** We can execute it parallelly in automation testing.
  - Required less human efforts.
  - Using automation accuracy is more.

- **Disadvantage of Automation:**
  - Required skilled labour.
  - More money investment required than manual.
  - Automation tool maintenance required.

- **Automation testing tools:**
  - Automation testing is used to change the manual test case into a test script with the help of some automation tool.
  - There are several types of automation testing tools available in the market.
    - Selenium
    - Watir
    - QTP
    - Telerik Studio
    - Testins
    - Applitools
    - Cucumber

❖ Selenium Components:

- Selenium is totally abstract i.e. we can say that it is an interface.
- **List of Selenium Components:**
  - Selenium Remote Control (R C)
  - Selenium Grid
  - Selenium Integrated Development Environment (IDE)
  - WebDriver

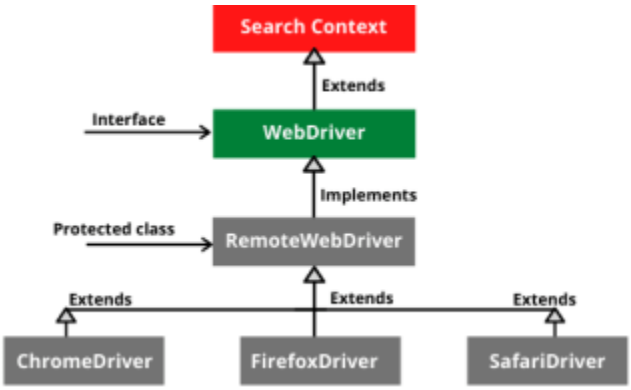
  - **Selenium RC: (Selenium Remote Control):**
    - Selenium remote control is officially deprecated by selenium and it used to work on JavaScript to automate the web applications.
    - It supports with all browsers like Firefox, chrome, safari, opera etc.
    - ***It doesn't supports for record and playback.***
    - ***Required to start server before executing the test script.***
    - Core engine is JavaScript based.
    - It is easy and small API – API's are less object oriented.
    - It does not supports listeners.
    - It does not support to test iPhone/Android applications.

- **Selenium Grid:**
  - Selenium grid is part of selenium suite that specialize in running test cases across different browsers, operating system and machine in parallel.
  - User need to configure the remote server in order to execute the tests.
  - Selenium Grid uses a hub node concept.
  - *Hub is a server or central point that control the execution on different machine.*
  - *Node is a machine which is attached to hub.There can be multiple nodes in selenium grid.*
  - We can run test on single machine called a hub and run test script on different machine with different operating system in parallel called as nodes.
  - Node is a remote device that consists of a native OS and a remote WebDriver.
  - It receives requests from the hub in the form of JSON test commands and executes them using WebDriver.
  - There can be one or more nodes in a grid.

- **Selenium Integrated Development Environment (IDE):**
  - Selenium IDE is a Firefox plugin that works on record and playback principle.
  - ***It supports for record and playback.***
  - ***Doesn't required to start server before executing the test script.***
  - It is a GUI Plug-in.
  - Core engine is JavaScript based.
  - *It only works in Mozilla Firefox browser.*
  - It is not object oriented.
  - It does not supports listeners.
  - It does not support to test iPhone/Android applications.

- **WebDriver:**
  - Selenium Web driver is used to automate web applications using browsers native methods.
  - Selenium webdriver is a browser automation framework that accepts commands and send them to a browser.
  - It is implemented through browser specific driver.
  - It control the browsers by directly communicating with it.
  - It supports with all browsers like Firefox, Chrome, Safari, Opera etc.
  - ***It doesn't supports for record and playback.***
  - ***Doesn't required to start server before executing the test script.***
  - It supports the implementation of listeners.
  - It support to test iPhone/Android applications.

❖ Selenium Architecture:



- Search context is the **super most Interface** which contains **abstract** method and **inherited** to **Webdriver interface**.
- WebDriver is an interface which contain abstract method of search context and its own abstract method.
- All the abstract methods are implemented in remote webdriver class.
- Remote webdriver is as class which implements all the abstract methods of both interfaces.
- Browser classes such as Firefox, chrome, safari, edge etc. extends to remote webdriver class.
- To achieve this we need to perform **up casting** in selenium.

- **Search Context:**
  - Search context is **super most interface** which provides two important methods.
  - Search context is the **super interface** of the Webdriver, which is extended by another interface called WebDriver.
  - All the abstract methods of search Context and WebDriver interfaces are implemented in RemoteWebDriver class.
  - Search Context interface has two methods like **find element and find elements.**

- **Selenium WebDriver :**
  - Selenium webdriver is an interface that defines **a set of methods.**
  - The webdriver has main functionality is to **control the browser**.
  - The object of the web driver is a **browser.**
  - However, implementation is provided by the browser specific classes.
  - Some of the implementation classes are *Chrome Driver, Firefox Driver, and Safari Driver* etc.

- **Selenium remote webdriver:**
  - Selenium remote webdriver **implements** the webdriver interface to execute test cases on remote server.
  - The browser driver classes like, Firefox Driver, Chrome Driver, etc. extend the RemoteWebDriverclass.

Questions:

- **What is Selenium Web driver?**
  - Selenium WebDriver is a browser automation framework that accepts commands and sends them to a browser.
  - It is implemented through a browser-specific driver.
  - It directly communicates with the browser and controls it.
  - Selenium WebDriver supports various programming languages like – Java, C#, PHP, Python, Perl, Ruby. And JavaScript.

- **What is Selenium Grid and when do we go for it?**
  - Selenium Grid can be use to execute same or different test script on multi-platform and browsers concurrently so as to achieve distributed test execution.
  - *We use Selenium Grid in the following scenarios:*
    - Execute your test on different operating systems
    - Execute your tests on different versions of same browser
    - Execute your tests on multiple browsers
    - Execute your tests in parallel and multiple threads.

- **What are the advantages of Selenium Grid?**
  - It allows multi browser testing.
  - It allows us to execute test case on multi-platform.
  - Reduces batch processing time.
  - Can perform multi-OS testing.
  - It allows running the test case in parallel there by saving test execution time.

- **When should I use Selenium IDE?**
  - Because of its simplicity, Selenium IDE should only be used as a prototyping tool, not an overall solution for developing and maintaining complex test suites.

- **What is a Node in Selenium Grid?**
  - Node is a remote device that consists of a native OS and a remote WebDriver.
  - It receives requests from the hub in the form of JSON test commands and executes them using WebDriver.
  - There can be one or more nodes in a grid.
  - Nodes can be launched on multiple machines with different platforms and browsers.
  - The machines running the nodes need not be the same platform as that of the hub.

- **What are the types of WebDriver API's that are supported/available in Selenium?**
  - Selenium Web driver supports most of the browser driver APIs like Chrome, Firefox, Safari and PhantomJS.

- **Which WebDriver implementation claims to be the fastest?**
  - The fastest implementation of web driver is the HTMLUnitDriver.
  - It is because the HTMLUnitDriver does not execute test on browser.
  - It is based on HtmlUnit.
  - It is known as Headless browser driver.
  - It is same as Chrome, Edge, Firefox driver, but it does not have GUI so we cannot see the test execution on screen.

- **Is Selenium Server needed to run Selenium WebDriver scripts?**
  - In case of selenium webdriver, it does not require to start selenium server for executing test scripts.
  - Selenium webdriver makes the calls between browser & automation script.

