STQA PRACTICAL THEORY

STQA PRACTICAL NO: 2

AIM: Implement Web Drivers on Chrome & Firefox Browsers.

THEORY:

WebDriver

WebDriver drives a browser natively, as a user would, either locally or on a remote machine using the Selenium server, marks a leap forward in terms of browser automation.

Selenium WebDriver refers to both the language bindings and the implementations of the individual browser controlling code. This is commonly referred to as just WebDriver.

Selenium WebDriver is a W3C Recommendation

- WebDriver is designed as a simple and more concise programming interface.
- WebDriver is a compact object-oriented API.
- It drives the browser effectively.

DOWNLOAD URL OF DRIVERS / LIBS:

- SELENIUM DRIVER: https://github.com/SeleniumHQ/selenium/releases/download/selenium-4.4.0/selenium-java-4.4.0.zip
- CHROME DRIVER: https://chromedriver.storage.googleapis.com/105.0.5195.52/chromedriver_win32.zip
- GECKODRIVER: https://github.com/mozilla/geckodriver/releases

---

STQA PRACTICAL

AIM: Demonstrate handling multiple frames in selenium

THEORY:

iFrame in Selenium Webdriver

iFrame in Selenium Webdriver is a web page or an inline frame which is embedded in another web page or an HTML document embedded inside another HTML document. The iframe is often used to add content from other sources like an advertisement into a web page. The iframe is defined with the <iframe> tag.
We can identify the frames in Selenium using methods given below:

- Right click on the element, if you find the option like ‘This Frame’ then it is an iframe. (Please refer the below steps)
- Right click on the page and click ‘View Page Source’ and Search with the ‘iframe’, if you can find any tag name with the ‘iframe’ then it means to say the page consists of an iframe.

In the above diagram, you can see that ‘**This Frame**’ option is available upon right clicking, so we are now sure that it is an iframe.

We can even identify the total number of iframes by using the below code:

```
Int size = driver.findElements(By.tagName("iframe")).size();
```

STQA PRACTICAL NO: 3

AIM:

A) Implement Browser command and navigation Commands.
B) Implement the find element command.
C) Demonstrate the Locator(id,css selector, path).
D) Demonstrate synchronization in selenium

THEORY:

Implement Browser Command and Navigation Commands:

In Selenium, browser commands and navigation commands are essential for controlling the behavior of the web browser. These commands allow you to open and manage browser sessions, navigate to different web pages, and interact with web elements.

Browser Commands:
get(String url): Loads a new web page in the current browser window or tab.
For example, driver.get("https://www.example.com"); opens the "https://www.example.com" URL.
close(): Closes the current browser window or tab.
quit(): Closes all open browser windows and ends the WebDriver session.

Navigation Commands:
navigate().to(String url): Navigates to a specified URL, similar to get().
navigate().back(): Navigates to the previous page in the browser's history.
navigate().forward(): Navigates to the next page in the browser's history.
navigate().refresh(): Reloads the current page.

Implement the Find Element Command

The "find element" command in Selenium is used to locate and interact with web elements on a web page. You can use this command to identify elements such as buttons, text fields, links, checkboxes, and more. For example, you can use the findElement method to locate an element by its attributes, like its id, name, class, or xpath. Here's a basic example:
java

Example Code:

```
WebElement element = driver.findElement(By.id("exampleId"));
element.click(); // Perform an action on the element, like clicking.
```

Demonstrate the Locator (id, CSS Selector, XPath)
Locators are used to identify and locate web elements on a web page.

Three common locators in Selenium are:
ID Locator: You can locate an element by its HTML id attribute. For example, By.id("exampleId") locates an element with the specified ID.
CSS Selector Locator: CSS selectors are a powerful way to locate elements using CSS attribute selectors. For example, By.cssSelector("input[name='username']") finds an input element with the attribute name equal to "username."
XPath Locator: XPath is a language for navigating XML documents, and it can be used to locate elements in HTML. For example, By.xpath("//input[@id='username']") finds an input element with the id attribute equal to "username." Each locator has its own syntax and strengths, and you can choose the most appropriate one based on the web page's structure and element attributes.

Demonstrate Synchronization in Selenium
Synchronization in Selenium is a crucial concept for dealing with dynamic web pages and ensuring that web elements are interacted with only when they are ready. There are two main types of synchronization:
Implicit Wait: An implicit wait sets a maximum time that the WebDriver will wait for a web element to appear on the page before throwing an exception. It's applied globally to the entire WebDriver instance.

