**Selenium Questions:**

1. What are the components of Selenium?

**Selenium** is a powerful suite of tools for automating web application testing. Let’s explore its key components:

* **Selenium IDE (Integrated Development Environment)**:
  + Selenium IDE is a **browser extension** available for **Firefox** and **Chrome**.
  + It allows you to **record, edit, and debug** functional tests.
  + Originally known as **Selenium Recorder**, it provides an **integrated environment** for creating Selenium tests.
  + Scripts are recorded in **Selenese**, a special test scripting language for Selenium.
  + Selenese commands perform actions in a browser (e.g., clicking a link) and retrieve data from resulting pages.
* **Selenium RC (Remote Control)**:
  + Selenium RC is a **Java-based server** that accepts commands via HTTP.
  + It enables writing automated tests in **any programming language**.
  + Selenium provides client drivers for languages like **PHP, Python, Ruby, .NET, Perl, and Java**.
  + An instance of Selenium RC server is needed to launch HTML test cases.
* **Selenium WebDriver**:
  + WebDriver is the **successor to Selenium RC**.
  + It directly interacts with browsers without needing a special server.
  + WebDriver accepts commands (in Selenese or via a Client API) and sends them to a browser.
  + Browser-specific **drivers** (e.g., ChromeDriver, GeckoDriver) handle communication.
  + WebDriver supports **headless browsers** like **HtmlUnit** for faster execution.
* **Selenium Grid**:
  + Selenium Grid allows **parallel execution** of tests across **multiple machines**.
  + It helps distribute test execution on **remote systems**.
  + Useful for **cross-browser testing** and handling large test suites.

1. What is Selenium IDE?

* Selenium IDE (Integrated Development Environment) is an open-source web automation testing tool under the Selenium Suite.
* Selenium IDE allows you to record your actions in the browser.
* You interact with the web application as you normally would, and Selenium IDE captures these interactions.
* Selenium IDE is available as a browser extension for Mozilla Firefox and Google Chrome.
* You can record and play back test cases directly within these browsers.

1. What is Selenium WebDriver?

* Selenium WebDriver is not a standalone framework; rather, it is a library or API that allows you to automate interactions with web browsers.
* It is a toolbox for automation which has components like selenium webdriver, selenium IDE, selenium RC , Selenium grid .
* Selenium WebDriver- it is a framework which is used to automate interactions with the web browsers .
* seleniumIDE – Selenium IDE (Integrated Development Environment) is an open-source web automation testing tool under the Selenium Suite.
* Selenium IDE allows you to record your actions in the browser.
* Selenium RC is a testing framework that allows testers and developers to write test scripts in multiple programming languages.
* It enables you to automate UI tests for web applications against any HTTP website.
* Selenium RC consists of two main parts:
* Client libraries: These libraries are available in various programming languages.
* Server: The server launches and manages browser sessions automatically.
* Selenium Grid is a feature in Selenium that allows you to run test cases in different machines across different platforms.

1. How do you specify browser configurations with Selenium WebDriver?

* WebDriver waits until the load event fire is returned.

ChromeOptions chromeOptions = new ChromeOptions();

chromeOptions.setPageLoadStrategy(PageLoadStrategy.NORMAL);

WebDriver driver = new ChromeDriver(chromeOptions);

WebDriver waits until DOMContentLoaded event fire is returned.

ChromeOptions chromeOptions = new ChromeOptions();

chromeOptions.setPageLoadStrategy(PageLoadStrategy.EAGER);

WebDriver driver = new ChromeDriver(chromeOptions);

1. Which web driver implementation is fastest?

The fastest implementation of WebDriver is the HTMLUnitDriver.

1. Can you show me one code example of setting Selenium Webdriver?

* import org.openqa.selenium.WebDriver;
* import org.openqa.selenium.chrome.ChromeDriver;
* public class WebDriverExample {
* public static void main(String[] args) {
* // Set the path to your ChromeDriver executable
* System.setProperty("webdriver.chrome.driver", "path/to/chromedriver");
* // Initialize ChromeDriver
* WebDriver driver = new ChromeDriver();

1. What are all different element locators are available with Selenium?

* Tag
* Id
* Name
* Css selector
* Xpath
* Linktext
* Partiallink text
* Class name

1. How to maximize the window using Selenium webdriver?

* **driver.manage().window().maximize()**
* **driver.manage().window().minimize()**

1. How do I clear content of a text box in Selenium webdriver?

.clear()

1. How to execute JavaScript function in Selenium webdriver?

import org.openqa.selenium.JavascriptExecutor;

WebDriver driver = new ChromeDriver();

javascriptExecutor js = (JavascriptExecutor)driver;

js.executeScript(“arguments[0].setAttribute(‘value’,’john’)”, element);

js.executeScript(“arguments[0].click();”,element);

1. How to select a drop-down value using Selenium webdriver?

We handle drop down using Select class and create reference of select

Select st = new Select(element);

st.selectByVisibletext(“ ”);

st.selectByValue(“4”) – if value attribute is available

st.selectByindex(2);

1. Explain about select class and its methods?

Select st = new Select(element);

st.selectByVisibletext(“ ”);

st.selectByValue(“4”) – if value attribute is available

st.selectByindex(2);

find total no of options in the dropdown

List<WebElement> option = select.getOption();

1. How to automate radio button in Selenium Webdriver?

To select a radio button, use the click() method on the WebElement representing the radio button2.

WebElement maleRadioButton = driver.findElement(By.id("option1"));

maleRadioButton.click();

You can verify if a radio button is selected using the isSelected() method. It returns true if the radio button is selected, otherwise false

boolean isMaleSelected = maleRadioButton.isSelected();

**Other Validations:**

Use isDisplayed() to check if the radio button is visible on the page.

Use isEnabled() to verify if the radio button is clickable.

1. How do you submit a form using Selenium?

* If button is present inside the form element then we will use submit and if outside of the form then we will use click.

1. difference between findElement and findElements methods?

* The findElements() method is used to locate multiple web elements on a page using different locator strategies.
* Collecting all links on a page.
* Counting the number of elements matching a specific condition.
* Validating the absence of certain elements.

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

* Unlike findElement(), which returns a single WebElement, findElements() returns a list of WebElement objects that match the specified

1. How to count total number of links in a page using Selenium Webdriver?

I will locate all the web elements using tag name called anchor tag <a>

Then I will store all the web elements in the list of webelements and then apply list.size()

List<WebElement> links = driver.findElements(By.*xpath*("//a"));

links.size()

1. How to capture page title using Selenium Webdriver?

We use driver .getTitl()

String tit = driver.getTitle();

1. How to store Current page URL using Selenium Webdriver?

String currentUrl = driver.getCurrentUrl();

1. How do you simulate browser back and forward?

* driver.navigate().back();
* driver.navigate().forward();
* The driver.navigate().to method in Selenium WebDriver is used to load a new web page in the current browser window. It is equivalent to the driver.get() method, which also opens a new web page. Here’s how you can use it:

1. What is the difference between single and double slash in Xpath?

**Single Slash (**/**)**:

* 1. **Purpose**: Used to create **absolute XPath**.
  2. **Selection Scope**: Begins selection from the **root node**.
  3. **Example**: /html selects the root HTML element.

