**Adv. selenium - Selenium IDE and scripts**

1. **Selenium RC**

* Selenium RC (Remote Control) is one of the older components of the Selenium testing framework. It allows testers to write automated web application tests in various programming language (like Java, C#, Python, PHP, Perl, Ruby) and execute them in real browsers.
* Selenium RC Architecture has two key components: *Selenium Server* and *Client Libraries*.
* How it Works:

Selenium RC Server acts as a middleman (or proxy) between your test script and the browser.

It injects JavaScript code (called Selenium Core) into the browser to control DOM elements and simulate user actions.

The RC server interprets commands from your test script and passes them to the browser.

* Features:

Supports multiple programming languages.

Can run tests in real browsers.

Allows for complex testing scenarios.

* RC supported testing across different browsers like Firefox, Chrome, and Internet Explorer.
* Limitations:
* Slower than newer alternatives (like Selenium WebDriver).
* Needs a separate server to be started before running tests.
* Selenium RC is now deprecated and replaced by Selenium WebDriver, which is more powerful, doesn't require a server, and interacts directly with the browser using native APIs.

1. **IDE**

* Selenium IDE (Integrated Development Environment) is a browser-based tool for creating, recording, and debugging automated test scripts for web applications. It is part of the Selenium suite, which also includes Selenium WebDriver and Selenium Grid.
* It is the simplest tool in the Selenium suite, designed primarily for beginners and manual testers who want to create quick automated tests without writing code.
* Selenium IDE (Integrated Development Environment) is primarily a record/run tool that a test case developer uses to develop Selenium Test cases.
* When to Use Selenium IDE:

For quick test prototyping or exploratory testing.

When testers don’t know programming.

* Limitations:

Not suitable for complex test logic or large test suites.

Limited support for advanced data-driven testing.

Slower and less flexible than WebDriver-based frameworks.

1. **Selenium Web Driver/Selenium 2.0**

* Selenium WebDriver is a powerful Automation tool widely used for web application testing.
* It provides a programming interface to interact with web browsers, allowing users to automate browser actions, navigate web pages, and perform functional testing.
* With support for multiple programming languages such as Python, Java, and JavaScript, Selenium WebDriver facilitates cross-browser and cross-platform testing, making it an essential tool for software developers and quality assurance professionals.
* Selenium WebDriver stands as a pivotal tool in web automation, offering a robust framework with support for multiple programming languages and cross-browser compatibility.
* Its architecture, driven by client libraries, JSON Wire Protocol, browser drivers, and real browsers, enables seamless automation.
* The advantages of WebDriver, including cost-effectiveness, language flexibility, and efficient handling of dynamic elements, make it an indispensable choice for developers and QA professionals in ensuring reliable and consistent web application testing.

1. **Selenium-Grid**

* Selenium Grid is a tool that allows you to run Selenium tests in parallel across multiple machines, browsers, and operating systems.
* It’s a key part of scalable test automation - especially for large test suites or cross-browser testing.
* It acts as a smart proxy server, routing commands from a client to remote browser instances (nodes) managed by a central server (hub). This allows for tests to be run against different browser versions, operating systems, and even different machines simultaneously.
* Use Selenium Grid when –
* You have large test suits.
* You need to test across browsers/OSes.
* You want to save time via parallel execution.
* You’re running tests on a on a CD/CI Server(like Jenkins).

1. **Why Web Driver?**

* WebDriver is essential for automated browser testing and real user simulation. It gives you full control over how a user would interact with your web application in a real environment, across multiple browsers and platforms.
* Advantages of selenium Webdriver
* Cross-Browser Compatibility.
* Multi-language Support.
* Cost-Effective.
* No Need for Remote Server.
* Supports Multiple Operating Systems.

1. **Architecture of selenium web driver**

* Selenium WebDriver is a robust framework for automating web browsers, and its architecture is built around a client-server model.
* It consists of several key components: Selenium Client Libraries, the JSON Wire Protocol, Browser Drivers, and the actual Web Browsers.

1) Selenium Client Libraries:

* They allow users to interact with WebDriver using various programming languages like Java, Python, Ruby, Java and C#.
* They provide an API that allows users to control the browser, locate elements, perform actions, and verify expected outcomes.
* The Selenium Client Libraries allowing users to write test scripts in their preferred language.

2) JSON Wire Protocol:

* JSON is an acronym for JavaScript Object Notation.
* This protocol acts as the communication bridge between the client library and the browser driver.
* It uses JSON (JavaScript Object Notation) to format data and HTTP requests to transmit commands and data between the client and the browser driver.
* It ensures a standardized communication channel for reliable and consistent execution of test scripts.

3) Browser Drivers:

* These drivers are specific to each web browser and are responsible for interacting with the browser.
* They act as a bridge between the Selenium client and the browser, translating commands and data into actions the browser can understand.
* Selenium browser drivers are native to each browser, interacting with the browser by establishing a secure connection. Selenium supports different browser drivers such as ChromeDriver, GeckoDriver, Microsoft Edge WebDriver, SafariDriver, and InternetExplorerDriver.