- **How to Start a Hub and node in Selenium Grid?**
  - Start the Hub

    java -jar selenium-server-standalone-3.14.jar -role hub

  - Connect Node to the Hub at localhost:4444

    java –jar selenium-server-standalone-3.14.jar –role node –hub

    http://hubIP:4444/grid/register

❖ WebDriver Commands:

- **Get Commands:**
  - **Get Method**
    - To launch URL in browser.
    - It is used to enter URL in a browser and it will wait for fully loaded the webpage.
    - **Syntax :-** driver.get("url");
    - driver.get("https://www.facebook.com/");
  - **Get Title Method**
    - Returns title of page
    - It is used to get/return title of webpage

- **Syntax :-** driver.getTitle();
- driver.getTitle();

- **Get CurrentURL Method**
  - Returns current URL of webpage
  - It is used to get/return current URL of webpage
  - **Syntax :-** driver.getCurrentUrl();
  - driver.getCurrentUrl();

- **Get PageSource Method**
  - **Returns HTML code of webpage**
  - **Syntax :-** driver.getPageSource();
  - driver.getPageSource();

- **Get Text Method:**
  - getText () command is used to retrieve *the inner text of the specified web element.*
  - The command doesn't *require any parameter and returns a* ***string value.***
  - String Text = driver.findElement(By.id("Text"))**.getText();**

- ***Navigate Commands/Methods:***

  Navigate is used to back, forward, refresh the browsers.

  - **navigate().to()**
    - It is used to navigate from one webpage to other webpage and load
    - **new URL/webpage in the existing window of browser**
    - **Syntax :-** driver.navigate().to("URL");

  - **Forward command**
    - This method is used to click on the forward button (arrow) of the browser window.
    - **Syntax :-**driver.navigate().forward();

  - **Back command**
    - This method is used to click on back button(arrow) of the browser window
    - **Syntax :-** driver.navigate().back();

  - **Refresh command**
    - This method is used to refresh/reloads the current webpage in the browser
    - **Syntax :-** driver.navigate().refresh();

- ***Browser Commands/Methods***
  - **Close:**
    - To close particular window
    - **This method is used to close current browser window or selenium focused browser window.**
    - **Syntax: -** driver. Close();

  - **Quit**
    - To close all windows.
    - This method is used to quit/close all windows present in browser. It will use to end selenium script.
    - **Syntax :-** driver.quit();

  - **Maximize**
    - To maximize launched browser.
    - This method is used to maximize the browser window.
    - **Syntax :-** driver.manage().window().maximize();

❖ Locators in Selenium:

- DOM stands for *Document Object Model.*
- In simple words, **DOM specifies the structural representation of HTML elements**.
- **Selenium Locators :**
  - Locators are nothing but it is technique used to find web elements which are present on webpage.
  - Each web element has its certain position on webpage so selenium locators are used to find web element from webpage.
  - Locators are methods of "**By** "class by which we can find element.
  - The locators are one of the most important parameters for *scripting base foundation* and they may lead script failure.
  - Locating elements in Selenium WebDriver is performed with the help of **findElement** () and **findElements**() methods provided by searchContext interface.
  - There are 8 locators which are used to find the web elements from the webpage.
    - TagName
    - Id
    - Name
    - ClassName
    - CSS selector

- Link Text
- Partial LinkText
- Xpath

- **tagName:**
  - A tagName is a part of a DOM structure where every element on a page is been defined via tag like input tag, button tag or anchor tag etc.
  - In the case of tagName Selenium locator, we will simply use the tag name to identify an element.
  - **Syntax:** driver.findElement(By.tagName ());

- **Name**
  - An element can be defined via multiple attributes, one such attribute is Name.
  - In case no such name matches with the defined attribute value, NoSuchElementException is raised.
  - **Syntax :** driver.findElement(By.name("name_value"));

- **className**
  - Class Name locator is used for locating web elements that are defined using the class attribute. Shown below is the DOM snapshot.
  - **Syntax :** driver.findElement(By.className("value"));

- **linkText**
  - Elements can be located via link text that is present in the hyperlinks.
  - it start with <a>tag
  - **Syntax :** driver.findElement(By.linkText("Text"));

- **partialLinkText**
  - Locating WebElements using partial link text is preferred when the link text is too long.
  - Condition is - partial link text will be in sequential manner as shown below.
  - **Syntax:** driver.findElement(By.partiallinkText("partial_text"));

- **ID**
  - ID locator in selenium is the most preferred and fastest way to locate desired web elements on the page.
  - ID Selenium locators are unique for each element in the DOM.
  - **Syntax:** driver.findElement(By.id("id value"));

- **CSS selector**
  - **CSS Selectors** are one of the locator strategies offered by Selenium to identify the web elements.
  - The **CSS Selectors** mainly use the character sequence pattern, which identifies the web elements based on their HTML structure.
  - Locating an element using CSS selector may seem a little difficult than using attributes like id, name, link, etc. but it's one of the **most efficient** strategies to locate dynamic elements that don't have consistent HTML attributes.
  - CSS Selector syntax is quite similar to the XPath syntax. It can be represented syntactically as follows:
  - *Syntax :* node[attribute_name = 'attribute_value']
  - Node is the tag name of the HTML element, which needs to locate.
  - *attribute_name* is the name of the attribute which can locate the element.
  - *attribute_value* is the value of the attribute, which can locate the element.

- **Xpath**:
  - Xpath or XML path is a query language for selecting the nodes from XML documents.
  - Xpath is one of the locators supported by selenium webdriver.
  - **Types of X-path :**
    - Absolute Xpath
    - Relative Xpath
  - **Absolute Xpath:**
    - An absolute Xpath is a way of locating an element using an XML expression beginning from root node i.e. html node in case of web pages.
    - In Xpath a single slash is used for creating Xpath with absolute paths beginning from root node.
    - Xpath which gets frame by using only single forward slash **'/'** is called ***absolute x path.***
    - Absolute Xpath will traverse entire HTML from root node/html.
    - Navigating from root of the parent to the immediate child it is nothing but absolute Xpath.
    - *PARENT HTML >> CHILD*
    - *But the disadvantage of the absolute XPath :*

      If there are any changes made in the path of the element then that XPath gets failed.

- Absolute Xpath are generally not preferable to use as they gets change dynamically.
- Copy as full Xpath is known as *absolute Xpath.*
- **Syntax:** (/tagname/tagname/tagname)

- **Relative Xpath:**
  - An absolute Xpath is a way of locating an element using an XML expression beginning from anywhere in the html document case.
  - There are different way of creating Xpath which are used for creating robust Xpath (unaffected by changes in other elements).
  - The Xpath which gets frame by using double forward slash **"//"** is called as ***relative Xpath.***
  - It can search elements anywhere on the webpage, means no need to write a long Xpath.
  - We can start from the middle of HTML DOM structure.
  - Advantage of relative is that we can traverse *from parent to child* by skipping the intermediate rows.
  - This is highly preferable over the other kind of Xpath.
  - Copy as Xpath is known as *relative Xpath.*

- **Xpath by using attribute name.**
  - We can use different attributes and its values and create Xpath.
  - **Syntax :** //tagname[@attribute = 'value'];
  - Eg: //input[@id = 'lp123'];

- **Xpath by using text function:**
  - If we want to find element which is present along with link.
  - **Syntax :** //tagname[text() = 'value']
  - Eg: //a[text () = 'Forgot Password?']

- **Xpath by index**
  - When we write an Xpath and if duplicates occur.
  - Whenever we have multiples nodes matching and we don't have any other options for the other attributes.
  - Then we can use index and give numbers to it.
  - **Syntax** : (whole Xpath expression)[number]
  - Eg : (//input[@name = 'Login'])[2]

- **Xpath by using contains function.**
  - **By attribute**
    - If there are some characters of attribute which gets change dynamically (with respect to time) then it is prefer to use contains in the Xpath which provides the flexibility to create the Xpath by using some of the characters.
    - **Syntax :** //tagname[contains(@ attribute, "value")]
    - Eg: //input[contains(@id, '567')]

  - **By text function**
    - **Syntax :** //tagname[contains(text(),"value")]
    - Eg : //a[contains(text(),'often pass')]

Questions:

- **What is an alternative to driver.get () method to open a URL using Selenium WebDriver?**
  - We can use driver.navigate ().To ("URL") method to open a URL.