**Double Slash (**//**)**:

* 1. **Purpose**: Used to create **relative XPath**.
  2. **Selection Scope**: Starts selection from **anywhere within the document**.
  3. **Functionality**:
     1. The first occurrence of // selects all elements that match the locator.
     2. The second occurrence of // selects all descendants of the previously selected elements.

1. How to handle dynamic xpath?

**XPath**, which stands for **XML Path Language**, is an expression language used to query or transform XML documents. It allows you to traverse among elements and attributes within an XML document. XPath provides a concise way to address specific nodes or node-sets in the XML structure. Here are the key points:

**Types of XPath**:

* 1. There are two types of XPath:
     1. **Absolute XPath**: Starts from the root node and ends with the desired descendant element’s node. It begins with a single forward-slash (/).
        1. Example: /html/body/div[1]/div[2]/div[2]/div[1]/form/div[1]/div/div[1]/div/div/input[1]
     2. **Relative XPath**: Starts from any node between the HTML page and the current element’s node (last node of the element). It begins with a double forward-slash (//).
        1. Example: //li[@id='menu-item-4335']//a[normalize-space()='Free Resources']

**Dynamic XPath Strategies**:

* 1. **Using Single Slash (**/**)**:
     1. Absolute path for element discovery.
     2. Example: /html/body/div[1]/div[2]/div[2]/div[1]/form/div[1]/div/div[1]/div/div/input[1]
  2. **Using Double Slash (**//**)**:
     1. Relative path that starts anywhere within the document.
     2. Example: //li[@id='menu-item-4335']//a[normalize-space()='Free Resources']
  3. **Using Single Attribute**:
     1. Select elements based on a single attribute.
  4. **Using Multiple Attributes**:
     1. Combine multiple attributes for precise selection.
  5. **Using AND and OR**:
     1. Combine conditions to create dynamic locators.
  6. **Using**contains()**and**starts-with():
     1. Useful for partial matching.
     2. //p[starts-with(text(), 'Practice')]
  7. **Using**text():
     1. Select elements based on their text content.
  8. **Using**last()**and**position():
     1. Handle elements based on their position.
  9. **Using XPath axes**:
     1. Methods like child, parent, ancestor, sibling, preceding, and self.
     2. Useful for locating dynamic elements.

1. Difference between navigate and geturl method ?

* get() Method:
* The get() method is used to load a web page in WebDriver.
* It waits until the web page is fully loaded and ready to return control.
* navigate().to() Method:
* The navigate().to() method is an alias for get().
* It also loads the specified URL, waits for the page to load, and behaves similarly to get().
* However, it has additional abilities:
* Browser history: It allows moving between pages using navigate().forward() and navigate().back().
* Refresh: It can refresh the current page using navigate().refresh().
* Cookie persistence: Unlike get(), it maintains cookies and session data across page loads.

1. There is id, name, XPath, CSS locator, which one should I use?

IDs are the most efficient and preferred locators. If an element has a unique ID, use it.

1. How to assert text of webpage using Selenium Webdriver?

* To assert the presence of specific text on a webpage using Selenium WebDriver, you can follow these approaches:
* Using XPath:
* Search for all elements containing the given text using XPath:
* List<WebElement> list = driver.findElements(By.xpath("//\*[contains(text(), '" + text + "')]"));
* Assert.assertTrue("Text not found!", list.size() > 0);
* This code snippet checks if the specified text exists on the page.
* Directly Checking Page Source:
* Another approach is to get the entire page source and check if the desired text is present:
* String bodyText = driver.findElement(By.tagName("body")).getText();
* Assert.assertTrue("Text not found!", bodyText.contains(text));
* This method avoids waiting for the text to appear due to implicit waits.

1. How to get element attribute using Selenium Webdriver?

In HTML, an attribute provides additional information about an element. Attributes modify the default behavior of the element or offer extra details about it.

* **By using getAttribute() method**
* String value = driver.findElement(By.id("Web element id")).getAttribute("Attribute name");

1. How to double click on element using Selenium Webdriver?

* Actions actions = new Actions(driver);
* actions.doubleClick(elementToDoubleClick).perform();
* Using Actions Class: The Actions class provides a convenient way to simulate user interactions. You can use it to perform a double click on a specific element. Here’s how:
* // Assuming you've already initialized your WebDriver instance (named 'driver')
* Actions actions = new Actions(driver);
* WebElement elementToDoubleClick = driver.findElement(By.id("yourElementId"));
* // Perform the double click action
* actions.doubleClick(elementToDoubleClick).perform();
* Replace "yourElementId" with the actual ID or other suitable locator for the element you want to double click.
* **Using JavaScript Executor (Alternative Approach): If the Actions class doesn’t work as expected, you can use JavaScript to trigger the double click event directly. Here’s an example:**
* Java
* // Assuming you've already initialized your WebDriver instance (named 'driver')
* WebElement elementToDoubleClick = driver.findElement(By.id("yourElementId"));
* // Execute JavaScript to simulate double click
* ((JavascriptExecutor) driver).executeScript("arguments[0].dispatchEvent(new MouseEvent('dblclick', { bubbles: true }));", elementToDoubleClick);
* Again, replace "yourElementId" with the appropriate locator for your target element.

1. How to perform drag and drop in Selenium Webdriver?

* To perform drag and drop we use Actions class.

Actions act = new Actions(driver)

WebElement source = driver.findElement(By.xpath(“));

WebElement source = driver.findElement(By.xpath(“));

act.dragAndDrop(source, target).perform();

for slider :

Actions.dragAndDropBy(sourceElement, xOffset, yOffset)

1. How to synchronize application window and Selenium?

By using implicit wait explicit wait and fluent wait

Implicit Wait:

The Implicit Wait in Selenium is used to tell the web driver to wait for a certain amount of time before it throws a “No Such Element Exception.”

The default setting for implicit wait is 0 seconds.

Once we set the time, the web driver will wait for the element for that time before throwing an exception.

It’s a global setting that applies to all elements on the page.

Example usage:

driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);

Explicit Wait:

Explicit Wait allows us to wait for a specific condition to occur before proceeding further.

It’s more precise and flexible than implicit wait.

We can define custom conditions (e.g., element visibility, presence, or clickability) and set a timeout for waiting.

Example usage:

WebDriverWait wait = new WebDriverWait(driver, 10);

WebElement element = wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("yourElementId")));

Fluent Wait:

Fluent Wait is an advanced form of explicit wait.

