**SELENIUM INTERVIEW QUESTIONS**

1.What are the locators are in Selenium?

In Selenium, locators are mechanisms used to locate elements on a web page accurately so that actions can be performed on them. Here are some common locators used in Selenium:

ID: This locator selects an element based on its unique ID attribute.

driver.find\_element\_by\_id("element\_id")

Name: It selects elements based on their name attribute.

driver.find\_element\_by\_name("element\_name")

Class Name: This locator selects elements based on their class attribute.

driver.find\_element\_by\_class\_name("element\_class")

Tag Name: It selects elements based on their HTML tag name.

driver.find\_element\_by\_tag\_name("tag\_name")

Link Text: This locator selects anchor elements <a> by the visible text in the link.

driver.find\_element\_by\_link\_text("link\_text")

Partial Link Text: Similar to link text, but it matches a partial text of the link.

driver.find\_element\_by\_partial\_link\_text("partial\_link\_text")

XPath: XPath is a powerful locator that can traverse the entire document structure.

driver.find\_element\_by\_xpath("//xpath\_expression")

CSS Selector: CSS selectors are patterns used to select elements.

driver.find\_element\_by\_css\_selector("css\_selector")

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

2.Which locator is faster among all the locators?

The speed of locators can vary depending on various factors such as the complexity of the web page, the browser being used, and the implementation details of the Selenium framework

Among the commonly used locators, ID and Name locators are often considered to be faster because they directly leverage the browser's native getElementById() and getElementsByName() functions.

Additionally, CSS selectors are usually faster than XPath because browsers are optimized to parse and evaluate CSS selectors efficiently.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

3)What is the difference between xpath and Css?

XPath (XML Path Language) and CSS (Cascading Style Sheets) are both powerful methods for locating elements on a web page in Selenium.

Performance:

XPath: XPath expressions can sometimes be slower than CSS selectors, especially in older browsers.

Syntax:

XPath: XPath uses a path expression to navigate through the XML structure of an HTML document. It allows for complex queries and traversals using a hierarchical structure.

CSS: CSS selectors follow the syntax of CSS and are used to select HTML elements based on their attributes, classes, IDs, and relationships with other elements.

Usage:

XPath: XPath is commonly used when complex traversals or attribute-based searches are required, or when elements cannot be easily targeted using CSS selectors.

CSS: CSS selectors are often preferred for their simplicity and readability, especially for selecting elements based on classes, IDs, or tag names.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

4)How can you perform click an element without using Click() method?

Performing a click on an element without using the click() method in Selenium can be achieved using JavaScript execution.

driver.get("https://example.com")

# Find the element you want to click

element = driver.find\_element\_by\_id("element\_id")

# Execute JavaScript to click the element

driver.execute\_script("arguments[0].click();", element)

# Close the WebDriver

driver.quit()

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

5)Explain about the JavaScriptExecutor?

The JavaScriptExecutor interface in Selenium allows you to execute JavaScript code within the context of the currently selected frame or window in the WebDriver. This interface provides methods for executing JavaScript code and returning values, which can be particularly useful for interacting with web elements.

example:

JavascriptExecutor executor =

(JavascriptExecutor)driver;

executor.executeScript("document.getElementBy

Id("textbox\_id").value='new value';);

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

6)Explain about the Actions class?

Action class is useful mainly for mouse and keyboard actions. In order to perform such actions, Selenium provides various methods.

Mouse Actions in Selenium:

doubleClick(): Performs double click on the element

clickAndHold(): Performs long click on the mouse without releasing it

dragAndDrop(): Drags the element from one point and drops to another

moveToElement(): Shifts the mouse pointer to the center of the element

contextClick(): Performs right-click on the mouse

Keyboard Actions in Selenium:

sendKeys(): Sends a series of keys to the element

keyUp(): Performs key release

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

7)Explain about the Frames concept?

The term "frames" in Selenium refers to segmenting an HTML document into several portions, each of which can contain a separate HTML document.

Selenium WebDriver has a few simple steps to handle frames:

Switch the driver's focus to the frame using the switchTo().frame() method.

Using web driver commands, interact with the elements of the frame and perform the operations.

Switch back to the web content by the switchTo().defaultContent() method.