- **What is the difference between driver.get ("URL") and driver.navigate ().to ("URL") commands?**
  - driver.get () is used to navigate particular URL (website) and wait till page load.
  - driver.navigate () is used to navigate to particular URL and does not wait to page load.
  - It maintains browser history or cookies to navigate back or forward.

- **What are the different types of navigation commands in Selenium?**
  - **Different navigation commands in selenium are: –**
  - navigate ().to (); -
  - Navigate ().forward (); -
  - Navigate ().back (); -
  - Navigate ().refresh ();

- **How to fetch the current page URL in Selenium WebDriver?**
  - We can use the method to get the current page URL.

- **How can we maximize browser window in Selenium WebDriver?**
  - We can use the maximize () method to maximize browser window.
  - driver.manage ().window ().maximize ();

- **What are the different ways for refreshing the page using Selenium WebDriver?**
  - Browser refresh operation can be performed using the following ways in Selenium.
    - **Refresh method**
      - driver.manage ().refresh ();
    - **Get method**
      - driver.get ("https://www.google.com");
      - driver.get (driver. GetCurrentUrl ());
    - **Navigate method**
      - driver.get ("https://www.google.com");
      - driver.navigate.to (driver. GetCurrentUrl ());
    - **Send Keys method**
      - driver. findElement (By.id("username")).sendKeys (Keys.F5);

- **What is the difference between driver.getWindowHandle () and driver.getWinowHandles () in Selenium WebDriver and their return type?**
  - **driver.getWindowHandle()** – To get the window handle of the current window. Returns a string of alphanumeric window handle
  - **driver.getWindowHandles()** – To get the window handle of all current windows. Return a set of window handles.

- **How to find more than one web element in the list?**
  - We can find more than one web element by using the findElements () method in Selenium.
  - List elements = driver.findElements (By.tagName ("a"));

- **List some scenarios which we cannot automate using Selenium WebDriver?**
  - Automating captcha is not possible using selenium webdriver.
  - We cannot read bar code using Selenium WebDriver.
  - We cannot automate OTP submission.

- **How to resize browser window using Selenium WebDriver?**
  - driver.manage ().window ().maximize ();

- **What is XPath Axes and what are the different Axes available?**
  - XPath axes search different nodes in XML document from current context node.
  - XPath Axes are the methods used to find dynamic elements.
  - **Preceding:** Select all nodes that come before the current node.
  - **Following:** Selects all elements that come after the current node.
  - **ancestor:** The ancestor axis selects all ancestors element (grandparent, parent, etc.) of the current node.

- **descendant:** Selects the descendants (child or grandchild) of the current node.
- **child:** Selects all children elements of the current node.
- **parent:** Selects the parent element of the current node.
- **Preceding-sibling:** Select the **Preceding** siblings of the context node.
- **Following-sibling:** Select the following siblings of the context node.

- **What is the default timeout of Selenium WebDriver?**
  - Default timeout is 30 seconds

- **What is the major differences between XPath Expressions and CSS Selectors?**
  - Using XPath we can traverse both forward and backward whereas CSS selector only moves forward in HTML DOM.

- **Which package can be imported while working with WebDriver?**
  - Following package can be imported:
    - *org.openqa.selenium.WebDriver.*

- **How to retrieve typed text from a text box?**
  - String text = driver.findElement(By.id("textBox")).getText ();

- **How to send alt or shift or control or enter or tab key in Selenium WebDriver?**
  - We can use sendKeys () method to send any key.
    - driver.findElement(element).sendKeys(Keys.TAB);

- **What is HtmlUnitDriver?**
  - HTML UnitDriver is the most light weight and fastest implementation headless browser for of WebDriver. It is based on HtmlUnit.
  - It is known as Headless Browser Driver.
  - It is same as Chrome, IE, or Firefox driver, but it does not have GUI so one cannot see the test execution on screen.

- **What happens when I write WebDriver driver new WebDriver ()?**
  - **Explanation:** WebDriver is an interface that is available in Selenium jar files. *driver is webDriver reference variable.*
  - New is a keyword, we use to create an instance of the class (browser class).

- **What happens if we write ChromeDriver driver new ChromeDriver?**
  - If you use ChromeDriver driver = new ChromeDriver();
  - The ChromeDriver instance which will get created through that we will be only able to invoke and act on the methods *implemented by ChromeDriver* and supported by *Chrome Browser only.*

- **Which locator is claim to be fast Xpath and CSS Selector?**
  - CSS selector locator is claim to be fast because it is short.

- **How to clear the text inside the text box fields using Selenium WebDriver?**
  - Syntax: driver.findElement (By.Id ("textbox_id")).clear ();

- **How to get an attribute value of an element using Selenium WebDriver?**
  - driver.findElement (By.Id ("button_id")).get Attribute ("text");

- **How to press Enter key on text box in Selenium WebDriver?**
  - driver.findElement (By.Id ("button_id")).sendKeys (keys. ENTER);

- **How to pause a test execution for 5 seconds at a specific point?**
  - We can pause test execution for 5 seconds by using the wait command.
  - **Syntax: driver.wait (5);**
  - **Thread.sleep (5000);**

- **Is Selenium Server needed to run Selenium WebDriver scripts?**
  - In case of Selenium WebDriver*, it does not require to start Selenium Server* for executing test scripts.
  - Selenium WebDriver makes the calls *between browser & automation script.*

- **What happens if we run this command driver.get ("www.google.com")?**
  - It will load a new web page in the current browser window with the website URL set to "www.google.com".
  - This is done using an *http get operation*, and the method will block until the load is complete.

- What is meaning of this statement **System.setProperty("webdriver.chrome.driver" ,''value'');**
  - **System**: System is a class belongs to the java. Lang package.
  - **SetProperty**() : It is used to set properties for the desired browser that to be used in the test automation.
  - ***The setProperty()methods has two attributes:***
  - **PropertyName** : It represents the name of the browser specific driver.
  - **Value**: value point to the path of that browser driver.

- **Cookies in Selenium Cookies:**
  - Cookies are the files created by websites which you visit and is sent to your browser.
  - It helps the site to remember information about your visit, which makes online experience easy by saving browsing data.

- How can we delete cookies in selenium.
  - **Syntax:** driver. manage ().deleteAllCookies ();

❖ Dimension and Point Class:

- **Dimension Class :**
  - While doing automation testing using selenium webdriver, there will be number of web elements and if we want to test stability and position of element is constant on webpage or not.
  - For that we need to change size of browser window and check it.
  - So, we have dimension class which help us to resize browser window with specified co-ordinate i.e., Height and Width.
  - We need to import the statement :
    - *'import org.openqa.selenium.Dimension'*
  - Dimension d = new Dimension(width,height);
    - **Dimension :** Class from webdriver along with width, height co-ordinate value
    - d : Reference variable
  - **Syntax :** Dimension d=new Dimension(width,height);
    - driver.manage().window().setSize(d);
  - If we want to know size of browser need to use get Size () method so it will return size in the form co-ordinate.
    - Syntax: driver.manage().window().getSize();

- **Point Class :**
  - Whenever we want to change the location/position of browser window.
  - We will use point class along with **setposition ()** method.
  - We need to import the statement :
    - *'import org.openqa.selenium.Point'*
    - Point p = new Point(x,y);
    - **Point : Class** from webdriver along with x,y co-ordinate value
    - p : Reference variable
    - **Syntax :** Point p = new Point(x,y);
    - driver.manage().window().setposition(p);
  - If we want to know *position of browser* need to use **getPosition ()** method so it will return position in the form of co-ordinate.
    - **Syntax :** driver.manage().window().getPosition();

❖ Window Handling in Selenium:

- **Window in Selenium:**
  - A window in any browser is the main webpage on which the user is landed after hitting a link/URL./WebElement.
  - Such a window in selenium is referred to as the parent window also known as the main window.
  - In automation, when we have multiple windows in any web application, the activity may need to switch control among several windows from one to other in order to complete the operation.
  - After completion of the operation, it has to return to the main window i.e. parent window in Selenium.