It allows us to define custom polling intervals and ignore specific exceptions during the wait.

Useful when dealing with dynamic web elements or situations where the element may appear/disappear intermittently.

Example usage:

Wait<WebDriver> fluentWait = new FluentWait<>(driver)

.withTimeout(Duration.ofSeconds(20))

.pollingEvery(Duration.ofMillis(500))

.ignoring(NoSuchElementException.class);

WebElement element = fluentWait.until(ExpectedConditions.presenceOfElementLocated(By.id("yourElementId")));

**Comparison:**

Implicit Wait sets a global timeout for all elements, which can cause unnecessary delays.

Explicit Wait and Fluent Wait allow more precise control and wait for specific conditions.

Choose explicit or fluent waits over implicit waits to avoid unnecessary delays and improve test efficiency.

1. What is Selenium WebDriver?

* Selenium WebDriver is not a standalone framework; rather, it is a library or API that allows you to automate interactions with web browsers.
* It is a toolbox for automation which has components like selenium webdriver, selenium IDE, selenium RC , Selenium grid .

1. How do I install Selenium WebDriver?

* Install Java Software Development Kit (JDK)
* **Install Eclipse IDE**

Download Selenium WebDriver:

Visit the Selenium Downloads page and download the Selenium WebDriver for Java Client Driver.

Extract the contents of the downloaded ZIP file (e.g., selenium-3.14.0.zip) to a directory (e.g., C:\selenium-3.14.0).

This directory contains all the necessary JAR files for Selenium setup.

* Configure Eclipse IDE with WebDriver:

Launch Eclipse by running eclipse.exe from the extracted Eclipse folder (e.g., C:\eclipse\eclipse.exe).

When prompted to select a workspace, accept the default location.

Create a new Java project via File > New > Java Project. Name it (e.g., “newproject”).

In the project settings, choose the appropriate execution JRE and layout.

Right-click on the newly created project and select Build Path > Configure Build Path.

Add the Selenium JAR files from the C:\selenium-3.14.0 directory to the project’s build path.

You’re now set up to write Selenium scripts in Eclipse!

Add the Selenium JAR files from the C:\selenium-3.14.0 directory to the project’s build path.

1. Can you give me an example of a test case using Selenium Webdriver?

Yes , lets create a simple test case where we open a website , enter some text into search field and then verify that search result page title contains the expected keyword .

1. Explain implicit and explicit wait.

* Implicit wait : this is a type of wait I selenium which is used for synchronization of web pages
* Wait for a fixed amount of time before execution of any further actions .
* It is set once per the session.
* It applies to all the elements of web pages.
* Less flexible as compare to explicit wait .
* If time is not sufficient then you will get **NoSuchElementException**
* **Syntax : driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(10));**
* Example : waiting for page load
* Waiting for a fixed amount of time before procedding with the next actions regardless of whether page has fully loaded or not .

Explicit wait :

* Wait until a certain condition is met.
* It applies to only specific element
* Used for specific elements or conditions
* Example : waiting for element to be clickable
* Declare once and used again and again .
* Syntax : WebDriverWait wait = new WebDriverWait(driver,10);
* Or = new webDriverWait(Duration.ofSeconds(10));
* Wait.until(ExpectedConditions.visibilityofElementLocated(element));
* Fluent wait
* Wait until certain condition is met with polling
* Advance form of explicit wait
* Applies to specific element or conditions
* Example : **waiting for an element to be visible**
* Wait <WebDriver> mywait = new Fluentwait<WebDriver>(driver)
* .withTimeout(Duration.ofseconds(30L))
* .pollingEvery(Duration.ofseconds(5L))
* .ignoring(NoSuchElementsException.class);

1. Is it possible to handle multiple windows in selenium?

Yes , by using driver.switchTo.window(windoehandles);

1. My application has lots of pop-up window, how do I work with them?

**By using alert class**

* **If it is javasvcript alert then use**
* **Alert alert = driver.switchTo().alert();**
* **Javascript alert only ok button accept (used for information or warnings on the screen )**
* **alert.accept()**
* **also write in single line driver.switchTo().alert().accept(): Clicks the “OK” button on the alert.**
* **Confirmation alert : used for ask for specific permission to perform actions**

1. How to mouse hover on an element?

By using Actions class

Actions act = new Actions(driver)

Act.moveToElement(element);

Act.moveToElement(element1).moveToElement(element2).perform();

1. How to capture screen shot in webdriver?
2. **Convert WebDriver Object to TakeScreenshot**: First, convert your WebDriver object to a TakesScreenshot object using the following code:

TakesScreenshot scrShot = ((TakesScreenshot) webdriver);

1. **Call getScreenshotAs Method**: Next, use the getScreenshotAs method to create an image file. This method captures the screenshot and stores it in memory.

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

1. **Copy the File to Desired Location**: Finally, copy the screenshot file to your desired location. For example, if you want to save the screenshot as C:/Test.png, use the following code:

**File destFile = new File("C:/Test.png");**

**FileUtils.copyFile(srcFile, destFile);**

1. How do you connect Data base from selenium?

* First, you need to establish a connection to the database. Use the DriverManager.getConnection method with the following syntax

Connection con = DriverManager.getConnection("jdbc:<dbtype>://ipaddress:portnumber/db\_name", "userid", "password");

* Replace <dbtype> with the appropriate driver for your database (e.g., "oracle" for Oracle Database or "mysql" for MySQL).
* Specify the ipaddress, portnumber, db\_name, userid, and password according to your database configuration.
* Once the connection is established, create a Statement object to send SQL queries:

Statement stmt = con.createStatement();

* Use the executeQuery method to execute your SQL queries:

ResultSet resultSet = stmt.executeQuery("SELECT \* FROM employee");

1. How do you get the data from an excel sheet?

* To retrieve data from an Excel sheet using Selenium, you can use the Apache POI library. This library provides classes and methods to read and write data from various formats of Excel files (such as .xls and .xlsx). Let’s dive into the steps:
* If you’re using Maven, add the following dependency to your pom.xml:
* Alternatively, you can download the latest POI JAR files directly from the Apache POI website and add them to your project’s classpath.
* Example using apach poi ;

**import org.apache.poi.ss.usermodel.\*;**

**import org.apache.poi.xssf.usermodel.XSSFWorkbook;**

**import java.io.FileInputStream;**

**import java.io.IOException;**

**public class ReadExcelData {**

**public static void main(String[] args) throws IOException {**

**String excelFilePath = "path/to/your/excel/file.xlsx";**

**FileInputStream fis = new FileInputStream(excelFilePath);**

**XSSFWorkbook workbook = new XSSFWorkbook(fis);**

**XSSFSheet sheet = workbook.getSheet("SheetName");**

**for (Row row : sheet) {**

**for (Cell cell : row) {**

**String cellValue = cell.getStringCellValue();**

**System.out.print(cellValue + "\t"); // Print cell values (adjust as needed)**

**}**

**System.out.println(); // Move to the next row**

**}**

**workbook.close();**

**fis.close();**

**}**

**}**

1. How do you get the data from a properties file?

* To retrieve data from a properties file in Selenium, you can use the java.util.Properties class. This class provides methods to load and read data from properties files.
* Create a Properties File:
* First, create a properties file (e.g., config.properties) that contains key-value pairs. Each line represents a property with the format key=value.
* Place this file in your project’s resources folder or any accessible location.
* Load the Properties File:
* Use the Properties class to load the properties file. Here’s an example:

**import java.io.FileReader;**

**import java.io.IOException;**

**import java.util.Properties;**

**public class ReadPropertiesFile {**

**public static void main(String[] args) {**

**try {**

**Properties props = new Properties();**

**FileReader reader = new FileReader("path/to/config.properties");**

**props.load(reader);**

**// Access properties by key**

**String databaseUrl = props.getProperty("db.url");**

**String username = props.getProperty("db.username");**

**String password = props.getProperty("db.password");**

**System.out.println("Database URL: " + databaseUrl);**

**System.out.println("Username: " + username);**

**System.out.println("Password: " + password);**

**reader.close();**

**} catch (IOException e) {**

**e.printStackTrace();**

**}**

**}**

**}**

1. What are the types of automation framework?

* Hybrid framework
* Data driven framework.
* Keyword framework
* **Linear Scripting Framework**:
  + Also known as a **record-and-playback framework**.
  + Testers record their actions (interactions with the application) and play them back as test scripts.
  + Simple to create but lacks flexibility and scalability.
* **Modular Testing Framework**:
  + Breaks down the test scripts into smaller, reusable modules or functions.
  + Each module focuses on specific functionality or feature.
  + Promotes code reusability and maintainability.
* **Library Architecture Testing Framework**:
  + Organizes test scripts into libraries or modules.
  + Libraries contain common functions or utilities (e.g., login, data validation).
  + Encourages code sharing and reduces redundancy.
* **Data-driven Testing Framework**:
  + Separates test data from test scripts.
  + Reads test data from external sources (e.g., Excel, CSV, databases).
  + Allows running the same test script with different data sets.
* **Keyword Driven Testing Framework**:
  + Uses keywords or action words to define test steps.
  + Test scripts are written in a tabular format (keywords + data).
  + Enhances test readability and maintenance.
* **Hybrid Testing Framework**:
  + Combines elements from multiple frameworks (e.g., data-driven + keyword-driven).
  + Provides flexibility and scalability.
  + Customizable based on project requirements.
* **Behavior Driven Development (BDD) Testing Framework**:
  + Focuses on collaboration between developers, testers, and business stakeholders.
  + Uses natural language (Gherkin syntax) to define test scenarios.
  + Popular tools include Cucumber and SpecFlow.

1. What is Data driven framework & Keyword Driven?

* **Data-driven Testing Framework**:
  + Separates test data from test scripts.
  + Reads test data from external sources (e.g., Excel, CSV, databases).
  + Allows running the same test script with different data sets.
* **Keyword Driven Testing Framework**:
  + Uses keywords or action words to define test steps.
  + Test scripts are written in a tabular format (keywords + data).
  + Enhances test readability and maintenance.

1. What is Selenium Grid, How Selenium Grid works?

**Selenium Grid** is a powerful tool that allows you to distribute your **test execution** across multiple machines, browsers, and platforms. Here’s how it works:

1. **Hub and Nodes**:
   * The **Hub** acts as a central control point.
   * **Nodes** are remote machines (physical or virtual) where tests run.
   * Hub routes test commands to available nodes.
2. **Parallel Execution**:
   * You define your test scripts (written in Selenium WebDriver) and specify desired capabilities (browser, OS, etc.).
   * Tests are executed in parallel across different nodes.
   * This enables faster test execution and efficient resource utilization.
3. **Configuration**:
   * Set up the Hub and register nodes with specific configurations.
   * Specify which browsers and versions each node supports.
   * Tests are automatically routed to compatible nodes.
4. **Cross-Browser Testing**:
   * Run the same test on different browsers (Chrome, Firefox, etc.) simultaneously.
   * Ensure your application works consistently across browsers.
5. **Cross-Platform Testing**:
   * Test on various operating systems (Windows, macOS, Linux).
   * Validate your app’s behavior across different platforms.
6. Difference between click and submit in selenium ?

**If button is present inside the form element then we will use submit and if outside of the form then we will use click.**

* **click():**
* **Targets a button or element.**
* **Executes JavaScript events (e.g., onclick).**
* **Useful for interacting with buttons, links, and other clickable elements.**
* **submit():**
* **Targets a form element (e.g., input field).**
* **Submits the form (like pressing Enter).**
* **Waits for the page to load.**
* **Ideal for form submissions.**

1. What is the default Port Id in Grid?

* The default port for Selenium Grid is 4444. When you start the Grid, it listens for RemoteWebDriver requests on
* http://localhost:4444
* This is where your WebDriver tests should point to when using Selenium Grid.

1. What is POM, what are the advantages of POM in selenium?

* **Page Object Model is an implementation of Object/Element Design pattern. A Page Object Model is a class which contains Reference(locators) of UI elements and Methods to perform possible interactions on web elements, then the test is the separate class, which uses the methods of those page object class whenever we need to interact with UI elements. In case of any change in UI elements, we don’t need to do any change in the test. We just simply change/add them in the respective page object class. So, we can say that all the new/modified UI element changes can be done in one class.**
* The main advantage of Page Object Model is that if the UI changes for any page, it don’t require us to change any tests, we just need to change only the code within the page objects (Only at one place). Many other tools which are using selenium, are following the page object model.
* There is clean separation between test code and page specific code such as locators (or their use if you’re using a UI map) and layout.
* There is single repository for the services or operations offered by the page rather than having these services scattered through out the tests.

1. **What is Page Factory?**

* Page Factory is an inbuilt Page Object Model (POM) framework concept in Selenium WebDriver. It simplifies the process of managing page objects, reduces code complexity, and makes your web testing more efficient and maintainable.
* Page Factory enhances the traditional POM by providing an optimized way to initialize and manage page objects.
* It follows separation techniques for handling object repositories and test cases.

