1. Setting up the Environment:

* **Node.js:**

Selenium with JavaScript requires Node.js, a runtime environment for JavaScript, to be installed. You can download it from the official [Node.js website](https://nodejs.org/).

* **NPM (Node Package Manager):**

NPM is the package manager for Node.js and is used to install Selenium WebDriver. You'll also need to check if NPM is installed along with Node.js.

* **Selenium WebDriver:**

Install Selenium WebDriver using NPM: npm install selenium-webdriver.

* **IDE (Integrated Development Environment):**

Choose an IDE like VS Code or Eclipse for writing and running your test scripts.

* **Browser Driver:**

Download the appropriate browser driver for your chosen browser (e.g., ChromeDriver for Chrome, GeckoDriver for Firefox).

Writing and Executing JavaScript Test Scripts:

* **Import Selenium:** Import the Selenium WebDriver library in your JavaScript file (e.g., const webdriver = require('selenium-webdriver');).
* **Create a WebDriver Instance:** Create a WebDriver instance for your browser: const driver = new webdriver.Builder().forBrowser('chrome').build();.
* **Navigate to a Website:** Use the driver.get() method to navigate to a URL: await driver.get('https://www.example.com');.
* **Locate Elements:** Use methods like driver.findElement() or driver.findElements() to locate elements on the page using various locators (e.g., ID, name, class, CSS selector).
* **Interact with Elements:** Perform actions on the elements, such as sending keys to a text field, clicking a button, or selecting a dropdown option.
* **Execute JavaScript:** Use driver.executeScript() to execute JavaScript code within the browser.
* **Wait for Elements:** Use driver.wait() to wait for elements to load before interacting with them.
* **Assertions:** Use assertions (e.g., using testing frameworks like Mocha or Jest) to verify expected outcomes.
* **Close the Driver:** Use driver.quit() to close the browser and release resources.
* **Running the script:** Use node <filename.js> in the terminal to run the test script.

3. Example:

JavaScript

const webdriver = require('selenium-webdriver');  
  
async function example() {  
 const driver = new webdriver.Builder().forBrowser('chrome').build();  
  
 try {  
 await driver.get('https://www.google.com');  
 const title = await driver.getTitle();  
 console.log('The title is: ', title);  
 } finally {  
 await driver.quit();  
 }  
}

JavaScript has several data types that can be categorized into **primitive** and **non-primitive** types. Here's a concise breakdown:

Primitive Data Types

1. **String**: Represents textual data.
   * Example: "Hello, World!", 'JavaScript'
2. **Number**: Represents numeric values (both integers and floating-point numbers).
   * Example: 42, 3.14
3. **BigInt**: Represents integers larger than the Number type can handle.
   * Example: 1234567890123456789012345678901234567890n
4. **Boolean**: Represents logical values: true or false.
   * Example: true, false
5. **Undefined**: A variable that has been declared but not assigned a value.
   * Example: let x; // x is undefined
6. **Null**: Represents the intentional absence of any object value.
   * Example: let y = null;
7. **Symbol**: Represents a unique and immutable value, often used as object keys.
   * Example: let sym = Symbol('unique');

Non-Primitive (Reference) Data Types

1. **Object**: A collection of key-value pairs or more complex entities.
   * Example: { name: "Alice", age: 25 }
2. **Array**: A special type of object used to store ordered collections.
   * Example: [1, 2, 3, 4]
3. **Function**: A callable object that executes a block of code.
   * Example: function greet() { console.log("Hello!"); }
4. **Date**: Represents dates and times.
   * Example: new Date()
5. **RegExp**: Represents regular expressions.
   * Example: /\d+/

Special Notes

* **Dynamic Typing**: JavaScript is dynamically typed, meaning variables can hold any data type and change types during execution.
* **Type Checking**: Use typeof to check the type of a value.
  + Example: typeof 42; // "number"

For Printing:

console.log("Hello, World!");

1. **alert()**: Displays a message in a popup dialog box.

alert("Hello, World!");

1. **document.write()**: Writes directly to the HTML document (not recommended for modern web development).
2. Copy the codedocument.write("Hello, World!");
3. **innerHTML**: Updates the content of an HTML element.