  - **Identify parent window and child windows:**
    - When a user hits a URL, a webpage opens. This main page is the parent window i.e. the main window on which the user has currently landed.
    - All the windows which will open inside your main window will be termed as child windows.

  - **Window Handle:**
    - A window handle is a unique identifier that holds the address of all the windows.
    - This is basically a pointer to a window, which returns the ***"String"*** value.
    - Window handle function helps to getting the ID/Handles of all the windows.
    - Each window having unique window ID.
    - To handle window popup we need to change focus of selenium from main page to window popup for that we required ID of window.
    - *To change focus of selenium from main page to window popup:*
    - Syntax : driver.switchTo().window(ID);

  - **There are 2 methods to handle window :**
    - **Driver.getWindowHandle();**
      - When the site opens, we need to handle the main window by driver.getWindowHandle ().
      - This will handle the current window that uniquely identifies it within this driver instance.
      - Its return type is String.
      - **Syntax:** String ObjVar = driver.getWindowHandle();

- **Driver.getWindowHandles();**
  - To handle all opened windows by web driver, we can use driver.getWindowHandles ().
  - And then we can switch window from one window to another in a web application.
  - Its return type is Iterator<String>.
  - **Syntax:** set<String> objVar = driver.getWindowHandles();

❖ Screenshot In Selenium :

- **Why we required screenshot :**
  - Whenever we are testing an application, if defects will find to us we required some kind of proof.
  - Whenever script (test case) get pass or fail we need to show results and in those scenarios, screenshot behaves most important role as proof.
  - The script can take a screenshot, which helps to check the application functionality when the test execution completes.
  - Screenshots also help us when the test case fails so that we can identify what went wrong in our script or application.

  - **Capture screenshot in selenium?**
    - To capture screenshot in selenium web-driver we need to type caste driver object along with takes screenshot interface.
    - TakesScreenshot ts = ((TakesScreenshot).driver);
    - Then we need to call method getScreenshot() and pass argument as OutputType.File
    - File source = ts.getScreenshotAs(OutputType.FILE);
    - Which get screenshot of page but available in the internal memory let's stored in source file. So to move screenshot from source file to our destination, provide path of folder in File class from java.io
    - File des = new File(path);
    - With the help of FileHandler.copy(source,des) we can easily get screenshot in specified location/folder
    - FileHandler.copy(source,des);

❖ Iframe in Selenium Webdriver:

- iframe in Selenium Webdriver is a web page which is embedded in another web page
- Or an HTML document embedded inside another HTML document.
- The iframe is used to add content from other sources like an advertisement into a web page.
- The iframe is defined with the ***<iframe> tag.***

- ***How to identify the iframe :***
  - We cannot detect the frames by just seeing the page or by inspecting web element or with other plugin.
  - Right click on the element, if we find the option like 'This Frame' then it is an iframe.
  - Right click on the page and click *'View Page Source'* and Search with the *'iframe',* if you can find any tag name with the *'iframe'* then we can say the page consist.

- **How to switch over the elements in iframes:**
  - **Switch to the frame by index:**
    - Index is one of the attributes for frame handling in selenium through which we can switch to it.

      driver.switchTo ().frame (0);

      driver.switchTo ().frame (1);

  - **Switch to the frame by Name or ID:**
    - Name and ID are attributes for frame handling in Selenium through which we can switch to the iframe.

      driver.switchTo ().frame ("iframe1");

      driver.switchTo ().frame ("id of the element");

  - **Switch to the frame by Web Element:**
    - We can switch to the iframe using web element.

      driver.switchTo().frame(WebElement);

  - *How to switch back to the Main Page.*
    - To move back to the parent frame

      driver.switchTo ().parentFrame ();
    - **To get back to the main (or most parent) page,**

      driver.switchTo ().default Content ();

❖ JavaScriptExecutor :

- JavaScript Executor is an interface that helps to execute java Script functions through selenium Web driver.
- JavaScriptExecutor provides two methods *"**executescript**"* & *"**executeAsyncScript**"* to run JavaScript on the selected window or current page.

- **Scroll down a page using JavaScript:**
  - JavaScriptExecutor js = (JavaScriptExecutor) driver;
  - js.executeScript ("window.scrollBy (0, 1000)");

- **Scroll down a page up to Web Element:**
  - JavascriptExecutor js = (JavascriptExecutor) driver;
  - js.executeScript ("arguments [0].scrollIntoView ();", Element);

- **What is the alternative way to click on login button**
  - JavascriptExecutor js = (JavascriptExecutor) driver;
  - js.executeScript ("arguments [0].click ();", button);

- **What is the alternative way to send keys**
  - JavascriptExecutor js = (JavascriptExecutor) driver;
  - WebElement username =
    driver.findElement (By.*xpath* ("//input [@name='username']"));
  - js.executeScript("arguments[0].value='Admin'", username);

❖ **Handling of List Box (Drop down):**

- List box is web element which is available on webpage, has multiple options and user need select one of them.
- Control element that allows the user to select one or more items from a list contained within a static, multiple line text box.

- **Handling of List Box:**
  - Whenever we want to handle listbox we have to find that web element and store in the reference variable.
  - In list there will be multiple options are present and we need to select one of them for that create object of *Select class along with ref variable.*

    Select s = new Select (web element);

- **We have 3 types of method:**
  - SelectByIndex(i);
  - SelectByValue("value");
  - SelectByVisibleText("text");

- **For example:**

  - WebElement xyz= driver.findElement(By.xpath("Xpath"));
  - Select s=new Select(xyz);
  - s.selectByIndex(5);
  - s.selectByValue("5");
  - s.selectByVisibleText("1997");

- **Count of options:**
  - We have methods getoptions () and size () which help us to find count options from particular list box
  - Below statement gives us total count of options.

```
Select s=new Select();
List all_options = s.getOptions();
int count=all_options.size();
SOP(count);
```

- **Print all Options from listbox:**

```
for (int i=0; i<count; i++)
{
String s=all_options.get (i).getText();
SOPln(s);
}
```

- *2nd Way to handle dropdown: if select tagname is not present in DOM then we can handle the dropdown or listbox by using find elements Methods*

```
List<WebElement> daylist = driver.findElements(By.xpath("(//*[@id='day'])//option"));

int count = daylist.size();

for (WebElement day : daylist) {

    System.out.println("The Daylist as :"+ " "+  day.getText() + " " );

     if (day.getText().equals("27")) {

          day.click();
     }
    }
```

- **Use of Actions Class in Selenium WebDriver:**

  - **Drag and Drop:**
    - This is an action performed with a mouse when a user moves (drags) a web element and then places (drops) it into an alternate area.
    - *dragAndDrop(WebElementsource, WebElement target):*
    - This method performs left click, hold the click operations to hold the source element, moves to the location of the target element and then releases the mouse click.
    - *Syntax:*

```
Actions a = new Actions(driver);
WebElement source = driver.findElement(By locator);
WebElement target = driver.findElement(By locator);
a.clickAndHold (source).moveToElement (target).release ().build
().perform ();
```

- **clickAndHold() :**
  - *dragAndDrop()* method first performs *click-and-hold* at the location of the source element.
  - *moveToElement() :* Then source element gets moved to the location of the target element.
  - *release() :* Finally, it releases the mouse.
  - *build () :* build () method is used to perform action on multiple webelement at a time means it help to combine all the webelement to perform action. The build is executed in perform method internally.
  - *perform () :* method used to perform single action on single click.