By Index

Use the switchTo().frame() method in Selenium WebDriver and supply the frame's index as an argument to switch to a frame by index

By Name or ID

You can use the name or ID as a parameter to switch to the frame using the switchTo().frame()

By Web Element

We can also switch to the frame by web elements. Pass the web element as an argument in the switchTo().frame()

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

8)Explain about the windows concept?

it is a unique identifier that holds the address of all the windows. Think of it as a pointer to a window, which returns the string value.

Syntax

getwindowhandle(): This method helps to get the window handle of the current window

getwindowhandles(): This method helps to get the handles of all the windows opened

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

9)How will you select a value from a dropdown list and its return type?

To select a value from a dropdown list using Selenium, you can use the Select class, which provides methods for interacting with dropdown elements. The return type for selecting a value from a dropdown list depends on the method you use.

Select s=new Select(WebElement)

s.selectByValue()

s.selectByVisibleText()

s.selectByIndex()

s.getOptions(); List<WebElement>

s.getAllSelectedOptions(); List<WebElement>

s.getFirstSelectedOptions(); WebElement

s.isMultiple() Boolean

s.deSelectByValue()

s.deSelectByVisibleText()

s.deSelectByIndex()

s.deSelectAll()

10)Find element and find elements return type.?

|  |  |
| --- | --- |
| findElement | findElements |
| a)Returns the first matching web element if the locator discovers multiple web elements | a)Returns a list of multiple matching web elements |
| b)Detects a unique web element | b)Returns a collection of matching elements |
| c)Throws NoSuchElementException if the element is not found | c)Returns an empty list if no matching element is found |

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

11)How will you handle stale element exception?

Solution 1- Refresh the web page

Solution 2- Use the explicit wait

Solution 3- Use the try-catch statement

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

12)Difference btw Explicit wait and Fluent wait / wait in selenium?

In Selenium, both explicit waits and fluent waits are mechanisms for waiting for specific conditions to occur before

proceeding with the execution of the test script. However, they differ in terms of their implementation and flexibility.

Explicit Wait:

An explicit wait is implemented using the WebDriverWait class in Selenium.

It allows you to wait for a certain condition to be met before proceeding with the execution.

You specify the maximum amount of time to wait and the condition that needs to be satisfied.

Fluent Wait:

A fluent wait is a more flexible form of waiting compared to explicit wait.

It allows you to define polling intervals and exceptions to ignore during the waiting period.

Fluent waits are implemented using the FluentWait class in Selenium.

You can specify the maximum amount of time to wait, the polling interval, and the exceptions to ignore.

In Selenium, both explicit waits and fluent waits are mechanisms for waiting for specific conditions to occur before proceeding with the execution of the test script. However, they differ in terms of their implementation and flexibility.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

13)Difference between Absolute and Relative xpath?

Absolute XPath:

An absolute XPath specifies the complete path of an element from the root element (/) to the target element.

It starts with a single forward slash (/) representing the root node of the document and includes all the intermediate nodes and the target node, separated by forward slashes.

Absolute XPaths tend to be lengthy and may become brittle if there are changes in the structure of the document, as any modifications to the hierarchy above the target element will require updating the XPath.

Example of an absolute XPath:

/html/body/div[1]/div[2]/form/input[1]

Relative XPath:

A relative XPath specifies the path of an element relative to another element in the document.

It starts with the reference point (usually the current node or a known parent node) and includes only the nodes necessary to locate the target element.

Relative XPaths are more flexible and robust compared to absolute XPaths, as they are less affected by changes in the document structure. They are recommended for locating elements within dynamic web pages.

Example of a relative XPath:

//form/input[@name='username']

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

14.Give some xpath using following,sibiling?

XPath provides axes to navigate through elements in an XML or HTML document. Two commonly used axes are following-sibling and preceding-sibling, which allow you to select siblings that appear after or before a specific element, respectively.

Here are examples of XPath expressions using the following-sibling and preceding-sibling axes:

Following Sibling:

Selects sibling elements that appear after a specified element.

Syntax: //element\_name/following-sibling::sibling\_name

Example: If you have the following HTML structure:

<div>

<p>Paragraph 1</p>

<p>Paragraph 2</p>

<p>Paragraph 3</p>

</div>

To select the <p> elements that appear after the first <p> element:

//p[1]/following-sibling::p

This XPath expression will select the <p> elements with the text "Paragraph 2" and "Paragraph 3".

