1. **Tell me about yourself**

Professional Summary:

I'm **Ashok Reddy**, a **QA and Automation Test Engineer** with **4 years of experience** in both **manual and automation testing**.

At **Tech Mahindra**, I worked on two major projects:

- **Cisco CCIE** – as an **Automation Engineer**, I:
    - Designed and maintained **Selenium WebDriver** test scripts
    - Used **TestNG** and **Maven** for test execution and management
- **Google Calendar** – as a **QA Engineer**, I:
    - Performed **manual functional, regression, and exploratory testing**
    - Logged and tracked bugs using **buganizer**
    - Collaborated with developers and product teams for quick issue resolution

Key Responsibilities Across Projects:

- Developed and executed automation scripts for web applications
- Actively participated in daily Scrum meetings, sprint planning, retrospectives, and other Agile ceremonies to discuss progress, blockers, and ensure alignment with the team
- Created and maintained test cases, test data, and test plans

2) **Have you worked on Framework implementation? If not explain your framework end to end execution flow.**

In our project, we used **Java with Selenium** for automation. Selenium supports many languages, but Java is widely used and preferred.

We built a **Hybrid Framework** using the **Page Object Model with Page Factory**.

🔹 **Page Object Model (POM)**:
Each web page has a separate class that contains locators and methods related to that page.

🔹 **Packages**:

`pages` package → contains all page classes

`tests` package → contains all test classes
Example: A login test uses methods from `HomePage` and `ArticlePage`.

🔹 **Maven Project Structure**:

Test classes → `src/test/java`

Other code (pages, utilities) → `src/main/java`

🔹 **Test Base Class**:

Initializes WebDriver

Loads properties from config file

Common setup like choosing the browser, launching URL

🔹 **Utility Class**:

Common reusable functions like:

- Taking screenshots
- Sending emails
- Waiting for elements

🔹 **Property File (`config.properties`)**:

Stores static data like browser type, URL, credentials

Easy to update without touching code

🔹 **Screenshots Folder**:

Saves failed test screenshots for debugging

🔹 **Test Data**:

Stored in Excel Handled using **Apache POI** for **Data-Driven Testing**

- **TestNG**:

Manages test execution (regression, smoke, etc.)

Supports parallel and group execution

- **Maven**:

Manages project dependencies

Build and run tests using `pom.xml`

- **Jenkins**:

Used for CI/CD

Triggers test execution automatically from `pom.xml`

3) Explain Java Main method 'public static void main(String args[])'?

The **`main` method** is the **entry point** of any Java program. When you run a Java class, this method gets called first by the JVM (Java Virtual Machine).

Part	Meaning
`public`	Access modifier – the method is **accessible from anywhere**, including JVM.
`static`	The method belongs to the **class**, not an instance. JVM can call it without creating an object.
`void`	The method returns **no value**.
`main`	The name of the method – it's **predefined** and recognized by JVM as the starting point.
`String args[]` or `String[] args`	Array of **command-line arguments** passed to the program.

Let's break down each part:

```
public static void main(String[] args)
```

◆ `public` : This is an access modifier

- It means the method is **accessible from anywhere**.
- JVM needs to access this method to start the program, so it must be public.

◆ `static` : This keyword

- It means **you don't need to create an object** to call this method.
- JVM can call it **without creating an object** of the class.

◆ `void : this is a return type`

- It means the method **does not return anything**.

◆ `main`

- This is the **name** of the method.
- JVM looks for this method as the **starting point**.

◆ `String[] args`: This represents an array of `String` objects

- This is an Array of **command-line arguments** passed to the program
- Example: If you run the program with `java MyClass hello world`, then `args[0] = "hello"`, `args[1] = "world"`.

4) What is Method overloading?

Method Overloading means defining multiple **methods with the same name** in the same class, but with different parameters (type, number, or order).

It is an example of Compile-time Polymorphism (also called Static Polymorphism).

Will Use Method Overloading

- To **perform similar operations** in **different** ways.
- Increases **readability** and **code reusability**.

Rules for Method Overloading:

1. Method name **must be the same**
2. Parameter list must be **different**

    - Different number of parameters
    - Different type of parameters
    - Different order of parameters

Ex:

```
public class OverloadDemo {

    // Overloaded method 1

    public void show(int a) {

        System.out.println("Integer: " + a);

    }

    // Overloaded method 2

    public void show(String s) {

        System.out.println("String: " + s);

    }

    // Overloaded method 3

    public void show(int a, int b) {

        System.out.println("Sum: " + (a + b));

    }

    public static void main(String[] args) {

        OverloadDemo obj = new OverloadDemo();

        obj.show(10);           // Calls show(int)          //Integer: 10

        obj.show("Ashok");       // Calls show(String)  // String: Ashok

        obj.show(5, 15);         // Calls show(int, int)    //Sum: 20

}}
```

5) What is Method overriding?

