1. **dComponents of Selenium**:
   * Selenium comprises four major components:
     + **Selenium IDE (Integrated Development Environment)**: A complete IDE for Selenium tests. It’s implemented as a Firefox Add-On and Chrome Extension. You can record, edit, and debug functional tests using Selenium IDE. It was previously known as Selenium Recorder.
     + **Selenium RC (Remote Control)**: A server (written in Java) that accepts commands via HTTP. It allows writing automated tests for web applications in various programming languages. Selenium provides client drivers for languages like PHP, Python, Ruby, .NET, Perl, and Java.
     + **Selenium WebDriver**: The successor to Selenium RC. WebDriver directly controls browsers by sending commands to browser-specific drivers. It doesn’t require a special server. You can use it with various browsers (e.g., Chrome, Firefox, Safari).
     + **Selenium Grid**: Used with WebDriver to execute tests on remote systems.
2. **Selenium IDE**:
   * An essential tool in the Selenium Suite.
   * Allows recording, editing, and debugging of functional tests.
   * Implemented as a Firefox Add-On and Chrome Extension.
   * Previously known as Selenium Recorder.
   * Scripts are written in **Selenese**, a special test scripting language for Selenium.
3. **Selenium WebDriver**:
   * The heart of Selenium automation.
   * Accepts commands (in Selenese or via a Client API) and sends them to a browser.
   * Uses browser-specific drivers (e.g., ChromeDriver, GeckoDriver) to control browsers.
   * No need for a special server; WebDriver directly starts a browser instance.
   * Example code snippet to set up WebDriver in Java:
4. **Element Locators in Selenium**:
   * Common locators include:
     + **ID**: Find elements by their unique ID attribute.
     + **Name**: Locate elements by their name attribute.
     + **XPath**: Powerful but slower; uses path expressions to navigate XML documents.
     + **CSS Selector**: Similar to XPath but more concise.
     + **Class Name**: Locate elements by their class attribute.
     + **Link Text**: Find links by their visible text.
     + **Partial Link Text**: Locate links by partial visible text.
5. **Maximizing Window with Selenium WebDriver (Java)**:

**Java**

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

1. **Clearing Content of a Text Box**:

**Java**

WebElement textBox = driver.findElement(By.id("textbox\_id"));

textBox.clear();

1. **Executing JavaScript in Selenium WebDriver (Java)**:

**Java**

// Execute a JavaScript function

JavascriptExecutor jsExecutor = (JavascriptExecutor) driver;jsExecutor.executeScript("alert('Hello, Selenium!')");

1. **Declare the drop-down element as an instance of the**Select**class**:

Select dropdown = new Select(driver.findElement(By.id("your\_dropdown\_id")));

* + **Select the desired option** using one of the following methods:
    - selectByVisibleText("Option Text"): Selects an option by its visible text.
    - selectByValue("Option Value"): Selects an option by its value attribute.
    - selectByIndex(index): Selects an option by its index (0-based).

1. **Explain about the**Select**class and its methods:** The Select class in Selenium provides methods to interact with dropdowns. Here are some commonly used methods:
   * selectByVisibleText(String arg0): Selects an option by its visible text.
   * selectByIndex(int arg0): Selects an option by its index.
   * selectByValue(String arg0): Selects an option by its value attribute.
   * getOptions(): Returns a list of all options within the dropdown.
   * deselectAll(): Clears all selected entries (applicable for multi-select dropdowns).
2. **How to automate radio buttons in Selenium WebDriver?**

To automate radio buttons, locate the radio button element and use the click() method to select it.

1. **How to submit a form using Selenium WebDriver?** Locate the form element and use the submit() method to submit the form.
2. **Difference between**findElement**and**findElements**methods.**

* findElement: Locates a single web element based on a unique locator strategy. Returns a WebElement. Returns the first matching element or throws an exception if not found.
* findElements : Locates multiple web elements based on the same criteria. Returns a list of WebElement objects. Returns a list of all matching elements (empty list if none found).

1. **How to count the total number of links on a page using Selenium WebDriver?**

Use the findElements(By.tagName("a")) to get all anchor tags (links) and then count the size of the list.

1. **How to capture the page title using Selenium WebDriver?**

Use driver.getTitle() to retrieve the page title.

1. **How to store the current page URL using Selenium WebDriver?**

Use driver.getCurrentUrl() to get the current URL.