4) Web Browsers:

* These are the target browsers where the automation tests are executed.
* Selenium provides support for multiple browsers like Chrome, Firefox, Safari, Internet Explorer , and others.
* The browser drivers interact with the browser to execute commands and perform actions as specified in the test scripts.
* In essence, the Selenium WebDriver architecture enables users to:
* Write test scripts in their preferred language using the Selenium Client Libraries.
* Communicate with the browser using the JSON Wire Protocol.
* Control the browser through the browser drivers.
* Execute tests on various browsers and platforms.

1. **Drivers for Firefox, IE, Chrome, IPhone, Android etc**

| * Platform | * Driver Name | * Tool Required | * Notes |
| --- | --- | --- | --- |
| * Firefox | * geckodriver | * Selenium | * Official Firefox driver, supports modern standards |
| * IE | * IEDriverServer | * Selenium | * Legacy support, Windows-only |
| * Chrome | * chromedriver | * Selenium | * Most commonly used driver |
| * iPhone (iOS) | * XCUITest via Appium | * Appium | * Needs macOS, Xcode, automates Safari/iOS apps |
| * Android | * UiAutomator2 via Appium | * Appium | * Automates Chrome or Android native apps |

1. **What is the difference between Selenium RC and WD**

* Selenium Remote Control (RC) and Selenium WebDriver (WD) are both automation frameworks for testing web applications, but they differ significantly in their architecture and approach to browser interaction. Selenium RC uses a server to communicate with the browser, making it slower and more complex, while Selenium WebDriver directly interacts with the browser, leading to faster and more reliable automation.
* **Selenium RC:**
* Architecture:

RC uses a separate Selenium Remote Control Server that acts as an intermediary between your test code and the browser.

* Communication:

It communicates with the browser through JavaScript, which can be slower and less reliable.

* Complexity:

The use of a server adds complexity to the setup and execution of test scripts.

* Limitations:

RC is considered deprecated and is less suitable for modern web applications and advanced testing scenarios.

* **Selenium WebDriver:**
* Architecture:
* WebDriver interacts directly with the browser without the need for a server.
* Communication:
* It uses native browser APIs and drivers, leading to faster and more reliable execution.
* Simplicity:
* WebDriver's architecture is simpler, making it easier to set up and use.
* Features:
* WebDriver offers a more extensive API, better support for advanced testing scenarios, and native language bindings for popular programming lang

1. **IE Developer tools**

* Internet Explorer Developer Tools (IE DevTools) are built-in tools that allow developers to inspect, debug, and analyze web pages in Internet Explorer (primarily versions 8 through 11).
* When to use IE Developer tools :
* Only when – Maintaining Legacy enterprise applications.
* Debugging apps still required to run in Internet Explorer 11.
* Checking compability or layout issues specific to order IE versions.

1. **Google Chrome Developer tools**

* Chrome DevTools are built-in developer tools provided by the Google Chrome browser. They allow developers to inspect, debug, and optimizeweb pages and applications directly in the browser.
* Use Device Toolbar (📱 icon or Ctrl+Shift+M) to test responsive layouts.
* Why use Chrome DevTools ?
* It’s one of the most powerful web debugging tools available.
* Used by nearly all front-end developers.
* Continuously updated with new features by the Chromium team.

1. **Finding Elements by using className, Name, Link Text, Partial link Text, XPath, using CSS, Using TagName**

* Ways to Locate Elements in Selenium/WebDriver
* **1. By Class Name -** driver.find\_element(By.CLASS\_NAME, "my-class")
* **2. By Name -** driver.find\_element(By.NAME, "username")
* **3. By Link Text -** driver.find\_element(By.LINK\_TEXT, "Click Here")
* **4. By Partial Link Text -** driver.find\_element(By.PARTIAL\_LINK\_TEXT, "Click")
* **5. By XPath -** driver.find\_element(By.XPATH, "//input[@id='email']")
* **6. By CSS Selector -** driver.find\_element(By.CSS\_SELECTOR, "input#email")
* **7. By Tag Name -** driver.find\_element(By.TAG\_NAME, "button")

1. **Some Special IDE command**

* **Flow Control Commands :**

| * **Command** | * **Description** |
| --- | --- |
| * if | * Start of a conditional block. |
| * else if | * Optional condition within an if block. |
| * else | * Executes if the if/else if condition fails. |
| * end | * Marks the end of if/loop blocks. |
| * while | * Starts a loop that runs while the condition is true. |
| * do | * Starts a do-while loop. |
| * repeat if | * Repeats a block if the condition is true. |
| * for each | * Iterates over items (e.g., in a list or table). |

* **Wait and Pause Commands**

| * **Command** | * **Description** |
| --- | --- |
| * pause | * Pauses execution for a set time (in milliseconds). |
| * waitForElementPresent | * Waits until an element appears in the DOM. |
| * waitForElementVisible | * Waits until the element is both present and visible. |
| * waitForText | * Waits until specific text is present. |

