**Selenium IDE:**

Selenium IDE (Integrated Development Environment) is primarily a record/run tool that a test case developer uses to develop Selenium Test cases. Selenium IDE is an easy-to-use tool from the Selenium Test Suite and can even be used by someone new to developing automated test cases for their web applications. One does not require any special setup to get started with Selenium IDE. You just need to add the extension of your specific browser. Selenium IDE provides you with a GUI (Graphical User Interface) for easily recording your interactions with the website.

Selenium IDE allows a user or a test case developer to create the test cases and test suites and edit it later as per their requirements. The development environment also provides the capability of converting test cases to different programming languages, which makes it easier for the user and does not mandate the need for knowing a specific programming language.

**Features of Selenium IDE**

There are several features provided in the IDE under the toolbar, using which one can control the execution of test cases:

* Speed Control – Helps control the speed of test cases
* Run All – Allows execution of the entire Test Suite
* Run – Runs the currently selected test
* Pause/Resume – Allows a user to pause and resume a particular test case
* Step – Helps step into each specific command in the test script
* Rollup – Helps group all the **Selenese** Commands together and make them execute as a single operation

**Selenium WebDriver:**

Selenium WebDriver is a collection of open source APIs which are used to automate the testing of a web application.

Selenium supports automation of all the major browsers in the market through the use of WebDriver. WebDriver is an API and protocol that defines a language-neutral interface for controlling the behaviour of web browsers. Each browser is backed by a specific WebDriver implementation, called a driver. The driver is the component responsible for delegating down to the browser, and handles communication to and from Selenium and the browser.

This separation is part of a conscious effort to have browser vendors take responsibility for the implementation for their browsers. Selenium makes use of these third party drivers where possible, but also provides its own drivers maintained by the project for the cases when this is not a reality.

The Selenium framework ties all of these pieces together through a user-facing interface that enables the different browser backends to be used transparently, enabling cross-browser and cross-platform automation.

Selenium setup is quite different from the setup of other commercial tools. Before you can start writing Selenium code, you have to install the language bindings libraries for your language of choice, the browser you want to use, and the driver for that browser.  
 It supports many browsers such as Firefox, Chrome, IE, and Safari. However, using the Selenium WebDriver, we can automate testing for web applications only. It does not qualify for window-based applications. It also supports different programming languages such as C#, Java, Perl, PHP and Ruby for writing test scripts. Selenium Webdriver is platform-independent since the same code can be used on different Operating Systems like Microsoft Windows, Apple OS and Linux. It is one of the components of the selenium family, which also includes Selenium IDE, Selenium Client API, Selenium Remote Control and Selenium Grid.  
  
Selenium WebDriver does not handle window component, but this limitation can be overcome by using external tools such as AUTO IT tool, Sikuli etc. It has different location strategies as well such as ID, Name, Link text, Partial link text, Class name, CSS selector and Xpath. It also has better support dynamic web pages like Ajax, where elements of the web page may change without the page itself being reloaded. By using different jar files, we can also test API, Database Test etc. using Selenium WebDriver.

**Selenuim Grid:**

Selenium Grid is a smart proxy server that makes it easy to run tests in parallel on multiple machines. This is done by routing commands to remote web browser instances, where one server acts as the hub. This hub routes test commands that are in JSON format to multiple registered Grid nodes.

Hub enables simultaneous execution of tests on multiple machines, managing different browsers centrally, instead of conducting different tests for each of them. Selenium Grid makes cross browser testing easy as a single test can be carried on multiple machines and browsers, all together, making it easy to analyze and compare the results.

The two major components of the Selenium Grid architecture are:

* **Hub** is a server that accepts the access requests from the WebDriver client, routing the JSON test commands to the remote drives on nodes. It takes instructions from the client and executes them remotely on the various nodes in parallel
* **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.

Testers should use Selenium Grid in the following circumstances:

* To run tests on multiple browsers and their versions, different devices, and operating systems
* To reduce the time that a test suite takes to complete execution

Selenium Grid improves the turnaround time of the test results. It is especially useful when the test suite is large and takes more time to run. It offers flexibility and ensures maximum test coverage within a limited time. Since the virtual infrastructure is in use, maintenance becomes easy.

**Selenuim:**

Selenium is a framework of automation testing tools, based on the JavaScript framework. Automated Selenium testing is greatly favoured by QAs for replicating end-user actions on websites to monitor their behaviour. It drives the interactions that occur on the target web page and could run them automatically, without requiring manual inputs.

Selenium suite comprises four components:

* Selenium IDE
* Selenium RC
* Selenium Webdriver
* Selenuim Grid

Selenium WebDriver works by emulating the actions of a user. When you write a Selenium test, you specify the actions you want the user to take, and Selenium will automatically execute those actions on the browser.