- *dragAndDropBy(WebElementsource, int xOffset, int yOffset)*

  - This method clicks & holds the source element and moves by a given offset, then releases the mouse. Offsets are defined by x & y.
  - xOffset is horizontal movement (pixel value)
  - yOffset is a vertical movement (pixel value)
  - *Syntax :*

```
Actions a = new Actions (driver);
WebElement from = driver.findElement(By locator);
WebElement to = driver.findElement(By locator);
a.dragAndDropBy (from, xOffset, yOffset).release ().build ( ).perform ();
```

- *dragAndDrop(Source, target)*
  - By using this method we can directly drag and drop with our located source and target.
  - *Syntax :*

```
Actions a = new Actions(driver);
WebElement source = driver.findElement(By locator);
WebElement target = driver.findElement(By locator);
a.dragAndDrop(source, target).perform();
```

- **Some of Click operation also we can do by using Action Class:**
  - **click() :**
    - This method we used to single click on any webelement.
    - *Syntax :*

```
Actions a = new Actions(driver);
WebElement r = d.findElement(By.xpath("//a[@title='jQuery UI']"));
```

```
a.click();
```

- **doubleClick() :**
  - This method we used to double click on any webelement.
  - *Syntax :*

```
Actions a = new Actions(driver);
WebElement r = d.findElement(By.xpath("//a[@title='jQuery UI']"));
a.doubleClick(r).perform ();
```

- **contextClick() :**
  - This method we used to right click on any webelement.
  - *Syntax :*

```
Actions a = new Actions(driver);
WebElement r = d.findElement(By.xpath("//a[@title='jQuery UI']"));
a.contextClick(r).perform();
```

- **Some of Keyword action also we can do by using action class :**
  - **For Autosuggestion:**
  - Below code here we have selected apple search 3rd option:

```
System.setProperty("webdriver.chrome.driver", "E:\\D_ChromeDriver\\chromedriver.exe");
            WebDriver driver = new ChromeDriver () ;
            driver.get("https://www.google.com/");
            driver.manage().window().maximize();
      WebElement textfield = driver.findElement(By.xpath("//*[@name='q']"));
                textfield.sendKeys("Apple");
                Thread.sleep(2000);
                Actions act = new Actions (driver);
        act.sendKeys (textfield, Keys.ARROW_DOWN).sendKeys (Keys.ARROW_DOWN).sendKeys
(Keys.ENTER).build ().perform ();
```

- **For copy and paste data from one field and to enter in other field:**

```
	System.setProperty("webdriver.chrome.driver",
"E:\\D_ChromeDriver\\chromedriver.exe");

		WebDriver driver = new ChromeDriver () ;

	driver.get("https://mbasic.facebook.com/reg/?cid=103&refsrc=deprecated&_rdr");

		driver.manage().window().maximize();

		WebElement namefield =
driver.findElement(By.xpath("//*[@name='firstname']"));

		Actions act = new Actions (driver);

		//to select all

		act.sendKeys(namefield,
"Tanvir").keyDown(Keys.CONTROL).sendKeys("a").keyUp(Keys.CONTROL).build().perform();

	// to copy data from one field
	act.keyDown(Keys.CONTROL).sendKeys("c").keyUp(Keys.CONTROL).build().perform();

		WebElement lastname =
driver.findElement(By.xpath("//*[@name='lastname']"));

		lastname.click();

		// to paste data to other field
act.keyDown (Keys. CONTROL).sendKeys ("v").keyUp (Keys.CONTROL).build ().perform ();
```

- **Tab Actions execution by Action Class:**

```
System.setProperty("webdriver.chrome.driver", "E:\\D_ChromeDriver\\chromedriver.exe");

		WebDriver driver = new ChromeDriver () ;

	driver.get("https://mbasic.facebook.com/reg/?cid=103&refsrc=deprecated&_rdr");

		 driver.manage().window().maximize ();

driver.findElement(By.xpath("//input[@name='firstname']")).sendKeys("Tanvir");

		 Actions act = new Actions (driver);

		act.sendKeys(Keys.TAB).sendKeys("Shinde").build().perform();

		act.sendKeys(Keys.TAB).sendKeys("8208159093").build().perform();

	WebElement gender = driver.findElement(By.xpath("//*[@value='2' and @id='sex']"));

	act.sendKeys (gender, Keys.TAB).sendKeys (Keys.ENTER).build().perform ();
```

❖ Pop-Up Handling In Selenium WebDriver :

- **Popup:**
  - Small window which display on webpage with some information when we perform operation on web element.
  - *Types of popup:*
  - ***Selenium supported***
    - Hidden Division popup
    - Child browser(window) popup
    - Alert popup

    - **Hidden Division popup:**
      - Hidden division pop is nothing but HTML code which is hidden initially, hidden division pop up also known *as dialog or overlay.*
      - The overlay is triggered when an application user performs specific tasks like clicking a button, submitting a form, or on page load.
      - Ex. calendar popups, Contact forms, Error and success Messages.
      - Hidden division pop cannot be moved here and there.
      - We can inspect the overlay.
      - This is not java script popup.
      - We can resize and customize the content of the pop-up.
      - If the content is more than the pop-up size, pop up shows scroll bar.
      - When hidden division pop up is opened, pop up takes the focus from the application.
      - When pop up is closed, focus automatically goes to the application.
        We can write Xpath for the Name text bar : //input[@type='text']
      - Send text for the Name, using send Keys in selenium.
      - No Special Operation required to handle hidden division popup.

    - **Child browser popup :**
      - **How to identify:**
      - We can inspect this Popup with right click-Inspect.
      - We can move this Popup.
      - Popup will be colorful.
      - Will have minimize and maximize button.

- **How to handle :**
  - We can close the child browser window in Selenium webdriver. The **getWindowHandles** and **getWindowHandle** methods can be used to handle child windows.
  - The **getWindowHandles** method is used to store all the opened window handles in the Set data structure.
  - In order to switch its focus from the parent to the child window, we shall take the help of the switchTo ().window () method and pass the window handle id of the child window as an argument to the method.
  - Then to move from the child window to the parent window, we shall take the help of the switchTo ().window () method and pass the window handle id of parent window as an argument to the method.

Syntax: driver. SwitchTo ().window ()

- **Alert popup:**
  - An alert is a small message box which appears on screen to give some information or notification to the user.
  - It notifies the user with some specific information or error, asks for permission to perform certain tasks and it also provides warning messages as well.

- *Simple Alert(with Ok Button):*
  - These alerts are just informational alerts and have an OK button on them.
  - Users can click on the OK button after reading the message displayed on the alert box.

```
driver. SwitchTo ().alert ().accept ();
```

- *Confirmation Alert. (With Ok and cancel button)*
  - These alerts get some confirmation from the user in the form of accepting or dismissing the message box.
  - Users can only read the message and provide the inputs by pressing the OK/Cancel button.

  - *To accept the confirmation alert:*

```
driver. SwitchTo ().alert ().accept ();
```

  - *To reject the confirmation alert:*

```
driver.switchTo().alert().dismiss();
```

- *To get the text present on confirmation alert:*

```
String textonalert = driver.switchTo ().alert ().get Text ();
```

- **Prompt Alert.**
  - This *Prompt Alert* asks some input from the user and selenium web driver can enter the text using *send keys(" input…. ").*
  - In prompt alerts, some input requirement is there from the user in the form of text that needs to enter in the alert box.

```
Alert galert = driver.switchTo().alert();

galert.sendKeys("12345");

galert.accept();
```

❖ Synchronization (Selenium Waits):

- Matching selenium script speed with browser/application speed called as *Synchronization.*
- Waits are nothing but the dynamic component which waits for a particular event and once it founds it move forwards irrespective of the time remaining hence they are called as *dynamic waits.*
- **Types of waits :**
  - Implicit wait
  - Explicit wait
  - Fluent wait

- **implicit wait in Selenium:**
  - This is also known as *global wait* because once it applies at a point it will be applicable to *all the web elements* available below it. On the completion of configured time it throws the ***NoSuchElement exception.***
  - The implicit wait will tell the web driver to wait a certain amount of time before it throws a *"No Such Element Exception."*
  - ***The default setting of implicit wait is zero***.
  - Once we set the time, the web driver will wait for that particular amount of time before throwing an exception.
  - It can be configurable with different time unit such as ***SECONDS, MINUTES, and HOURS etc.***
  - **Syntax:** driver.manage ().timeouts ().implicitlyWait (30, TimeUnit.SECONDS)