Preceding Sibling:

Selects sibling elements that appear before a specified element.

Syntax: //element\_name/preceding-sibling::sibling\_name

Example: Using the same HTML structure as above, to select the <p> elements that appear before the last <p> element:

//p[last()]/preceding-sibling::p

This XPath expression will select the <p> elements with the text "Paragraph 1" and "Paragraph 2".

1)How to find an element with dynamically changing ID?

Finding an element with a dynamically changing ID can be challenging but not impossible. Here are a few strategies you can try:

Use a CSS Class: If possible, ask the developer to add a specific CSS class to the element you want to target. Classes are more stable than IDs and less likely to change.

Use Relative XPath: If the element is within a predictable structure on the page, you can use XPath expressions that are relative to other elements. For example, if the element you want to find is always inside a <div> with a specific class, you can use XPath to navigate to it.

Use Partial Matches: If the ID contains a part that remains constant, you can use partial matching techniques. For example, if the ID always starts with "dynamic\_element\_", you can use a selector that targets elements with IDs starting with that string.

Attribute Contains Selector: In CSS or jQuery, you can use the "attribute contains" selector to target elements based on a part of their ID. For example, if the ID contains a specific keyword, you can select it using [id\*=keyword].

Data Attributes: If possible, ask the developer to add custom data attributes to the element. These attributes are designed to store extra information about an element and can be targeted reliably.

2)how to handle dynamic web Table?

Handling dynamic web tables can be a bit tricky, but there are several approaches you can take depending on your specific use case and the tools you're using. Here's a general guide:

Inspect the Table Structure: Before interacting with the table, inspect its structure using browser developer tools. Identify the HTML elements that make up the table, such as <table>, <tr> (table rows), <td> (table cells), and <th> (table headers).

Identify Stable Elements: Look for stable elements within the table, such as unique IDs, classes, or attributes of specific cells or rows that you can reliably use to locate and interact with them.

Use Automation Tools: If you're using a web automation tool like Selenium WebDriver, you can use its methods to find elements within the table. You can locate rows and cells based on their XPath, CSS selectors, or other attributes.

Dynamic XPath: Construct XPath expressions that dynamically locate elements based on changing attributes or text content. XPath functions like contains(), starts-with(), and following-sibling can be useful for navigating dynamic tables.

Iterate Through Rows and Columns: If the table structure is predictable but the content is dynamic, you can iterate through rows and columns using loops to extract data or perform actions on each cell.

3)How can you handle alerts and what are methods are available?

Alert is interface,it is a small message box displayed on the screen to give some infirmation to the user.alret and webpage are different,alret has no locators,when alert appeared on displayed first we need to switch into the alret to handle the alert then only user can perform the next opeartion in the webpage.

To switch into the alert:

Alert r = WebDriver.switchTo().alert();

Types of Alert:

1)simple alert:contains only ok button

2)confirm alert:contains both ok and cancel button

3)Promt alert:contains text box with ok and cancel button

r.accept(); --> to ACecpt thr alert

r.dismiss();---->to dismiss the alert

r.sendKeys();---->to insert the values(First we will send kend keys and then we click accept like Ok button)

r.getText(); ---> to print the text in the alert

4)Why we are using Set for getWindowHandles() and List for findElements() methods ?

The choice between using a Set and a List in Selenium WebDriver depends on the specific requirements and characteristics of the data being handled by each method.

getWindowHandles():

getWindowHandles() returns a set of window handles representing the currently open browser windows or tabs.

Using a Set ensures that each window handle is unique. Since window handles are typically unique identifiers for browser windows or tabs, a Set prevents duplication and ensures that each handle is included only once.

Duplicate window handles would not make sense in this context because each window or tab should have a distinct identifier.

Additionally, using a Set allows for efficient membership testing and does not guarantee any specific order of the window handles, which aligns well with the nature of browser windows or tabs.

findElements():

findElements() returns a list of WebElement objects matching the specified locator strategy.

Using a List allows for multiple matching elements to be stored and accessed in a specific order.

In web pages, it's common for multiple elements to match a given locator (e.g., multiple links, buttons, or table rows). Therefore, using a List enables you to capture all matching elements and iterate through them as needed.

The order of elements in the list corresponds to their order in the HTML document or the order in which they were found during the search process, which can be important for certain testing scenarios or automation tasks.