1. What are the advantages of using TestNG?

**TestNG** offers several advantages over other testing frameworks, especially when used with **Selenium**. Here are the key benefits:

1. **Parallel Execution**:
   * TestNG allows running test methods in parallel, significantly reducing overall test execution time.
   * Parallel execution is essential for efficient test suites.
2. **Flexible Test Configuration**:
   * Define test configurations (e.g., browser, environment) using the testng.xml file.
   * Easily group and prioritize test methods.
3. **Annotations and Easy-to-Understand Structure**:
   * TestNG uses annotations (e.g., @Test, @BeforeMethod, @AfterMethod) for better code organization.
   * Clear structure: @BeforeSuite, @BeforeTest, @BeforeClass, @BeforeMethod, etc.
4. **Data-Driven Testing**:
   * TestNG supports data-driven testing using @DataProvider and @Parameters.
   * Easily parameterize test cases without hardcoding data.
5. **Cross-Browser Testing**:
   * Execute the same test on multiple browsers (cross-browser testing) using TestNG.
   * Specify browser configurations in the testng.xml file.
6. **Reports and Logging**:
   * TestNG generates detailed HTML reports with information about test runs, passed/failed/skipped tests, and execution times.
   * Integrated logging for better debugging.
7. **Integration with CI/CD Tools**:
   * TestNG seamlessly integrates with tools like Jenkins, Maven, and Gradle.
   * Ideal for continuous integration and delivery pipelines.

In summary, TestNG simplifies test management, enhances reporting, and provides flexibility for efficient test execution!

1. What are the basic annotations used to run TestNG tests in Selenium?

**@BeforeSuite, @BeforeTest, @BeforeClass, @BeforeMethod, ., @Test, @BeforeMethod, @AfterMethod**

**@Test:**

**The most important TestNG annotation.**

**Represents a test method where the main business logic resides.**

**All functionalities to be automated are kept inside the @Test-annotated method.**

**@BeforeSuite:**

**Executed once before the entire suite of tests.**

**Useful for setting up global configurations or resources.**

**@AfterSuite:**

**Executed once after the entire suite of tests.**

**Useful for cleaning up resources or performing final actions.**

1. What is the difference between @beforemethod and @beforeclass?

@BeforeMethod:

Executed before each @Test method.

Ideal for setting up preconditions specific to each test method.

@BeforeClass:

Executed once before the first @Test method in a class.

Suitable for setting up class-level resources.

1. How to run test cases with dependent in Selenium using TestNG?

TestNG, you can create dependent test cases to ensure that one test runs only after another test has successfully executed. This helps manage test order and ensures proper sequencing.

1. How to run the test cases in group in Selenium using TestNG?

* You can specify test groups using the <groups> tag in your testng.xml file.
* To run specific test cases, invoke methods belonging to those groups or specify groups containing other groups.

1. What is the default priority of a test method in TestNG?

* The default priority of a test method, when not specified, is integer value 0.
* If two test cases have the same priority, TestNG executes them in alphabetical order.

1. What is the importance of testng.xml file?

* The testng.xml file is crucial for:
* Organizing tests into suites.
* Configuring test parameters.
* Defining test execution order.
* Grouping test methods.
* Specifying parallel execution settings.

1. How to pass parameters from testng.xml into test case

* Use the @Parameters annotation in your test method.
* Define parameters in the testng.xml file and reference them in your Java files.

1. What is the use of @Listener annotation in TestNG?

* The @Listeners annotation defines listeners on a test class.
* Annotated methods listen to events during test execution (e.g., success, failure, start, skip).
* Useful for customizing behavior or generating logs during test runs.

**BDD Cucumber:**

* What is BDD? How does Behavioural Driven Development work?,

BDD (Behavior-Driven Development) is an Agile software development process that emphasizes collaboration among stakeholders (customer, developer, tester). It uses conversation and concrete examples to illustrate system behavior. BDD combines techniques from TDD and DDD. Unlike TDD, which focuses on implementation, BDD emphasizes the system’s behavior. It’s a good approach for automated testing using natural language to express expected outcomes.

* What is Cucumber? and list its advantages?
* Cucumber is a tool for writing and running acceptance tests. Its advantages include:
* Human-Readable Scenarios: Scenarios are written in plain language (Gherkin syntax) that is easy to understand by both technical and non-technical stakeholders.
* Collaboration: Cucumber encourages collaboration among team members to define and illustrate system behavior.
* Business Goals: BDD with Cucumber focuses on achieving business goals and requirements.
* Avoiding Unnecessary Features: By emphasizing behavior, Cucumber helps avoid unnecessary features and includes only important ones.
* What are the main files required to run a Cucumber test scenario?
* Feature File: Contains human-readable scenarios written in Gherkin language. Describes the feature, scenarios, and feature description to be tested.
* Step Definition File: Contains the code to execute the scenarios defined in the feature file. Maps Gherkin steps to actual automation code.
* Nm,l;
* A feature file in Cucumber is an entry point for writing acceptance tests. It describes the feature, scenarios, and feature description to be tested. Feature files use Gherkin syntax and are usually named with the .feature extension. Each functionality of the software typically has a separate feature file.
* What are the various keywords that are used in Cucumber for writing a scenario?
* Feature: Provides a high-level description of a software feature and groups related scenarios.
* Scenario: Represents a single test case or scenario.
* Given, When, Then, And, But: Steps that define the behavior of the system. These steps are matched to step definitions in the code.
* Background: Defines steps that are common to all s b/cenarios in a feature.
* Scenario Outline (or Scenario Template): Used for parameterized scenarios.
* Examples (or Scenarios): Provides test data for a Scenario Outline.

1. **Purpose of a Scenario Outline in Cucumber**:
   * A **Scenario Outline** is used for parameterized scenarios.
   * It allows you to define a scenario template with placeholders for input data.
   * The same scenario can be executed with different sets of data, making testing more efficient and concise.
2. **Purpose of the Step Definition file in Cucumber**:
   * The **Step Definition file** contains the actual code to execute the scenarios defined in the feature file.
   * It maps Gherkin steps (Given, When, Then, And, But) to the corresponding automation code.
3. **Use of the Background keyword in Cucumber**:
   * The **Background** keyword defines steps that are common to all scenarios in a feature.
   * It allows you to set up a consistent context for all scenarios.
4. **Meaning of hooks in Cucumber**:
   * **Hooks** in Cucumber are special methods that run before or after scenarios.
   * They allow you to set up or tear down resources, such as database connections or browser sessions, before or after test execution.
5. **Running multiple scenarios in Cucumber**:
   * Group scenarios in a feature file or use tags to selectively execute scenarios based on criteria such as priority or category.

* What is the purpose of a Scenario Outline in Cucumber?
* What is the purpose of the Step Definition file in Cucumber?
* What is the use of the Background keyword in Cucumber?
* What is meant by hooks in Cucumber?
* How do you run multiple scenarios in Cucumber?

**GIT:**

1. What is Git?

* Git is a free and open-source distributed version control system designed to handle everything from small to very large projects with speed and efficiency. It allows developers to track changes in source code and collaborate effectively.

1. What do you understand by the term ‘Version Control System’?

A Version Control System (VCS) is software that tracks and manages changes to code over time. It helps developers record modifications, collaborate, and maintain a history of their work.