- **Explicit Wait in Selenium:**
  - The explicit wait in selenium is used to tell the web driver to wait for certain conditions *(expected conditions)* or maximum time exceeded before throwing ***"ElementNotVisibleException"***
  - This only verifies whether that element is available or not but for the particular conditions we won't be able to use it further.
  - For that purpose we have to use explicit wait.
  - It can't be configurable with different time units .***It has only SECONDS as a Time Unit.***
  - **Syntax:**

```
WebDriverWait wait = new WebDriverWait (driver, 10);
WebElement element = wait.until
(ExpectedConditions.elementToBeClickable (By.id ("button")));
```

- **The following are the Expected Conditions that can be used in Selenium Explicit Wait:**
  - *alertIsPresent()*
  - *elementSelectionStateToBe()*
  - *elementToBeClickable()*
  - *elementToBeSelected()*
  - *invisibilityOfTheElementLocated()*
  - *invisibilityOfElementWithText()*
  - *presenceOfAllElementsLocatedBy()*
  - *presenceOfElementLocated()*
  - *textToBePresentInElement()*
  - *textToBePresentInElementLocated()*
  - *textToBePresentInElementValue()*
  - *visibilityOfAllElementsLocatedBy()*
  - *visibilityOfElementLocated()*

- **Fluent Wait in Selenium:**
  - Each FluentWait instance defines the maximum amount of time to wait for a condition, as well as the frequency with which to check the condition.
  - Furthermore, the user may configure the wait to ignore specific types of exceptions such as **NoSuchElementExceptions** when searching for an element on the page.
  - Fluent wait have the ability to poll, timeout and exception which can be configured according to the requirement.
  - **Syntax:**

```
FluentWait wait = new FluentWait (driver)
                    .withTimeout (50, TimeUnit.SECONDS)
                    .pollingEvery (9, TimeUnit.SECONDS)
                   .ignoring (NoSuchElementException. Class);

Wait. Until (ExpectedConditions.elementSelectionStateToBe (By.xpath
("//*[@type= 'check box']"), true));
```

❖ Parameterization:

- The process of fetching information from the external source and using it into selenium test script is called as *'Parameterization'.*
- To perform the action of parameterization we need to import the third party tool that *"**Apache POI**"*.
- APECHE POI- Apache provides a very famous library POI, by using this library. We can read & write the **XLSX** file format **Excel**.

- **CLASSES TO HANDLE EXCEL-SHEET**
  - To fetch the data from excel sheet we have to use the class.

    FileInputStream f = new FileInputStream (path);
  - To export the data to the excel-sheet we have another class.

    FileOutputStream j = new FileOutputStream (path);

- **Methods:**
  - *getStringCellValue();* - it will fetch the string data from the excelsheet.
  - *getNumericCellValue();* - it will fetch only numeric value from the excelsheet.
  - *getCellType();* - it will help us to check which type of data stored in particular cell.
  - *getLastCellNumber();* - it will shows the last cell number in excel sheet.
  - *getRow();*
  - *getCell();*

- DIFFERENT INTERFACES
  - **Workbook**- This interface has two implementation classes
    - ✓ *HSSFWorkbook*- it represents .xls file
    - ✓ *XSSFWorkbook*- it represents .xlsx file

  - **Sheet**- It is also an interface; it has to two implementation classes.
    - ✓ *HSSFSheet*- it represents .xls file
    - ✓ *XSSFSheet*- it represents .xlsx file

  - **Row**- It is also an interface; it has to implementation classes.
    - ✓ *HSSFRow*- it represents .xls file
    - ✓ *XSSFRow*- it represents .xlsx file

  - **Cell**- It is also an interface; it has to implementation classes.
    - ✓ *HSSFCell*- it represents .xls file
    - ✓ *XSSFCell*- it represents .xlsx file.

- **To fetch the data from excel sheet :**

```
public static  String readData (int row , int column)throws IOException  {
		String path =".\\TestData\\MavenProjectTestData.xlsx";
	FileInputStream excelfile = new FileInputStream(path);
		//to load the workbook
		XSSFWorkbook workbook = new XSSFWorkbook(excelfile);
		// to get the Sheet from the workbook
		XSSFSheet sh1 = workbook.getSheetAt(0);
		String value = sh1.getRow(row).getCell(column).getStringCellValue();
		System.out.println(value);
		return value;
	}
```

- **To Write the data into the excel sheet :**

```
	public static void writeData ( int row , int column , String value)throws
IOException {
		String path =".\\TestData\\MavenProjectTestData.xlsx";
	  XSSFWorkbook workbook = new XSSFWorkbook();
	 XSSFSheet sheet = workbook.createSheet(value);
	  sheet.createRow(row).createCell(column).setCellValue(value);
		FileOutputStream fout = new FileOutputStream(path);
		workbook.write(fout);
	}
}
```

Questions:

- **What are the different types of exceptions you have faced in Selenium WebDriver?**
  - Different types of exceptions in Selenium are: –
    - *NoSuchElementException.*
    - *NoSuchWindowException.*
    - *NoSuchFrameException.*
    - *NoAlertPresentException.*
    - *ElementNotVisibleException.*
    - *ElementNotSelectableException.*
    - *TimeoutException.*

- **How to clear the text inside the text box fields using Selenium WebDriver?**
  - Syntax: driver.findElement (By.Id ("textbox_id")).clear ();

- **How to get an attribute value of an element using Selenium WebDriver?**
  - driver.findElement (By.Id ("button_id")).get Attribute ("text");

- **How to press Enter key on text box in Selenium WebDriver?**
  - driver.findElement (By.Id ("button_id")).sendKeys (keys. ENTER);

- **How to delete cookies in Selenium?**
  - We can use deleteAllCookies () method to delete cookies in selenium.
    driver.manage ().deleteAllCookies ();

- **What is the difference between driver.getWindowHandle () and driver.getWinowHandles () in Selenium WebDriver and their return type?**
  - *driver.getWindowHandle() –*
    - To get the window handle of the current window.
    - Returns a string of alphanumeric window handle
  - *driver.getWindowHandles() –*
    - To get the window handle of all open windows.
    - Return a set of window handles.
- **How to handle hidden elements in Selenium WebDriver?**
  - We can use the JavaScriptExecutor to handle hidden elements.
    - JavascriptExecutor js = (JavascriptExecutor) driver;
    - js.executeScript ("document.getElementById ('displayed-text').value='text123'");

- **How to find more than one web element in the list?**
    - We can find more than one web element by using the findElements () method in Selenium.
      List elements = driver.findElements (By.tagName ("a"));

- **What is JavascriptExecutor and in which case JavascriptExecutor will help in Selenium automation?**
  - JavaScriptExecutor is an **Interface** that helps to execute JavaScript functions through Selenium Web driver.
  - In case, when selenium locators do not work you can use JavaScriptExecutor.
  - You can use JavaScriptExecutor to perform a desired operation on a web element.

- **How to scroll web page up and down using Selenium WebDriver?**
  - To scroll using Selenium, you can use JavaScriptExecutor interface that helps to execute JavaScript methods through Selenium Webdriver.

```
JavascriptExecutor js = (JavascriptExecutor) driver;

//This will scroll the page till the element is found
js.executeScript ("arguments [0].scrollIntoView ();" , Element);
```

- **What is the alternative way to click on login button?**
  - We can use JavaScriptExecutor to click on login button.

```
JavascriptExecutor js = (JavascriptExecutor) driver;
js.executeScript ("arguments [0].click ();" , button);
```

- **How to scroll down a page using JavaScript in Selenium?**

```
JavaScriptExecutor js = (JavaScriptExecutor)driver;
js.executeScript ("window.scrollBy (0, 1000)");
```

- **How to handle Ajax calls or elements in Selenium WebDriver?**
  - The best approach would be to wait for the required element in a dynamic period and then continue the test execution as soon as the element is found/visible.
  - This can be achieved with **WebDriverWait** in combination with **ExpectedCondition**, the best way to wait for an element dynamically, checking for the condition every second and continuing to the next command in the script as soon as the condition is met.

```
WebDriverWait wait = new WebDriverWait (driver, wait Time);
wait.until (ExpectedConditions.visibilityOfElementLocated (locator));
```

- **How can we use Recovery Scenario in Selenium WebDriver?**
  - We can develop Recovery scenarios using exception handling i.e. by using “Try Catch Block” within your Selenium WebDriver Java tests.