Example Code:
```
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
```

Explicit Wait: An explicit wait allows you to wait for a specific condition to be met before proceeding with further actions. It's more flexible and can be applied to specific elements or conditions.

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

Synchronization helps you handle situations where web elements are loaded dynamically, have animations, or are located in iframes. It ensures that your test scripts are robust and reliable.
++

Selenium WebDriver - Browser Commands:

The very basic browser operations of WebDriver include opening a browser; perform few tasks and then closing the browser.

Given are some of the most commonly used Browser commands for Selenium WebDriver.

1. **Get Command:**
   Method: get(String arg0) : void
   In WebDriver, this method loads a new web page in the existing browser window. It accepts String as parameter and returns void.

   Implementation:

```
driver.get(URL);
OR
String URL = "URL";
driver.get(URL);
```

0. **Get Title Command:**
   Method: getTitle(): String
   In WebDriver, this method fetches the title of the current web page. It accepts no parameter and returns a String.

   Implementation:

```
driver.getTitle();
OR
String Title = driver.getTitle();
```

0. **Get Current URL Command:**
   Method: getCurrentUrl(): String
   In WebDriver, this method fetches the string representing the Current URL of the current web page. It accepts nothing as parameter and returns a String value.

   Implementation:

```
driver.getCurrentUrl();
OR
String CurrentUrl = driver.getCurrentUrl();
```

0. Close Command:

Method: close(): void

This method terminates the current browser window operating by WebDriver at the current time. If the current window is the only window operating by WebDriver, it terminates the browser as well. This method accepts nothing as parameter and returns void.

Implementation:

```
driver.close();
```

++

FindElement:

Selenium Find Element command takes in the By object as the parameter and returns an object of type list WebElement in Selenium. By object in turn can be used with various locator strategies such as find element by ID Selenium, Name, Class Name, XPATH etc.

Below is the syntax of FindElement command in Selenium web driver:

```
WebElement elementName =
driver.findElement(By.LocatorStrategy("LocatorValue"));
```

Locator Strategy can be any of the following values.

- ID
- Selenium find element by Name
- Class Name
- Tag Name
- Link Text
- Partial Link Text
- XPATH

Locator Value is the unique value using which a web element can be identified. It is the responsibility of developers and testers to make sure that web elements are uniquely identifiable using certain properties such as ID or name.

FindElements:

FindElements in Selenium command takes in By object as the parameter and returns a list of web elements. It returns an empty list if there are no elements found using the given locator strategy and locator value.

Below is the syntax of find elements command.

```
List<WebElement> elementName =
driver.findElements(By.LocatorStrategy("LocatorValue"));
```

STQA PRACTICAL NO: 4

AIM:

1. Demonstrate different types of alerts.
2. Demonstrate : Handling Drop Down & List Boxes.
3. Demonstrate: Command Button, Radio buttons & text boxes, Waits command in selenium.
4. Demonstrate action classes in Selenium.

THEORY:

Demonstrate Different Types of Alerts:
Alerts are popup windows that appear in a web application to provide information or request user input.

There are three types of alerts:

- **Alerts:** These are simple pop-up boxes that display a message and require the user to click "OK" to dismiss them.
- **Confirmation Alerts:** These present the user with a choice, usually "OK" or "Cancel." They are used to confirm or cancel an action.
- **Prompt Alerts:** These allow the user to enter some text in addition to the "OK" and "Cancel" options. They are used when user input is required.

Selenium's **Alert interface** is used to interact with these pop-up alerts.

Handling Drop Down & List Boxes:

Dropdowns and list boxes are HTML elements that allow users to select one or more options from a list.
In Selenium, you can interact with dropdowns and list boxes using the Select class, which provides methods to select and deselect options by index, value, and visible text. You can also get the selected options and check if they are multi-select.

Demonstrate Command Button, Radio Buttons, Text Boxes, and Waits in Selenium:

- **Command Button:** A command button is a form element used to trigger an action, such as form submission.
- **Radio Buttons:** Radio buttons are used when you want users to select a single option from a list. They are typically used in groups, and only one option can be selected at a time.
- **Text Boxes:** Text boxes are used for user input, such as entering a name or other information.
- **Waits:** Waits are used to ensure that Selenium waits for a certain condition to be met before proceeding with the script. There are two main types of waits in Selenium:
    - **Implicit Wait:** This sets a maximum time for Selenium to wait for an element to be available. It is defined once and applies to all subsequent interactions.
    - **Explicit Wait:** This is used for a specific element and condition. It allows you to wait for an element to meet a certain condition before proceeding.