Copy the codedocument.getElementById("output").innerHTML = "Hello, World!";

In JavaScript, NaN stands for "Not-a-Number." It is a special value that represents an invalid or undefined numerical result. Despite its name, NaN is technically of the **number** data type.

**Key Characteristics of NaN:**

1. **Type**:
2. Copy the codeconsole.log(typeof NaN); // "number"

Even though it means "Not-a-Number," its type is still number.

1. **Self-Inequality**:  
   NaN is the only value in JavaScript that is **not equal to itself**:
2. Copy the codeconsole.log(NaN === NaN); // false

To check if a value is NaN, use Number.isNaN():

Copy the codeconsole.log(Number.isNaN(NaN)); // true

1. **When Does NaN Occur?**  
   NaN is returned when a mathematical operation or function fails to produce a valid number:
   * Invalid arithmetic:
   * Copy the codeconsole.log(0 / 0); // NaN
   * console.log(Math.sqrt(-1)); // NaN
   * Parsing errors:
   * Copy the codeconsole.log(Number("abc")); // NaN
   * console.log(parseInt("xyz")); // NaN
2. **Propagation**:  
   Any operation involving NaN results in NaN:
3. Copy the codeconsole.log(NaN + 5); // NaN
4. console.log(NaN \* 2); // NaN
5. **Global isNaN() vs Number.isNaN()**:
   * isNaN() converts the value to a number before checking, which can lead to unexpected results:
   * Copy the codeconsole.log(isNaN("abc")); // true (because "abc" becomes NaN)
   * Number.isNaN() is stricter and checks if the value is exactly NaN:
   * Copy the codeconsole.log(Number.isNaN("abc")); // false

**Summary:**

NaN is a quirky but essential part of JavaScript's number system. Always prefer Number.isNaN() for accurate checks, and be cautious when dealing with operations that might result in NaN.

In JavaScript, let is not a datatype but a **keyword** used to declare variables. It allows you to create variables that are block-scoped, meaning they are only accessible within the block, statement, or expression where they are defined. This is different from the older var keyword, which has function or global scope.

In JavaScript, Selenium WebDriver is commonly used for browser automation. It provides various **methods** and **classes** to interact with web elements and control browser behavior. Below is an overview of key methods and classes used in Selenium with JavaScript:

1. **Builder**
   * Used to create a new WebDriver instance.
   * Example:
   * Copy the codeconst { Builder } = require('selenium-webdriver');
   * const driver = new Builder().forBrowser('chrome').build();
2. **By**
   * Provides mechanisms to locate elements on a web page.
   * Example:
   * Copy the codeconst { By } = require('selenium-webdriver');
   * const element = driver.findElement(By.id('exampleId'));
3. **WebElement**
   * Represents an element on the web page.
   * Example:
   * Copy the codeconst element = driver.findElement(By.name('username'));
   * element.sendKeys('testUser');
4. **Key**
   * Provides keyboard actions like pressing Enter, Tab, etc.
   * Example:
   * Copy the codeconst { Key } = require('selenium-webdriver');
   * element.sendKeys('testUser', Key.RETURN);
5. **Actions**
   * Used for complex user interactions like drag-and-drop, hover, etc.
   * Example:
   * Copy the codeconst { Actions } = require('selenium-webdriver');
   * const actions = driver.actions();
   * actions.moveToElement(element).click().perform();

Common Methods in Selenium (JavaScript)