- **How to switch to frames in Selenium WebDriver?**
  - For switching between frames, use driver.switchTo ().frame ().

- First locate the frame id and define it in a WebElement.

```
WebElement fr = driver.findElementById ("theIframe");
driver.switchTo ().frame (fr);
```

- **What are the DesiredCapabitlies in Selenium WebDriver and their use?**
  - The Desired Capabilities Class helps us to tell the web driver, which environment we are going to use in our test script.
  - *The set Capability* method of *the Desired Capabilities Class*, can be used in *Selenium Grid.*
  - It is used to perform a *parallel execution* on *different machine configurations.*
  - *It is used to set the browser properties* (Ex. Chrome, IE), Platform Name (Ex. Linux, Windows) that are used while executing the test cases.

- **How do you handle frames in Selenium?**
  - We can switch to frames by following methods: –
    - *By Index*

```
driver.switchTo ().frame (0);
```

    - *By Name or Id*

```
driver.switchTo ().frame ("id of the element");
```

    - *By Web Element*

```
driver.switchTo ().frame (WebElement);
```

- **Give an example for method overloading concept that you have used in Selenium?**
  - Implicit Wait in Selenium use method overloading as we can provide different Timestamp or *Time Unit like SECONDS, MINUTES,* etc.

- **How do you select a value from a drop-down field and what are the different methods available?**
  - We can select value from drop-down using methods of Select class.
  - Following are the methods:
    - *selectByVisibleText*
    - *selectByValue*
    - *selectByIndex*

- **How do you capture screen-shots in Selenium and what is the best place to have the screen-shot code?**
  - **type cast driver object with TakeScreenshot**

```
TakesScreenshot snapshot =  (TakesScreenshot) driver;
```

  - **Call getScreenshotAs method to create image file**

```
File source = snapshot.getScreenshotAs (OutputType.FILE);
```

- **Move image file to new destination**
  File destination = **new** File (path);

- **Copy file at destination**
  FileHandler.*copy*(source, destination);

- **How do you debug your automation code when it is not working as expected?**
  - *Add breakpoints* on the lines of code where it is not working.
  - Run code in debugging mode – Use different actions like *F7 (Step Into*), *F8 (Step Over), F9 (Step Out)* to debug the problem.

- **How do you implement collections in your framework?**
  - Collections can be used in framework in situations where you have to store large number of objects.
  - For example, findElements () method returns a list of all matching elements.

- **Tell me about the Selenium WebDriver architecture?**
  - WebDriver is made of following components:
    - Language Bindings.
    - JSON Wire Protocol.
    - Browser Drivers.
    - Real Browsers.
  - When a test script is executed with the help of web driver, the following tasks are performed in the background.
  - An HTTP request is generated and it is delivered to the browser driver for every selenium command.
  - The HTTP request is received by the driver through an HTTP server.
  - All the steps/instructions to be executed on the browser is decided by an HTTP server.
  - The HTTP server then receives the execution status and in turn sends it back to the automation scripts.

- **Can I navigate back and forth in a browser using Selenium WebDriver?**
  - Yes. We can use Navigate method to move back and forth in a browser.

    driver.navigate ().forward ();
    driver.navigate ().back ();

- **What kind of mouse actions can be performed using Selenium?**
  - Following mouse actions can be performed:
    - *doubleClick ():* Performs double click on the element.
    - *clickAndHold():* Performs long click on the mouse without releasing it.
    - *dragAndDrop():* Drags the element from one point and drops to another.
    - *moveToElement():* Shifts the mouse pointer to the center of the element.
    - *contextClick* (): Performs right-click on the mouse.

- **What kind of keyboard operations can be performed in Selenium?**
  - Following keyboard actions can be performed:
    - *SendKeys ():* Sends a series of keys to the element
    - *KeyUp():* Performs key release
    - *KeyDown ():* Performs keypress without release.

- **What is StaleElementException? When does it occur? How do you handle it?**
  - This Exception occurs when driver is trying to perform action on the element which is no longer exists or not valid.
  - Try to get around this by first using an explicit wait on the element to ensure the Ajax call is complete, then get a reference to the element again.

```
By byPageLoc = By.id ("element");
wait.until(ExpectedConditions.elementToBeClickable (byPageLoc));
driver.findElement(By.id("element")).click();
```

- **How to get the number of frames on a page?**

```
List frames = driver.findElement("By.id("frame-id"));
int count = frames.getSize();
```

- **How to send alt or shift or control or enter or tab key in Selenium WebDriver?**
  - We can use sendKeys () method to send any key.

```
driver.findElement(element).sendKeys(Keys.TAB);
```

- **How to set the size of browser window using Selenium?**

```
Dimension d = new Dimension (300,1080);
driver.manage ().window ().setSize (d);
```

What are the different types of exceptions you have faced in Selenium?

- **NoSuchElementException:**
  - Happens when the locators are unable to find elements on the web page.
  - Ideally, the exception occurs due to the use of incorrect element locators in the findElement (By, by) method.
  - To handle this exception, use the *wait* command.
  - Use try/catch to ensure that the program flow is interrupted if the wait command doesn't help.
  - **For Example:**

```
driver.findElement(By.id("submit")).click ();
```

  - **Exception Handling:**

```
try {
driver.findElement(By.id("submit")).click();
}
catch (NoSuchElementException e)
```

  - In this case, the exception is thrown even if the element is not loaded.
  - **Avoiding-And-Handling**: Try giving a wait command
    - The wait command below waits 10 seconds for the presence of web element with id 'submit'. Then it tries to click it. If the element is available but still the click fails, an exception is caught.

```
try
{
WebDriverWait wait = new WebDriverWait(driver,
TimeSpan.FromSeconds(10));
wait.Until(ExpectedConditions.presenceOfElementLocated(By.id("submit")));
try
{
driver.findElement(By.id("submit")).click();
}
catch (WebDriverException e)
{
System.out.println("An exceptional case.");
}
}
catch (TimeOutException e)
{
```

```
System.out.println("WebDriver couldn't locate the element");
}
```

- **NoSuchWindowException:**
  - NoSuchWindowException occurs if the current list of windows is not updated. The previous window does not exist, and you can't switch to it.
  - *To handle this exception, use webdriver methods called "driver.getWindowHandles ()."*
  - It will throw *org.openqa.selenium.NoSuchWindowException* if the window handle doesn't exist or is not available to switch.