1. Name a few Git commands with their function.

Some common Git commands and their functions:

git init: Initializes a new Git repository.

git clone: Creates a local copy of a remote repository.

git add: Stages changes for commit.

git commit: Records changes to the repository.

git push: Uploads local commits to a remote repository.

1. What is the process for creating a repository in Git?

* he process for creating a Git repository:
* Use git init to initialize a new repository in your project folder.
* Add files and folders to your project.
* Commit changes using git add and git commit.

1. What is the git push command?

The git push command uploads local commits to a remote repository. It sends your changes from your local branch to the corresponding branch on the remote repository.

**Maven:**

1. What is Maven Tool?
2. Why should we use Maven?
3. What is a Maven POM file?
4. What are the repositories in Maven?
5. What is a Maven dependency?
6. What is the use of Maven clean?

1. **Maven** is a powerful project management tool based on the concept of a **Project Object Model (POM)**. It simplifies building, dependency management, and documentation for Java-based projects.
2. **Maven** streamlines project development, manages dependencies, and ensures consistent builds. It’s essential for reproducible and well-defined project structures.
3. A **Maven POM file** (pom.xml) is an XML configuration file that contains project information, dependencies, plugins, and build settings. It guides Maven in building the project.
4. **Maven repositories** are directories holding build artifacts and dependencies. Types include **local** (on your machine), **central** (publicly available), and **remote** (custom or private).
5. A **Maven dependency** is an external JAR or library required by your project. Maven automatically downloads and manages these dependencies based on your project configuration.
6. mvn clean is a Maven command that removes compiled artifacts and cleans the build directory. It ensures a fresh build by deleting any previously generated files.

**CI/CD and Jenkins:**

1. What is continuous integration and deployment?
2. What are CI Tools?
3. What is a CI CD pipeline?
4. What is Jenkins?
5. Why do we use Jenkins with selenium?
6. How do you create a Job in Jenkins?
7. How do you configure automatic builds in Jenkins?
8. **Continuous Integration and Deployment (CI/CD)**:
   * CI/CD automates software development processes.
   * **Continuous Integration (CI)**: Frequently integrating code changes into a shared repository to catch bugs early.
   * **Continuous Delivery (CD)**: Reducing friction in deployment, ensuring code can be safely released at any time.
   * **Continuous Deployment**: Automatically deploying code changes.
9. **CI Tools**:
   * Applications automating code integration, testing, and deployment.
   * Examples: **Jenkins**, TeamCity, GitLab.
10. **CI/CD Pipeline**:
    * A workflow streamlining software delivery.
    * Includes stages like integration, testing, delivery, and deployment.
11. **Jenkins**:
    * An open-source automation server for CI/CD.
    * Manages builds, tests, and deployments.
    * Widely used in DevOps.
12. **Jenkins with Selenium**:
    * Integrates Jenkins and Selenium for test automation.
    * Ensures smooth CI/CD for Selenium projects.
13. **Creating a Job in Jenkins**:
    * In Jenkins dashboard, click “New Item.”
    * Specify project name and configuration.
    * Configure source code management, build steps, and post-build actions.
14. **Configuring Automatic Builds in Jenkins**:
    * Set up triggers (e.g., SCM polling, webhook).
    * Define build steps (e.g., compile, test).
    * Configure notifications and artifacts.

Difference between single slash and double slash ?

* In **XPath**, both single slash (/) and double slash (//) have distinct purposes:
* **Single Slash (**/**)**:
* Used to create **absolute XPaths** that start selection from the **root node**.
* Selects an element directly **under the root node**.
* Example: /html selects the root HTML element.
* **Double Slash (**//**)**:
* Used to create **relative XPaths** that start selection from **anywhere within the document**.
* Selects **all descendants** of the current element that match the locator.
* Includes the **current node itself**.
* Example: //div[@id='add']//span[@id='addone'] selects all span elements that are descendants of div elements with an id attribute equal to ‘add’.

Remember:

* Single slash (/) is for absolute paths.
* Double slash (//) is for relative paths and includes descendants and self! .

Selenium Exceptions

In Selenium WebDriver, exceptions are crucial for handling unexpected situations during test execution. Let’s explore the common types of exceptions and their brief explanations:

1. **ElementNotVisibleException**: This exception occurs when an existing element in the Document Object Model (DOM) is hidden. In other words, the element exists but is not visible on the page.
2. **ElementNotSelectableException**: When an element is present in the DOM but cannot be selected (interacted with), this exception is thrown. It typically happens when an element is not in a selectable state.
3. **NoSuchElementException**: If an element cannot be found in the DOM, this exception occurs. It’s essential to handle this when locating elements dynamically.
4. **InterruptedException: when we use thread.sleep();**
5. **NoSuchFrameException**: When attempting to switch to a frame that doesn’t exist, this exception is raised. Make sure the target frame is valid.
6. **NoAlertPresentException**: If there’s no alert currently presented, this exception occurs when switching to an alert.
7. **NoSuchWindowException**: When trying to switch to a non-existent window, this exception is thrown. Ensure the window handle is correct.
8. **StaleElementReferenceException**: This exception occurs when a web element becomes detached from the current DOM. It often happens due to page refresh or dynamic content updates.
9. **SessionNotFoundException**: If you try to interact with the WebDriver after quitting the browser, this exception is raised.
10. **TimeoutException**: When a command takes too long to complete (e.g., element search), this exception is thrown.
11. **WebDriverException**: It occurs when the WebDriver is used after closing the browser.
12. **ConnectionClosedException**: This type of exception arises when there’s a disconnection in the driver.
13. **ElementClickInterceptedException**: When an element conceals the requested clickable element, this exception occurs.
14. **ElementNotInteractableException**: If an element is present in the DOM but cannot be interacted with (e.g., input fields), this exception is thrown.
15. **ErrorInResponseException**: Interacting with Firefox extensions or the remote driver server can trigger this exception.
16. **ErrorHandler.UnknownServerException**: Used as a placeholder when the server returns an error without a stack trace.
17. **ImeActivationFailedException**: Occurs when IME engine activation fails.
18. **ImeNotAvailableException**: Indicates that IME support is unavailable.
19. **InsecureCertificateException**: When navigating to a site with an invalid or expired TLS certificate, this exception may occur.
20. **InvalidArgumentException**: Raised when an argument does not match the expected type.

Launch the browsers

System.setProperty("webdriver.chrome.driver","C:\\Users\\2318407\\Downloads\\chromedriver\_win32\\chromedriver.exe");

WebDriverManager.chromedriver().setup();

Steps to connect database

* + - 1. Load the driver : Class.forName(“com.mysql.jdbc.Driver());
      2. Or we can write : DriverManager.registerDriver(new com.mysql.jdbc.Driver());
      3. Create a connection
      4. Connection con = DriverManager.getConnection(“url”, “username”, “password”);
      5. Create a query :
      6. Statement stm = con.createStatement();
      7. Stmt.executeQuery(query);
      8. Resultset set = stmt.executeQuery(query);
      9. Process the data : while (set.next()){

}

How to take screenshot ?