* **Assert and Verify Commands**

| * **Command** | * **Description** |
| --- | --- |
| * assertText | * Fails the test if the text doesn’t match. |
| * verifyText | * Logs a warning if the text doesn’t match (but continues). |
| * assertElementPresent | * Checks if an element exists. |
| * assertTitle | * Verifies the page title. |
| * storeText | * Stores the text of an element into a variable. |

* **Control Execution**

| * **Command** | * **Description** |
| --- | --- |
| * gotoIf | * Jumps to a label if condition is true. |
| * gotoLabel | * Defines a label to jump to. |
| * break | * Exits a loop early. |
| * continue | * Skips to the next iteration in a loop. |
| * echo | * Prints a message to the log (useful for debugging). |

* **Variables and Data**

| * **Command** | * **Description** |
| --- | --- |
| * store | * Stores a literal value. |
| * storeEval | * Stores result of a JavaScript expression. |
| * storeText | * Stores text from an element. |
| * storeValue | * Stores the value of a form element. |

1. **Web Tables**

* A web table is an HTML table displayed on a web page, often used to present structured data like product lists, search results, user records, etc. In test automation using Selenium, interacting with these tables is a common task - for reading data, validating values, or clicking buttons inside rows.
* It is used to display data in structured formate.
* Tabular format : Row , Columns
* HTML Table consists of several tags that define its structure:
* <table> - The main container for the table
* <thread> - Defines the tableheader, which contains header rows (<tr>)
* <tbody> - Contains the table body, which includes the data rows.
* <tr> - Defines a table row.
* <th> - Defines a header cell in a table row.
* <td> - Defines a standard data cell in a table row.

1. **Extracting data from web tables**

* Extracting data from web tables involves retrieving data from a webpage that is presented in a tabular format.
* This can be done using various tools and methods, including web scraping, programming languages like Python, and browser extensions. The process generally involves identifying the table, accessing the HTML, and then extracting the desired data.

1. **Examples on static and dynamic Web Tables**

* Static web tables have fixed content and structure, while dynamic web tables have data that can change over time, including the number of rows and columns.
* Examples of static tables include a table of days in a month or a student list with fixed enrollments, while dynamic tables could be a sales report or detailed product information on an e-commerce website.
* Static Web Tables: A static web table is a table whose content is hard-coded in the HTML. It does not change unless the page is manually edited or refreshed.
* Table of days in a month: The number of days and the order of days remain the same.
* A fixed list of students: The list remains constant, with no changes in student enrollment.
* A product catalog with fixed information: The product names, descriptions, and prices remain unchanged.
* Dynamic Web Tables: A dynamic web table loads its content at runtime, often using AJAX, JavaScript, or APIs. The data may be refreshed without reloading the page.
* A sales report: The data in the table changes as sales transactions occur.
* A list of customers: The list updates as new customers are added or removed.

1. **Junit Framework**

* JUnit is a unit testing framework for Java that can be used with Selenium to automate web application testing. It helps to structure, execute, and verify tests, making it a valuable tool for developers and testers.
* JUnit is a unit testing framework for Java programming. It is primarily used to write and run automated tests for Java code.
* JUnit helps developers and testers validate that individual units (methods or classes) of the code behave as expected**.**

1. **JUNIT Annotations / JUNIT Methods**

* JUnit annotations are used to define methods that serve specific roles in the testing lifecycle. These annotations help structure tests and ensure that tasks are performed before and after test cases.

| * **Annotation** | * **Purpose** |
| --- | --- |
| * @Test | * Marks a method as a test case |
| * @Before | * Runs before each test method |
| * @After | * Runs after each test method |
| * @BeforeClass | * Runs once before any test methods in the class |
| * @AfterClass | * Runs once after all test methods are executed |
| * @Ignore | * Skips a test method |

1. **Test Suites using JUNIT**

* JUnit test suites enable the grouping and execution of multiple test classes together.
* This is beneficial for organizing tests by functionality or module, streamlining test execution, and managing test dependencies.
* Using JUnit test suites, multiple tests can be run simultaneously. In most circumstances, testing each possible class individually is not preferable. The use of test suites is crucial in reaching this categorization goal.
* Test suites are the logical grouping or collection of test cases to run a single job with different test scenarios.
* For instance, a test suite for product purchase has multiple test cases, like: Test Case 1: Login. Test Case 2: Adding Products.

1. **Frequently used Selenium commands**

## **1. Browser Commands**

Used to open, close, or manage the browser.

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

driver.getTitle();

driver.getCurrentUrl();

driver.getPageSource();

driver.close();

driver.quit();

## **2. Navigation Commands**

Used to navigate browser history.

driver.findElement([By.id](http://by.id)("username")).sendKeys("Dharti123");

driver.findElement(By.name("password")).sendKeys("Test@123");

driver.findElement(By.id("loginBtn")).click();

driver.findElement(By.id("search")).clear();

1. **parameterise Test in Junit**

* JUnit's parameterized tests allow the execution of the same test method multiple times with different input values. This approach reduces code duplication and enhances test coverage.
* In JUnit, parameterized tests allow you to run the same test logic with different inputs.