Method Overriding means **providing a new implementation of a method in a child class** that is already defined in its parent class.

It is an example of Runtime Polymorphism (or Dynamic Method Dispatch).

In simple words, **overriding means to override the functionality of an existing method**.

When to Use?

- When a subclass wants to **modify or extend the behavior** of a method inherited from the parent class.

Key Rules of Method Overriding:

1. Method name, return type, and parameters **must be exactly the same** as the parent method.
2. The method **must be inherited** from a superclass.
3. The method in the parent class **must not be `private`, `final`, or `static`**.

Ex:

```
class Animal {
    public void sound() {
        System.out.println("Animal makes a sound");
    }
}

class Dog extends Animal {
    @Override
    public void sound() {
        System.out.println("Dog barks");
    }
}

public class OverrideDemo {
    public static void main(String[] args) {
        Animal a = new Dog();  // Parent reference to child object
        a.sound();             // Calls overridden method in Dog        //Dog barks
    }
}
```

6) Difference between Abstract class and Interface?

Abstract Class	**Interface**
To declare an abstract class, we use the `abstract` keyword.	To declare an interface, we use the `interface` keyword.
The `abstract` keyword is **mandatory** to declare an abstract **method**.	The `abstract` keyword is **optional** — all methods are abstract by default (before Java 8).
Can contain both **abstract methods** and **concrete methods** (with a body).	Can contain only **abstract methods** (until Java 7). From Java 8+, can have `default` and `static` methods.
Provides **partial abstraction** (can have some implementation).	Provides **full abstraction** (until Java 8). Only method declarations, no implementation.
Can have **public** and **protected abstract** methods.	Can have only **public abstract** methods.
Can have **static**, **final**, or **static final** variables with any access modifier.	Can have only **public static final** variables (constants).
Can **extend one class or one abstract class** (single inheritance).	Can **extend multiple interfaces** (supports multiple inheritance).
Does not support multiple inheritance of classes.	**Supports** multiple inheritance via interfaces.

- Use **abstract** class when you **need some common code across subclasses.**
- Use **interface** when you just **want to define rules** (what to do), not how.

Abstract Class Example:

```
abstract class Animal
{
   void eat()
 {
       System.out.println("Eating...");
   }
   abstract void sound();
}
```

Interface Example:

```
interface Animal
 {
 void sound();
}
```

7) Difference between HashSet and HashMap?

HashSet	HashMap
Stores **only unique elements**.	Stores **key-value pairs**.
Implements the **Set** interface.	Implements the **Map** interface.
No duplicate elements allowed.	**Duplicate keys not allowed**, but **duplicate values allowed**.
Allows **only one null element**.	Allows **one null key** and **multiple null values**.
Stores **only values** (no key).	Stores **both keys and values**.
Used when you need a **collection of unique items**.	Used when you need to **associate keys with values**.
Internally uses a **HashMap** to store data.	Internally uses **hash buckets** for storage.
Example: `HashSet<String> set = new HashSet<>();`	Example: `HashMap<Integer, String> map = new HashMap<>();`

Hash set example:

```
import java.util.HashSet;

HashSet<String> fruits = new HashSet<>();
fruits.add("Apple");
fruits.add("Banana");
fruits.add("Apple");  // Duplicate — ignored

System.out.println(fruits); // Output: [Apple, Banana]
```

Hashmap Example:

```
import java.util.HashMap;

HashMap<Integer, String> studentMap = new HashMap<>();
studentMap.put(1, "Kiran");
studentMap.put(2, "Ravi");
studentMap.put(1, "Ashok");  // Key 1 updated with new value

System.out.println(studentMap); // Output: {1=Ashok, 2=Ravi}
```

8) Write a Java Program to generate Output "aabbbcccc" with the input "a2b3c4"?

```
public class ConvertNumericToCharExample
{
        public static void main(String[] args)
        {
                String str = "a2b3c4";        / /output = aabbbcccc
                convertNumToChar(str);
        }
        public static void convertNumToChar(String s)
        {
                for(int i =0; i < s.length(); i++)
                {
                        if(Character.isAlphabetic(s.charAt(i)))
                        {
                                System.out.print(s.charAt(i));
                        }
                        else
                        {
                                int a = Character.getNumericValue(s.charAt(i));
                                for(int j =1; j <a; j++)
                                {
                                        System.out.print(s.charAt(i-1));
                                }
                        }
                }
        }

}
```

9) What is a static block?

A static block is a code block **that's executed when a class is loaded into memory**,before the main method
It's used **to initialize static variables** or perform other **one-time setup tasks**.

- Declared using the `static` keyword.
- Executes **only once** per class loading.

Ex:

```
Static
 {
      System.out.println("Static block executed");
      number = 100;
  }
```

10) Explain the compareTo() method?