Create object of TakesScreenshot interface

TakesScreenshot scr =((TakesScreenshot)driver)

File source = scr.getScreenshotAs(OutputType.FILE);

FileUtils.copyFile(source, new File(“c//img//test.png”);

Mouse event

Actions act = new Actions(driver) ; // passing driver instance to Actions

Howering on any element

act.moveToElement(element1).movoToElement(element2).click().perform();

Right click

act.contextClick(element).perform();

double click

act.doubleClick().perform();

* Actions class vs Action interface
* Action: it is an interface
* Represents a **single user-interaction action**.
* Defined in org.openqa.selenium.interactions.
* Contains the widely used method perform().
* java Action action = new SomeAction(); action.perform();
* Actions: it is a class
* Provides a user-facing API for emulating complex user gestures.
* Also defined in org.openqa.selenium.interactions.
* Implements the builder pattern and can build a CompositeAction containing all specified actions.
* java Actions actions = new Actions(driver); actions.someAction().anotherAction().build().perform();

Differences between the Alpha testing and Beta testing are:

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **Alpha Testing** | **Beta Testing** |
| **1.** | Alpha testing performed by a team of highly skilled testers who are usually the internal employee of the organization. | Beta testing performed by clients or end-users in a real-time environment, who is not an employee of the organization. |
| **2.** | Alpha testing performed at the developer's site; it always needs a testing environment or lab environment. | Beta testing doesn't need any lab environment or the testing environment; it is performed at a client's location or end-user of the product. |
| **3.** | Reliability or security testing not performed in-depth in alpha testing. | Reliability, security, and robustness checked during beta testing. |
| **4.** | Alpha testing involves both white box and black-box techniques. | Beta testing uses only black-box testing. |
| **5.** | Long execution cycles maybe require for alpha testing. | Only a few weeks are required for the execution of beta testing. |
| **6.** | Critical issues or fixes can be identified by developers immediately in alpha testing. | Most of the issues or feedback is collecting from the beta testing will be implemented for the future versions of the product. |
| **7.** | Alpha testing performed before the launch of the product into the market. | At the time of software product marketing. |
| **8.** | Alpha testing focuses on the product's quality before going to beta testing. | Beta testing concentrates on the quality of the product, but gathers users input on the product and ensures that the product is ready for real-time users. |
| **9.** | Alpha testing performed nearly the end of the software development. | Beta testing is a final test before shipping a product to the customers. |
| **10.** | Alpha testing is conducting in the presence of developers and the absence of end-users. | Beta testing reversed of alpha testing. |

**Common Annotations in Selenium**:

* **@BeforeMethod**: Runs before each test method. Useful for setup (e.g., opening a browser).
* **@AfterMethod**: Runs after each test method. Useful for cleanup (e.g., closing the browser).
* **@BeforeTest**: Runs before all test methods in a test suite.
* **@AfterTest**: Runs after all test methods in a test suite.

Import for taking screenshot

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.**TakesScreenshot;**

import org.openqa.selenium.OutputType;

import org.openqa.selenium.firefox.FirefoxDriver;

import java.io.File;

import org.apache.commons.io.FileUtils;

where org.opemqa.selenium : it is package

TakesScreenshot : it is interface

import for excel file

**import** org.apache.poi.xssf.usermodel.XSSFRow;

**import** org.apache.poi.xssf.usermodel.XSSFSheet;

**import** org.apache.poi.xssf.usermodel.XSSFWorkbook;

**import** java.io.File;

**import** java.io.FileInputStream;

**import** java.io.FileOutputStream;

**import** java.io.IOException;

import for using Actions class and Action interface

import org.openqa.selenium.interactions.Actions;

import for using alert interface

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.Alert;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.By;

import for using select class in selenium

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

import for using WebElement in selenium

import org.openqa.selenium.WebElement;

import for using javascript executor

import org.openqa.selenium.JavascriptExecutor;

Cucumber

Cucumber is a software tool used by the testers to develop test cases for the testing of behavior of the software.

It follows a **BDD** (Behavior Driven Development) framework to observe the behavior of the software's functionalities.

BDD (Behavioral Driven Development) is a software development approach that was developed from **Test Driven Development (TDD)**.

BDD includes test case development in the form of simple English statements inside a [feature file](https://www.javatpoint.com/feature-file-in-cucumber-testing), which is human-generated. Test case statements are based on the system's behavior and more user-focused.

BDD is written in simple English language statements rather than a typical programming language, which improves the communication between technical and non-technical teams and stakeholders.

* + **Test Case**: Specific steps to verify a feature.
  + **Test Scenario**: High-level functionality to be tested.
* **Derived From**:
  + **Test Case**: Derived from test scenarios.
  + **Test Scenario**: Derived from BRS or SRS.
* **Level of Detail**:
  + **Test Case**: Detailed, low-level actions.
  + **Test Scenario**: High-level, end-to-end functionality.
* **Resources and Time**:
  + **Test Case**: Requires more resources and time.
  + **Test Scenario**: Requires fewer resources and time.
* **Scroll to the Bottom of the Page**:
  + To scroll to the end of the page, you can use either of the following methods:
  + Js.executeScript(“window.scrollBy(0,document.body.scrollHeight);”);

JavascriptExecutor js = (JavascriptExecutor) driver;

js.executeScript("window.scrollTo(0, document.body.scrollHeight);");

* + - Using Java Robot class (simulating pressing Ctrl+End):

**Java**

driver.findElement(By.cssSelector("body")).sendKeys(Keys.CONTROL, Keys.END);

AI-generated code. Review and use carefully. [More info on FAQ](https://www.bing.com/new#faq).

* **Scroll to an Element’s Visibility**:
  + To scroll until an element becomes visible, you can use the scrollIntoView() method:

**Java**

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

((JavascriptExecutor) js.executeScript("arguments[0].scrollIntoView(true);", element);

Js.executeScript (“arguments[0].scrollIntoView();”, element);

To scoll down ;

Js.executeScript(“window.scrollBy(0,300)

To capture location

Js.executeScript(return window.pageYoffset;))

1. Action**Interface**:
   * **Definition**: The Action interface represents a single user-interaction action. It is defined in the package org.openqa.selenium.interactions.
   * **Purpose**:
     + Used for defining individual user interactions (e.g., click, double-click, drag-and-drop).
     + Represents a single step in a sequence of actions.
     + Typically used with the perform() method to execute the action.
   * **Example**:

**Java**

// Creating an Action to perform a click on an element

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

Action clickAction = new Actions(driver).click(element).build();

clickAction.perform();

1. Actions**Class**:
   * **Definition**: The Actions class is a user-facing API for emulating complex user gestures. It is also defined in the package org.openqa.selenium.interactions.
   * **Purpose**:
     + Provides a higher-level API for combining multiple actions into a sequence (e.g., mouse hover, drag-and-drop, key press).
     + Implements the builder pattern, allowing you to chain multiple actions together.
     + Useful for creating composite actions involving multiple steps.
   * **Example**:

**Java**

// Creating a composite action (mouse hover and click)

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

Actions actions = new Actions(driver);

actions.moveToElement(element).click().build().perform();

1. **Summary**:
   * **Action Interface**:
     + Represents a single user interaction.
     + Used for defining individual actions.
     + Often combined with perform() to execute the action.
   * **Actions Class**:
     + User-facing API for complex gestures.
     + Combines multiple actions into a sequence.
     + Implements the builder pattern.