For example, if you need to test a login functionality, you would write a Selenium test to enter the username and password into the appropriate fields and click on the login button. Selenium would automatically execute those actions on the browser and report any errors.

**Benefits of Selenium in Automation Testing:**

* Efficient and accurate web application testing.
* The ability to test your web application on multiple browsers and operating systems.
* The ability to run more than one test at the same time.

With a Selenium Grid, you can significantly reduce your time to test your web application. And by using a Selenium Grid, you can ensure that your web application is fully functional before releasing it to users. So if you want to improve your web application testing, consider using a Selenium Grid. It's one of the best ways to automate your web application testing.

**Selenium in an Agile Environment**

Selenium is an all-in-one tool that can help you streamline your agile testing process. By following the tips below, you can ensure that your automated tests are practical and efficient and that they play a valuable role in your agile development cycle

Automated tests should be run frequently as part of the continuous integration process.

* Tests should be written to allow them to be run quickly and easily.
* Tests should be designed to test a specific functionality or behavior and should not be too complex.
* New features and changes should be accompanied by automated tests to ensure that the application's functionality remains intact.
* Automated tests should supplement manual testing rather than replace it altogether.
* As part of the regression testing process, to ensure that existing features continue to work as expected after new code has been added.
* To verify that new features are working as expected before they are released to production.
* To help identify and troubleshoot bugs in web applications at both business and development levels before releasing the app.

**Browser Driver used in selenium:**

Selenium automates browser and helps us in automation of web application testing across different browsers. Selenium API has provided many classes and interfaces to work with different types of browsers and HTML elements.

**Selenium WebDriver Interface:**

Selenium WebDriver is an interface that defines a set of methods. However, implementation is provided by the browser specific classes. Some of the implementation classes are AndroidDriver, ChromeDriver, FirefoxDriver, InternetExplorerDriver, IPhoneDriver, SafariDriver etc. The WebDriver main functionality is to control the browser. It even helps us to select the HTML page elements and perform operations on them such as click, filling a form fields etc.

If we want to execute your test cases in a Firefox browser we have to use FirefoxDriver class. Similarly, if we want to execute the test cases in the Chrome browser we have to use ChromeDriver class.

**Selenium WebDriver Methods:**

SearchContext is the topmost interface in Selenium API which has two methods - findElement() and findElements(). Selenium WebDriver interface has many abstract methods like get(String url), quit(), close(), getWindowHandle(), getWindowHandles(), getTitle() etc. WebDriver has nested interfaces like Window, Navigation, Timeouts etc. These nested interfaces are used to perform operations like back(), forward() etc.

**Method Description:**

**get(String url):**

This method will launch a new browser and opens the given URL in the browser instance.

**getWindowHandle():**

It is used to handle single window i.e. main window. It return type is string. It will returns browser windlw handle from focused browser.

getWindowHandles():

It is used to handle multiple windows. It return type is Set. It will returns all handles from all opened browsers by Selenium WebDriver.

**close():**

This command is used to close the current browser window which is currently in focus.

**quit():**

This method will closes all the browsers windows which are currently opened and terminates the WebDriver session.

**getTitle():**

This method is used to retrieve the title of the webpage the user currently working on.

**Class implementing WebDriver:**

The major implementation classes of WebDriver interface are ChromeDriver, EdgeDriver, FirefoxDriver, InternetExplorerDriver etc. Each driver class corresponds to a browser. We simply create the object of the driver classes and work with them.

| **Class** | **Description** |
| --- | --- |
| ***ChromeDriver*** | It helps you to execute Selenium Scripts on Chrome browser. |
| ***FirefoxDriver*** | It helps you to execute Selenium Scripts on Firefox browser. |
| ***InternetExplorerDriver*** | It helps you to execute Selenium Scripts on InternetExplorer browser. |

**Commands used to access the WebElement:**

Selenium WebElement represents an HTML element. We can get an instance of WebElement using findElement() method and then perform specific actions such as click, submit etc. Some of the commonly used WebElement methods are:

| **Command** | **Description** | **Syntax** |
| --- | --- | --- |
| ***findElement()*** | This method finds the first element within the current web page by using given locator. | WebElement element = driverObject.findElement(By.locator(“value”)); |
| ***sendKeys()*** | This method enters a value in to an Edit Box or Text box. | driver.findElement(By.elementLocator(“value”)).sendkeys(“value”); |
| ***clear()*** | It clears the Value from an Edit box or Text Box. | driverObject.findElement(By.locatorname(“value”)).clear(); |