5)How to find the total number of radio button in a webpage?

To find the total number of radio buttons on a webpage using Selenium WebDriver, you can use the findElements() method to locate all radio button elements and then count the number of elements found

Here's how you can do it in Java:

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

import java.util.List;

public class RadioButtonCount {

public static void main(String[] args) {

// Set the path to the ChromeDriver executable

System.setProperty("webdriver.chrome.driver", "/path/to/chromedriver");

// Initialize ChromeDriver

WebDriver driver = new ChromeDriver();

// Navigate to the webpage

driver.get("https://example.com");

// Find all radio buttons on the page

List<WebElement> radioButtons = driver.findElements(By.xpath("//input[@type='radio']"));

// Get the count of radio buttons

int radioButtonCount = radioButtons.size();

System.out.println("Total number of radio buttons: " + radioButtonCount);

// Close the browser

driver.quit();

}

}

6)Which method is used to enter value in a text box?

To enter a value into a text box using Selenium WebDriver, you typically use the sendKeys() method. This method allows you to simulate keyboard input into the specified text box element. Here's how you can use it in Java:

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

public class TextBoxInput {

public static void main(String[] args) {

// Set the path to the ChromeDriver executable

System.setProperty("webdriver.chrome.driver", "/path/to/chromedriver");

// Initialize ChromeDriver

WebDriver driver = new ChromeDriver();

// Navigate to the webpage

driver.get("https://example.com");

// Find the text box element by its ID, name, XPath, or other selector

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

// Enter a value into the text box

textBox.sendKeys("Hello, world!");

// Close the browser

driver.quit();

}

}

83)How can we enter text without using sendkeys?

We can use JavaScript Executor to execute JavaScript code that sets the value of an input element directly. This bypasses the need for simulating keystrokes. Here's an example in Java:

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

JavascriptExecutor executor = (JavascriptExecutor)driver;

executor.executeScript("arguments[0].value='Your text here';", element);

84)Difference between cssSelector and Xpath? Which one is preferred?

Both CSS selectors and XPath are methods used to locate elements on a web page in order to interact with them using tools like Selenium. They have their own syntax and capabilities, and the preference between them often depends on the specific requirements of the project and personal preference. Here are some differences between CSS selectors and XPath:

Syntax:

CSS selectors use a syntax similar to CSS (Cascading Style Sheets) to locate elements. They use element attributes such as id, class, name, etc., to identify elements.

XPath (XML Path Language) is a language for navigating XML documents and is also applicable to HTML documents. XPath expressions can traverse the entire HTML structure and locate elements based on their relationships with other elements.

85)How will you switch to the third window and give the value in the textbox?

To switch to the third window and give a value in the textbox using Selenium WebDriver in Java, you can follow these steps:

Get handles of all the open windows.

Switch to the third window using its handle.

Find the textbox element and input the desired value.

Here's how you can do it:

// Assuming driver is initialized properly

String mainWindowHandle = driver.getWindowHandle();

// Get all window handles

Set<String> allWindowHandles = driver.getWindowHandles();

// Switch to the third window

String thirdWindowHandle = null;

int count = 0;

for (String handle : allWindowHandles) {

count++;

if (count == 3) {

thirdWindowHandle = handle;

break;

}

}

driver.switchTo().window(thirdWindowHandle);

// Now you are in the third window, locate the textbox and input the value

WebElement textBox = driver.findElement(By.id("textboxId")); // Replace "textboxId" with the actual ID of the textbox

textBox.sendKeys("Value to input");

// Switch back to the main window if needed

// driver.switchTo().window(mainWindowHandle);

86)How will you do mouse scroll in selenium?

import org.openqa.selenium.JavascriptExecutor;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.interactions.Actions;

public class ScrollExample {

public static void main(String[] args) {

// Set system property for Chrome driver

System.setProperty("webdriver.chrome.driver", "path\_to\_chromedriver");

// Initialize Chrome driver

WebDriver driver = new ChromeDriver();

// Open a webpage

driver.get("https://example.com");

// Create Actions object

Actions actions = new Actions(driver);

// Scroll down using Actions class

actions.moveToElement((WebElement) driver.findElement(By.tagName("body"))).sendKeys(Keys.PAGE\_DOWN).build().perform();

// Alternatively, you can scroll to a specific element

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

actions.moveToElement(element).perform();

// Scroll using JavaScript Executor

JavascriptExecutor js = (JavascriptExecutor) driver;

js.executeScript("window.scrollBy(0, 250)"); // Scrolls 250 pixels down

// Scroll to the bottom of the page

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

// Scroll to the top of the page

js.executeScript("window.scrollTo(0, 0)");

}

}

We first create an Actions object.