1. **Browser Control**
   * get(url) - Opens a URL in the browser.
   * Copy the codedriver.get('https://example.com');
   * quit() - Closes the browser and ends the session.
   * Copy the codedriver.quit();
2. **Element Interaction**
   * findElement(locator) - Finds a single element.
   * const element = driver.findElement(By.css('.example-class'));
   * findElements(locator) - Finds multiple elements.
   * const elements = driver.findElements(By.tagName('button'));
   * click() - Clicks on an element.
   * Copy the codeelement.click();
   * sendKeys(value) - Sends input to an element.
   * element.sendKeys('Hello World');
3. **Element Properties**
   * getText() - Retrieves the text of an element.
   * Copy the codeconst text = await element.getText();
   * getAttribute(attributeName) - Gets an attribute value.
   * Copy the codeconst value = await element.getAttribute('value');
4. **Waits**
   * wait(condition, timeout) - Waits for a condition to be met.
   * Copy the codeconst { until } = require('selenium-webdriver');
   * driver.wait(until.elementLocated(By.id('exampleId')), 5000);
5. **Navigation**
   * navigate().to(url) - Navigates to a URL.
   * Copy the codedriver.navigate().to('https://example.com');
   * navigate().back() - Navigates back in browser history.
   * Copy the codedriver.navigate().back();
6. **Screenshots**
   * takeScreenshot() - Captures a screenshot of the current page.
   * Copy the codeconst screenshot = await driver.takeScreenshot();

Example Script

Here’s a simple example of using Selenium with JavaScript:

Copy the codeconst { Builder, By, Key, until } = require('selenium-webdriver');

(async function example() {

let driver = await new Builder().forBrowser('chrome').build();

try {

await driver.get('https://example.com');

const searchBox = await driver.findElement(By.name('q'));

await searchBox.sendKeys('Selenium WebDriver', Key.RETURN);

await driver.wait(until.titleContains('Selenium'), 5000);

} finally {

await driver.quit();

}

})();

Doubts : why CJS

Warning in Typescript code

Frames?

Await ?

Typescript:

1. **WebDriver**
   * Represents the browser instance.
   * Used to control the browser (e.g., navigating, interacting with elements).
2. **By**
   * Used to locate elements on a web page (e.g., by ID, class name, CSS selector, etc.).
3. **WebElement**
   * Represents an element on the web page.
   * Provides methods to interact with elements (e.g., click, send keys).
4. **Options**
   * Used to configure browser-specific options (e.g., ChromeOptions, FirefoxOptions).
5. **Actions**
   * Used to perform advanced user interactions like drag-and-drop, hover, or keyboard actions.
6. **Key**
   * Represents keyboard keys (e.g., ENTER, TAB, etc.) for simulating keypress events.

Commonly Used Methods

**WebDriver Methods**

* get(url: string): Promise<void>  
  Opens the specified URL in the browser.
* quit(): Promise<void>  
  Closes the browser and ends the session.
* findElement(locator: By): Promise<WebElement>  
  Finds a single element on the page.
* findElements(locator: By): Promise<WebElement[]>  
  Finds multiple elements matching the locator.
* getTitle(): Promise<string>  
  Retrieves the title of the current page.
* getCurrentUrl(): Promise<string>  
  Retrieves the current URL.

**WebElement Methods**

* click(): Promise<void>  
  Simulates a mouse click on the element.
* sendKeys(...keys: string[]): Promise<void>  
  Sends keystrokes to the element.
* getText(): Promise<string>  
  Retrieves the visible text of the element.
* getAttribute(attributeName: string): Promise<string | null>  
  Retrieves the value of a specified attribute.
* isDisplayed(): Promise<boolean>  
  Checks if the element is visible on the page.

**By Methods**

* By.id(id: string)  
  Locates an element by its ID.
* By.className(className: string)  
  Locates an element by its class name.
* By.css(selector: string)  
  Locates an element using a CSS selector.
* By.xpath(xpath: string)  
  Locates an element using an XPath expression.

Example Code in TypeScript

Here’s a simple example of using Selenium with TypeScript:

Copy the codeimport { Builder, By, WebDriver, WebElement } from "selenium-webdriver";

async function example() {

// Initialize WebDriver for Chrome

const driver: WebDriver = await new Builder().forBrowser("chrome").build();

try {

// Navigate to a website

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

// Find an element by its ID

const element: WebElement = await driver.findElement(By.id("example-id"));

// Interact with the element

await element.sendKeys("Hello, Selenium!");

await element.click();

// Get the page title

const title: string = await driver.getTitle();

console.log(`Page title is: ${title}`);

} finally {

// Quit the browser

await driver.quit();

}

}

example();

Git Commands:

git checkout master

--------This command switches your working directory to the master branch

git merge feature-branch