    ```
    driver.switchTo ().window (handle_1);
    ```
  - **Avoiding-And-Handling**: We would use window handles to get the set of active windows and then perform actions on the same.

```
for (String handle : driver.getWindowHandles()) {
try {
driver.switchTo().window(handle);
}
catch (NoSuchWindowException e) {
System.out.println("An exceptional case");
}
                    }
```

- **NoSuchFrameException:**
  - When web driver is trying to switch to an invalid frame, NoSuchFrameException under NotFoundException class is thrown.
  - It will throw *org.openqa.selenium.NoSuchFrameException* if a frame "frame_11" doesn't exist or is not available.
  - driver.switchTo().frame("frame_11");
  - Exception Handling:
    - try {
    - driver.switchTo().frame("frame_11");
    - } catch (NoSuchFrameException e)
  - In this case, the exception is thrown even if the frame is not loaded.
  - **Avoiding-And-Handling**: Try to give a wait command.

```
try {
WebDriverWait wait = new WebDriverWait(driver,
TimeSpan.FromSeconds(10));
wait.Until(ExpectedConditions.frameToBeAvaliableAndSwitchToIt(frame_11));
try {
```

```
driver.switchTo().frame("frame_11");
}
catch (WebDriverException e) {
System.out.println("An exceptional case");
}
} catch (TimeOutException e) {
System.out.println("WebDriver couldn't locate the frame");
}
```

- **NoAlertPresentException :**
  - ***NoAlertPresentException* under *NotFoundException*** is thrown when web driver tries to switch to an alert, which is not available.
  - *Org.openqa.selenium.NoAlertPresentException* will be thrown If below automation code calls accept () operation on Alert () class when an alert is not yet on the screen.

```
driver.switchTo ().alert ().accept ();
```

  - **Exception Handling:**

```
try {
    driver.switchTo ().alert ().accept ();
    } catch (NoSuchAlertException e)
```

  - In this case, the exception is thrown even if the alert is not loaded completely.
  - **Avoiding-And-Handling**: Always use explicit or fluent wait for a particular time in all cases where an alert is expected. If the alert is available and still there is an exception, then it is caught.

```
try {
WebDriverWait wait = new WebDriverWait(driver,
TimeSpan.FromSeconds(10));
wait.Until(ExpectedConditions.alertIsPresent());
try {
driver.switchTo().alert().accept();
} catch (NoAlertPresentException e) {
System.out.println("An exceptional case");
}
} catch (TimeOutException e)
System.out.println("WebDriver couldn't locate the Alert");
}
```

- **InvalidSelectorException:**
  - This subclass of *NoSuchElementException* class occurs when a selector is incorrect or syntactically invalid. This exception occurs commonly when XPATH locator is used.
  - **Example:**

clickXPathButtonAndWait("//button[@type='button'][100]");

- This would throw an *InvalidSelectorExeption* because the XPATH syntax is incorrect.
- **Avoiding and Handling:** To avoid this, we should check the locator used because the locator is likely incorrect or the syntax is wrong. Using Firebug to find xpath can reduce this exception.

```
try {
clickXPathButtonAndWait("//button[@type='button']");
} catch (InvalidSelectorException e) {
        }
```

- **ElementNotVisibleException:**
  - *ElementNotVisibleException* class is a subclass of *ElementNotInteractableException* class.
  - This exception is thrown when web driver tries to perform an action on an invisible web element, which cannot be interacted with.
  - That is, the web element is in a hidden state.
  - **For example:**
  - In the below code, if the type of button with id 'submit' is 'hidden' in HTML, *org.openqa.selenium.ElementNotVisibleException* will be thrown.

```
driver.findElement(By.id("submit")).click();
Exception Handling:
try {
driver.findElement(By.id("submit")).click();
} catch (ElementNotVisibleException e)
```

- In this case, the exception is thrown even if the page has not loaded completely.
- **Avoiding-And-Handling**: There are two ways to do this. We can either use wait for the element to get completely.

```
try {
WebDriverWait wait = new WebDriverWait(driver,
TimeSpan.FromSeconds(10));
wait.Until(ExpectedConditions.visibilityOfElementLocated(By.id("submit
"));
try {
driver.findElement(By.id("submit")).click();
} catch (WebDriverException e) {
System.out.println("Exceptional case");
}
} catch (TimeOutException e)
System.out.println("WebDriver couldn't find this element visible");}
```

- **ElementNotSelectableException:**
  - This exception comes under *InvalidElementStateException* class.
  - *ElementNotSelectableException* indicates that the web element is present in the web page but cannot be selected.
  - **For example,** the below code can throw an *ElementNotSelectableException* if the id "swift" is disabled.

    Select dropdown = new Select (driver.findElement (By.id ("swift")));
    - **Exception Handling:**

      try {
      Select dropdown = new Select (driver.findElement (By.id("swift")));
      } catch (ElementNotSelectableException e)

  - In this case, exception is thrown even if the element becomes enabled after a while.
  - **Avoiding-And-Handling**: We can add a wait command to wait until the element becomes clickable. If there is still an exception, it is caught.

    try {
    WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(10));
    wait.Until(ExpectedConditions. elementToBeClickable(By.id("swift"));
    try {
    Select dropdown = new Select(driver.findElement(By.id("swift")));
    } catch (WebDriverException e) {
    System.out.println("Exceptional case");
    }
    } catch (TimeOutException e)
    System.out.println("WebDriver found that this element was not selectable.");}

- **TimeoutException:**
  - This exception occurs when a command completion takes more than the wait time.
  - Waits are mainly used in web driver to avoid the exception *ElementNotVisibleException.*
  - Sometimes test page might not load completely before next command in the program. If web driver tries to find an element in the webpage before the page completely loads, then exception *ElementNotVisibleException* is thrown.
  - To avoid this exception, waits commands are added.
  - However, if the components don't load even after the wait, the exception *org.openqa.selenium.**TimeoutException*** will be thrown.

    driver.manage ().timeouts ().implicitlyWait (10, TimeUnit.SECONDS);
    driver.get ("https://www.softwaretestinghelp.com");
  - In the above program, an implicit wait of 10 seconds is added.

- If the page *www.softwaretestinghelp.com* doesn't load in 10 seconds, then TimeoutException will be thrown.
- ***Avoiding and Handling**: To avoid this, we can manually check the average time for a page to load and adjust the wait*
- *Or, we can add explicit wait using JavaScript executor until the page is loaded.*
- *In the below example, JavaScript executor is used. After page navigation, we call JavaScript return document.readyState for 20 seconds until "complete" is returned.*

```
WebDriverWait wait = new WebDriverWait (driver, TimeSpan.FromSeconds(30));

wait.until(webDriver -> ((JavascriptExecutor)webDriver).executeScript("return
document.readyState").equals("complete"));

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

- **NoSuchSessionException:**

  - This exception is thrown when a method is called after quitting the browser by WebDriver.quit();
  - This can also happen due to web browser **issues like crashes** and WebDriver cannot execute any command using the driver instance.
  - To see this exception, the code below can be executed.

    ```
    driver. Quit ()
    Select dropdown = new Select (driver.findElement(By.id("swift")));
    ```
  - **Avoiding and Handling**: Always choose the latest stable version of browser to run Selenium web driver testcases.
  - This exception can be reduced by using driver.quit () at the completion of all tests.
  - Do not try to use them after each test case. This can lead to issues when driver instance is null and upcoming test cases try to use it without initializing.
  - The below code creates WebDriver instance in the @Before Suite TestNG annotation and destroys it in @AfterSuite TestNG annotation

    ```
    @BeforeSuite
    public void setUp() throws MalformedURLException {
    WebDriver driver = new FirefoxDriver();
    }
    @AfterSuite
    public void testDown() {
    driver.quit();
    }
    ```

- **StaleElementReferenceException:**

  - This exception says that a web element is no longer present in the web page.
  - This error is not the same as *ElementNotVisibleException.*
  - *StaleElementReferenceException* is thrown when an object for a particular web element was created in the program without any problem and however; this element is no longer present in the window or web page.
  - This can happen if there was a:
    - Navigation to another page
    - DOM has refreshed
    - A frame or window switch.

```
WebElement firstName = driver.findElement (By.id (“first_name”));
driver.switchTo ().window (Child Window);
element.sendKeys (“Aaron”);
```

  - In the code above, object firstName was created and then the window was switched. Then, WebDriver tries to type ‘Aaron’ in the form field. In this case StaleElementReferenceException is thrown.

  - ***Avoiding and Handling****: Confirm that we are trying to do the action in the correct window. To avoid issues due to DOM refresh, we can use Dynamic Xpath.*

  - In this case, we can use a dynamic Xpath like,

```
try {
driver.findElement (By.xpath (“//*[contains (@id, first_name’)]”)).sendKeys
(“Aaron”);
} catch (StaleElementReferenceException e)
```

- In the example above dynamic XPATH is used and if the exception is still found, it is caught.