We use the moveToElement() method to move the mouse to a specific element (or the body element if you want to scroll the entire page).

Then, we use sendKeys(Keys.PAGE\_DOWN) to simulate pressing the PAGE DOWN key, which scrolls the page down.

We can also use JavaScriptExecutor to scroll using JavaScript code. window.scrollBy(xOffset, yOffset) scrolls the window by the specified number of pixels.

To scroll to a specific element, we can use moveToElement() method with the desired element, and then call perform() to perform the action.

87)In popup,if you want cancel the popup Which commend use it?

// Assuming driver is initialized properly

Alert alert = driver.switchTo().alert();

alert.dismiss();

This code assumes that the popup is a JavaScript alert, prompt, or confirmation dialog. driver.switchTo().alert() is used to switch the WebDriver's focus to the alert, and then alert.dismiss() dismisses (cancels) the alert.

88)How will you do Double click and right click in selenium?

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.interactions.Actions;

public class DoubleClickExample {

public static void main(String[] args) {

// Set system property for Chrome driver

System.setProperty("webdriver.chrome.driver", "path\_to\_chromedriver");

// Initialize Chrome driver

WebDriver driver = new ChromeDriver();

// Open a webpage

driver.get("https://example.com");

// Find the element to double click

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

// Create Actions object

Actions actions = new Actions(driver);

// Perform double click

actions.doubleClick(element).perform();

}

}

right click:

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.interactions.Actions;

public class RightClickExample {

public static void main(String[] args) {

// Set system property for Chrome driver

System.setProperty("webdriver.chrome.driver", "path\_to\_chromedriver");

// Initialize Chrome driver

WebDriver driver = new ChromeDriver();

// Open a webpage

driver.get("https://example.com");

// Find the element to right click

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

// Create Actions object

Actions actions = new Actions(driver);

// Perform right click

actions.contextClick(element).perform();

}

}

We first create an Actions object.

We find the element on which we want to perform the action.

We call the respective method (doubleClick() for double-click and contextClick() for right-click) on the Actions object, passing the element as a parameter.

Finally, we call perform() to execute the action.

89)What is the usage of right click option in selenium?

In Selenium WebDriver, the right-click option (or context click) can be used to simulate user interactions with right-click menus on web elements. This functionality is useful for testing scenarios where interactions with context menus are necessary or to automate actions that involve right-clicking

Testing Context Menus: Many web applications utilize context menus to provide additional options or actions when right-clicking on certain elements. Selenium can simulate this behavior by right-clicking on elements and then interacting with the resulting context menu options.

Interacting with Web Applications: Some web applications have functionality that is triggered by right-click events. Automating these interactions with Selenium allows for comprehensive testing of such applications.

Verifying Context Menu Actions: Selenium can verify that the correct context menu options appear when right-clicking on specific elements, and then validate the behavior of those options when selected.

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.interactions.Actions;

public class RightClickExample {

public static void main(String[] args) {

// Set system property for Chrome driver

System.setProperty("webdriver.chrome.driver", "path\_to\_chromedriver");

// Initialize Chrome driver

WebDriver driver = new ChromeDriver();

// Open a webpage

driver.get("https://example.com");

// Find the element to right click

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

// Create Actions object

Actions actions = new Actions(driver);

// Perform right click

actions.contextClick(element).perform();

}

}

90)What is the usage movetoElement option once use of Action class?

The moveToElement() method in the Actions class of Selenium WebDriver is used to move the mouse cursor to the center of a specified web element. This method is often used in conjunction with other actions to perform complex user interactions, such as hovering over an element to reveal a dropdown menu or triggering a tooltip.