Remember, use the Action interface for single actions and the Actions class for more complex interactions!

1. Why we need TestNG with selenium ?

TestNG is a powerful testing framework for Java that enhances automated testing for web applications. When combined with Selenium, it offers several advantages that improve the efficiency and effectiveness of your test automation efforts.

* For managing and maintaining test suite easier
* For structured test organization
* For parallel execution
* Annotations for customization of tests
* For data driven testing (same test case with multiple data set)
* For enhanced reporting and data driven testing
* For executing tests cases in specific order

1. Different types of annotations available in TestNG ?

**@BeforeSuite**:

* + **Use**: Runs before all test methods in the suite.
  + **Purpose**: Set up global configurations or resources.

**@AfterSuite**:

* + **Use**: Executes after all test methods in the suite.
  + **Purpose**: Perform cleanup tasks or finalize resources.

**@Test**

* **Use**: The primary annotation where the main business logic resides.
* **Purpose**: Define individual test methods.

**@BeforeTest**:

* + **Use**: Runs before all test methods within a test tag.
  + **Purpose**: Initialize common test data or resources.

**@AfterTest**:

* + **Use**: Executes after all test methods within a test tag.
  + **Purpose**: Clean up resources or perform post-test actions.

**@BeforeClass**:

* + **Use**: Invoked before the first method of the current class.
  + **Purpose**: Set up class-level configurations.

**@AfterClass**:

* + **Use**: Called after all test methods in the current class.
  + **Purpose**: Perform teardown tasks specific to the class.

**@BeforeMethod**:

* + **Use**: Runs before each test method.
  + **Purpose**: Prepare test-specific data or resources.

**@AfterMethod**:

* + **Use**: Executes after each test method.
  + **Purpose**: Clean up after test execution.

**@BeforeGroups**:

* + **Use**: Runs once for a group before executing group-related tests.
  + **Purpose**: Set up shared resources for group tests.

**@AfterGroups**:

* + **Use**: Executes once for a group after group-related tests.
  + **Purpose**: Perform cleanup or finalization for group tests.

1. **What is difference between Suite, Test, and Class?**
   * Suite: A suite is made of one or more tests.
   * Test: A test is made of one or more classes.
   * Class: A class is made of one or more methods.
2. **What is the test method?**

A method annotated with @Test is called test method which serves as a unit test. In the @Test method, we will write the logic of the application which we want to automate.

1. **What is the return type of @DataProvider annotation provided by TestNG?**

@DataProvider annotation return an object double array (Object [ ][ ]) as data to the test method.

**What is the return type of @Factory annotation?**

This annotation returns an array of class objects (Object [ ]).

**What is the default value for TestNG Priority?**

Ans: The default value for TestNG priority is zero.

If you have not mentioned test priority, TestNG will assign all @Test a priority as zero(0) and execute them.

**How to group multiple test methods in multiple groups?**

Ans: The grouping of test methods belonging to multiple groups can be done by providing the group names as an array in the groups attribute of the @Test annotation.

The below syntax lets you add test methods in multiple groups.  
**Syntax:**

@Test(groups = { "GroupName1", "GroupName2", "GroupName3" .... })

**9. How to group multiple test methods with Priority?**

Ans: You can also group multiple test methods with priority. The syntax for grouping test methods with priority is as follows:  
**Syntax:**

@Test(groups = {"GroupName"}, priority=0) // The test method annotated with this group will execute first.

@Test(groups = {"GroupName"}, priority=1) // The test method annotated with this group will execute after executing the first group.

**10. What is Inclusion & Exclusion Groups in TestNG?**

Ans: A group that is included in test execution is called inclusion group. A group which is excluded from test execution is called exclusion group.

**11. How to disable a test in TestNG?**

Ans: We can also disable tests on an individual basis by using the “enabled” property available on both @Test and @Before/After annotations. The syntax can be like this:  
**Syntax:**

@Test(enabled = false)

@Test(groups = {"Cricket Player"}, enabled=false)

**12. What is Default Group, Partial Groups, and MetaGroups in TestNG?**

Ans: **Default Group:** When an entire class is added to a group, it is called default group. It is a good way of defining a default group for all unit tests within a class.

**Partial Groups:** When you define groups at the class level and then add groups at the method level, it is called partial groups.

**MetaGroups:** When groups include other groups, these groups are called metagroups.

**How will you skip a particular test case in TestNG?**

@Test(enabled = false)

**. What is invocationCount in TestNG?**

Ans: invocationCount in TestNG is an attribute that is used to execute a method or test in the number of times. It acts as a loop.  
**For example:**

@Test(invocationCount = 5) Hence, this method will execute 5 times.

**9. What is timeOut in TestNG?**

Ans: timeOut in TestNG is an attribute that is used for a time out test. It specifies a time period (in milliseconds) to wait for a test for complete execution.

**What is testng.xml file?**

Ans: Testng.xml file is a configuration file (XML file) for TestNG in which we can create test suites, test groups, mark tests for parallel execution, add listeners, and pass parameters to test script. It defines the runtime definition of a test suit.

**2. What is the importance or use of testng.xml file?**

Ans: There are following importance or usage of the testng.xml file in TestNG. They are as follows:

1. The testng.xml file can be used to control the execution of whole tests from a single file.  
2. We can run a set of test cases from a single place.  
3. We can pass parameters to test methods or classes.  
4. Using testng.xml file, we can perform parallel test execution.  
5. We can add the listener.

6. It defines the order of the execution of all the test cases.  
7. It allows us to group test cases and can be executed as per requirements.  
8. It helps to execute selected test cases.  
9. With the help of testng.xml file, we can integrate TestNG framework with Jenkins.

**3. How to pass parameters in test cases through testng.xml file?**

Ans: We can pass the parameter value to test methods at runtime through the testng.xml file. For this, we can use @Parameter annotation:

SwitchTo() command use in selenium ?

driver.switchTo().newWindow(WindowType.TAB);

driver.switchTo().window();

driver.switchTo().frame(0);

driver.switchTo().defaultContent();(Switch back to the default content (outside iframes)

driver.switchTo().alert();

driver.switchTo().parentFrame() (to move up one level in the frame hierarchy.)