Demonstrate Action Classes in Selenium:

Action classes in Selenium provide methods for advanced user interactions like mouse movements, drag-and-drop, double-click, right-click, etc.
The Actions class in Selenium allows you to create complex sequences of actions, such as:

- **Moving the mouse to an element.**
- **Clicking on an element.**
- **Dragging and dropping elements.**
- **Performing keyboard actions.**

Action chains are typically used when simulating user interactions, like moving the mouse to a menu item before clicking it.

STQA PRACTICAL NO: 5

AIM:

1. Installation of TestNg , running testNg and TestNg annotations.
2. Demonstrate data driven Framework.
3. Demonstrate Validation testing.
4. Perform regression testing.

THEORY:

TestNG and TestNG Annotations:

TestNG is a testing framework for Java that makes it easy to perform unit testing, integration testing, and end-to-end testing of applications. It is inspired by JUnit but introduces new functionalities that make it more powerful and easier to use. Here are the key steps to install TestNG, run TestNG tests, and understand TestNG annotations:

- **Installation of TestNG:** You can install TestNG in several ways, but the most common method is to use a build tool like Maven or Gradle. For Maven, you need to add a dependency to your project's pom.xml file. For Gradle, you can add it to your build.gradle file. Alternatively, you can download the TestNG JAR and add it to your project's classpath.

- **Running TestNG Tests:** To run TestNG tests, you can use the TestNG XML configuration file or run tests programmatically. You can execute TestNG tests from your IDE or through the command line.
- **TestNG Annotations:** TestNG uses annotations to mark methods as test methods and to control the test execution flow. Some common TestNG annotations include @Test, @BeforeSuite, @AfterSuite, @BeforeTest, @AfterTest, @BeforeClass, @AfterClass, @BeforeMethod, and @AfterMethod. These annotations help you set up test conditions and manage the test lifecycle.

Data-Driven Framework:

A data-driven testing framework is a methodology that separates test scripts and test data. It allows you to execute the same test script with multiple sets of data. Here's how it works:

- **Test Data:** You maintain test data separately, typically in external files like Excel spreadsheets, CSV files, or a database.
- **Test Scripts:** You create test scripts that are parameterized to accept data from the external sources. Instead of hardcoding data within the script, you fetch the data dynamically.
- **Execution:** The test framework runs the same test script multiple times, each time with a different set of test data. This allows you to perform tests with different input values and verify the results.

Data-driven frameworks are useful for running tests with various data permutations, which is common in scenarios like form validation, login testing, and more.

Validation Testing:

Validation testing is a type of testing that focuses on validating the software's compliance with specific standards, requirements, or regulations. It typically involves:

- **Compliance Testing:** Checking if the software adheres to industry standards, legal requirements, or regulatory guidelines.
- **Functional Testing:** Ensuring that the software's functions and features perform as expected according to specifications.
- **Security Testing:** Validating the software's security measures, such as authentication, authorization, and data protection.
- **Usability Testing:** Verifying that the software is user-friendly and meets user experience expectations.

Validation testing is essential for software that needs to meet specific compliance or regulatory requirements, such as healthcare software (HIPAA compliance) or financial software (SOX compliance).

Regression Testing:

Regression testing is a type of testing that ensures that new code changes (fixes, enhancements, new features) do not adversely affect the existing functionality of the software. It involves the following steps:

- **Test Selection:** Select a set of test cases from your test suite that covers the existing functionality of the software.
- **Re-execution:** Execute the selected test cases on the new version of the software.
- **Comparison:** Compare the results of the new test runs with the expected results. Any deviations indicate potential regression issues.
- **Bug Reporting:** If regressions are identified, they need to be reported and fixed.

Regression testing is critical to maintaining software quality as it helps catch unintended side effects of code changes and ensures that the software remains reliable.

1. Installation of TestNg , running testNg and TestNg annotations.

Installation Steps:

Step 1: Go to the github repository:
https://github.com/testng-team/testng-eclipse

Step 2: Download the latest version zip file.

Step 3: Add all the jar files to the project classpath.