Here's how you can use moveToElement() in Selenium WebDriver:

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.interactions.Actions;

public class MoveToElementExample {

public static void main(String[] args) {

// Set system property for Chrome driver

System.setProperty("webdriver.chrome.driver", "path\_to\_chromedriver");

// Initialize Chrome driver

WebDriver driver = new ChromeDriver();

// Open a webpage

driver.get("https://example.com");

// Find the element to move the mouse to

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

// Create Actions object

Actions actions = new Actions(driver);

// Move mouse to the center of the element

actions.moveToElement(element).perform();

}

}

91)What kind of report we can generate in Cucumber?

Cucumber gives us the capability to generate reports as well in the form of

HTML Reports: Cucumber by default generates HTML reports which provide a detailed view of test execution results.

These reports typically include information such as feature files, scenarios, step definitions, test results (pass/fail), and any associated error messages or stack traces.

HTML reports are easy to read and can be shared with stakeholders who may not be familiar with the technical details of the tests.

JSON Reports: Cucumber can also generate JSON-formatted reports which contain structured data representing the test results.

JSON reports can be processed programmatically to extract specific information or integrate with other tools and systems for further analysis or reporting.

JUnit XML Reports: Cucumber can output test results in JUnit XML format, which is commonly used for integration with Continuous Integration (CI) tools like Jenkins.

JUnit XML reports provide a standardized format for representing test results, making it easy to integrate Cucumber tests into CI/CD pipelines.

Custom Reports: Cucumber allows you to customize and extend the reporting capabilities to generate reports tailored to your specific requirements.

You can create custom report formats using plugins or by writing code to process the test results and generate the desired output.

Execution Reports: These reports provide a high-level summary of test execution, including the total number of scenarios, passed scenarios, failed scenarios, and any pending or skipped scenarios.

Execution reports help track the overall progress and success of the test suite.

Tag Reports: Cucumber allows you to tag scenarios with descriptive tags to categorize them based on different criteria (e.g., features, priority, environment).

Tag reports provide a breakdown of test results based on the tags applied to scenarios, making it easy to analyze results by specific categories.

Historical Reports: Some reporting tools and plugins for Cucumber offer historical reporting capabilities, allowing you to track trends and changes in test results over time.

Historical reports help identify patterns, regression issues, and improvements in test coverage and reliability.

92)How to customize the extent report? (Which means while generate the report how we can add the details like which system is executed person name)

Extent Reports is a popular reporting library in Java for test automation frameworks like TestNG and JUnit. Here's a step-by-step guide on how to achieve this:

Add Dependencies: Ensure you have the necessary dependencies added to your project. Include the Extent Reports library in your project's build path or dependency management tool (e.g., Maven, Gradle).

Initialize Extent Reports: Initialize an Extent Reports object at the beginning of your test suite or test execution. This object serves as the entry point for creating your report.

ExtentReports extent = new ExtentReports("path/to/your/report.html", true);

Create ExtentTest Objects: For each test case, create an ExtentTest object. This object represents an individual test in your report.

ExtentTest test = extent.createTest("Test Name", "Test Description");

Add Details to ExtentTest: Utilize the ExtentTest object to add details specific to each test case. You can add system information and person's name using the assignAuthor() and assignCategory() methods.

test.assignAuthor("John Doe");

test.assignCategory("System: XYZ");

Execute Tests: Execute your test cases as usual, ensuring that you call appropriate ExtentReports and ExtentTest methods at relevant points in your test scripts.

Flush and Close ExtentReports: After all tests have been executed, flush the ExtentReports object to generate the report and close any associated resources.

extent.flush();

extent.close();

93)What test closure reports you will send?

The test closure report includes all the necessary details that can support the completion of the testing process. The report also has screenshots, query results, recordings, and log files that gives a detailed account of the testing process. The contents of test closure report are as follows:

Test Summary Report

Identifier

Test Summary

Variances

Comprehensive Assessment

Summary of Results

Evaluation

Summary of Activities

Approval

94)What are the reports you have used in your previous project/company?

TestNG generates HTML reports by default after test execution, providing detailed information about test results, including passed, failed, and skipped tests.

The reports include information such as test method names, execution time, status (pass/fail/skip), stack traces for failed tests, and more.

TestNG reports also include overall statistics like total tests run, passed, failed, and skipped tests.

Cucumber generates reports in various formats, including HTML, JSON, and XML, after test execution.

HTML reports provide a detailed view of test results, including feature and scenario names, status (pass/fail/skip), execution time, and any associated error messages.

The reports also include aggregate statistics like total features, scenarios, passed, failed, and skipped scenarios.