1. **How to simulate browser back and forward navigation?**
   * To go back: driver.navigate().back()
   * To go forward: driver.navigate().forward();
2. **Difference between single and double slash in XPath:**
   * Single slash (/): Selects the immediate child element.
   * Double slash (//): Selects any descendant element (not just immediate children).
3. **How to handle dynamic XPath?**
   * Use relative XPath with attributes that remain constant.
   * Use functions like contains(), starts-with(), or text() to handle dynamic values.
4. **Choosing the Right Locator**: When selecting a locator, consider the following options:
   * **ID**: Use when the element has a unique ID attribute.
   * **Name**: Suitable for elements with a unique name attribute.
   * **XPath**: Powerful and flexible, but can be slower. Useful for complex scenarios.
   * **CSS Selector**: Efficient and concise. Good for styling-based locators.
5. **Asserting Text on a Webpage**: To assert text on a webpage, use assertions from testing frameworks (e.g., TestNG, JUnit). For example:

String expectedText = "Welcome to Selenium!";

String actualText= driver.findElement(By.id("welcomeMessage")).getText();

Assert.assertEquals(actualText, expectedText);

1. **Getting Element Attributes**:

You can retrieve an attribute value using getAttribute():

String hrefValue = driver.findElement(By.linkText("Learn More")).getAttribute("href");

1. **Double Clicking an Element**:

Use the Actions class to perform double-click actions:

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

Actions actions = new Actions(driver);

actions.doubleClick(element).perform();

1. **Drag and Drop**: Drag an element to a target location using dragAndDrop():

WebElement source = driver.findElement(By.id("sourceElement"));

WebElement target = driver.findElement(By.id("targetElement"));

Actions actions = new Actions(driver);

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

1. **Synchronizing Application Window and Selenium**:
   1. Use implicit waits (driver.manage().timeouts().implicitlyWait()) to wait for elements to appear.
   2. Use explicit waits (WebDriverWait) for specific conditions (e.g., element visibility, presence).

26. **Implicit vs. Explicit Wait**:

* + **Implicit Wait**: Set once for the entire session. Waits for a specified time before throwing an exception if an element is not found.
  + **Explicit Wait**: Applied to specific elements. Waits for a certain condition to be met (e.g., element visibility) within a specified timeout.

1. **Handling Multiple Windows**:
   1. Use getWindowHandles() to get handles of all open windows.
   2. Switch between windows using driver.switchTo().window(windowHandle).
2. **Working with Pop-Up Windows**:
   1. Identify pop-up elements using locators.
   2. Switch to the pop-up window using driver.switchTo().window(windowHandle).
3. **Mouse Hover on an Element**:

Use Actions class for mouse interactions

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

Actions actions = new Actions(driver);

actions.moveToElement(element).perform();

1. **Capturing Screenshots**:

To take a screenshot:

File screenshotFile = ((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);

FileUtils.copyFile(screenshotFile, new File("path/to/save/screenshot.png"));

1. **Getting Data from an Excel Sheet**:

To read data from an Excel sheet, you can use libraries like Apache POI. Here’s a basic example:

// Add Apache POI dependencies to your project

FileInputStream file = new FileInputStream("path/to/your/excel.xlsx");

XSSFWorkbook workbook = new XSSFWorkbook(file);

XSSFSheet sheet = workbook.getSheet("Sheet1");

// Read data from cells

String value = sheet.getRow(0).getCell(0).getStringCellValue();

1. **Reading Data from a Properties File**:

Properties files store key-value pairs. You can read them using Java’s Properties class:

Properties properties = new Properties();

FileInputStream file = new FileInputStream("path/to/your/config.properties");

properties.load(file);

String username = properties.getProperty("username");

String password = properties.getProperty("password");

1. **Types of Automation Frameworks**:
   1. **Linear/Scripting Framework**: Simplest, where tests are written sequentially.
   2. **Modular Framework**: Divides tests into reusable modules.
   3. **Data-Driven Framework**: Separates test data from test logic.
   4. **Keyword-Driven Framework**: Uses keywords to represent test actions.
   5. **Hybrid Framework**: Combines elements of multiple frameworks.
2. **Data-Driven Framework & Keyword Driven**:
   1. **Data-Driven Framework**: Tests are driven by external data (e.g., Excel, CSV). Reusable test scripts handle different data sets.
   2. **Keyword-Driven Framework**: Tests are written using keywords (actions). Keywords map to specific test steps.
3. **Selenium Grid**:
   1. Allows parallel execution of tests across multiple machines and browsers.
   2. Consists of a hub (central control) and nodes (machines with browsers).
   3. Distributes test execution based on requested capabilities.
4. **Default Port ID in Selenium Grid**:

The default port for the Selenium Grid hub is **4444**.

1. **Page Object Model (POM)**:
   1. Design pattern to create an object repository for web elements.
   2. Advantages:
      1. **Modularity**: Separates test logic from page structure.
      2. **Reusability**: Page objects can be reused across tests.
      3. **Maintainability**: Changes to UI are localized to page objects.
2. **Page Factory**:
   1. An extension of POM.
   2. Uses annotations (@FindBy) to locate elements.
   3. Initializes elements lazily (when accessed) for better performance.