The `compareTo()` method is part of the **Comparable interface** in Java.
It is used to **compare two objects** (mostly Strings or user-defined objects) **for natural ordering** (like alphabetical or numerical order).

`compareTo()` is used to define the **natural ordering** of objects (like strings or custom classes) by returning **0**, **positive**, or **negative** integers.

Return Value	Meaning
`0`	This object is **equal to** the specified one
`< 0` (negative)	This object is **less than** the specified one
`> 0` (positive)	This object is **greater than** the specified one

```
public class CompareExample
{
   public static void main(String[] args) {
      String s1 = "apple";
      String s2 = "banana";
      String s3 = "apple";

      System.out.println(s1.compareTo(s2));  // Output: -1 (apple < banana)
      System.out.println(s1.compareTo(s3));  // Output: 0  (apple == apple)
      System.out.println(s2.compareTo(s1));  // Output: 1  (banana > apple)
   }
}
```

11) How do you manage a frame in Selenium WebDriver?

The iframe is an element of HTML **that puts another webpage within the parent page**.
Ways to switch:

- **By Index**:driver.**switchTo**().frame(0); // Switches to the first frame on the page
- **By Name or ID**:driver.switchTo().frame("frameName"); // If frame has a name or ID
- **By WebElement:**

```
WebElement frameElement =driver.findElement(By.xpath("//iframe[@title='example']"));
        driver.switchTo().frame(frameElement);
```

12) What is a Page Factory in Selenium?

Page Factory is a built-in class in Selenium that makes it easier to **create Page Objects**

It helps in **initializing web elements** defined in a Page Object class using the `@FindBy` annotation.

Makes code more **readable** and **maintainable**
Ex:
@FindBy(id="username")
WebElement usernameField;

PageFactory.**initElements**() -Used to initialize all `@FindBy` elements in the class

13) How would you handle dynamic elements that change attributes or IDs on each page load?

Strategies to Handle Dynamic Elements:
1.Use XPath with **`contains()`** - `//input[contains(@id, 'username')]`
2.Use XPath with **`starts-with()`** or **`ends-with()`** - `//div[starts-with(@id, 'user_')]`
3.Use CSS Selectors with partial match - **`input[id*='email']`**
4.Wait for Element Visibility (Explicit Wait)

14) What is the difference between findElement and findElements in Selenium?

`findElement()`	`findElements()`
Returns a single `WebElement`	Returns a `List<WebElement>`
Throws `NoSuchElementException` if not found	Returns an empty list if no elements found
Returns only the **first matching** element	Returns **all matching** elements
Used when only **one element is expected**	Used when **multiple elements** are expected

```
// Login button - only one expected
WebElement loginBtn = driver.findElement(By.id("loginBtn"));
loginBtn.click();

// All links in footer
List<WebElement> footerLinks =
driver.findElements(By.cssSelector(".footer a"));
```

```
for (WebElement link : footerLinks) {
    System.out.println(link.getText());
}
```

15) How do you upload a file using Selenium WebDriver?

In Selenium, you can upload a file using the `sendKeys()` method on an `<input type="file">` element

Ex:

```
driver.findElement(By.id("myFile")).sendKeys("C:\\Users\\..Path");
```

two additional methods using the AutoIT tool and Robot Class

- **Upload Files using AutoIT**

AutoIT is a Windows automation tool designed to handle **OS-level pop-ups** like file upload dialogs, which Selenium alone cannot manage.

Ex:

```
driver.findElement(By.id("uploadButton")).click(); // Opens Windows
Runtime.getRuntime().exec("C:\\Path\\To\\upload.exe"); //Executes
AutoIT script
```

- **Upload Files using Robot Class**

Robot Class is a Java class that simulates **keyboard and mouse** actions at the OS level.

Platform-independent (works on Windows, macOS, Linux).

Ex:

```
driver.findElement(By.id("uploadBtn")).click();

UploadWithRobot.uploadFile("C:\\Users\\Ashok\\Resume.pdf");
```

16) What is Robot class?

In Selenium, the **Robot class** is a Java-based utility from the `java.awt` package that is **used to automate keyboard and mouse actions** at the **OS level**.

The Robot class in Selenium is used when Selenium WebDriver alone cannot handle tasks such as interacting with native OS-level components like file **upload dialogs, print popups, or keyboard/mouse** events

Important Methods of Robot Class:

Method	**Description**
`keyPress(int keycode)`	Presses a key
`keyRelease(int keycode)`	Releases a key
`mouseMove(int x, int y)`	Moves mouse pointer to x, y location
`mousePress(int buttons)`	Simulates mouse button press
`mouseRelease(int buttons)`	Simulates mouse button release
`delay(int ms)`	Adds delay in milliseconds

17) What are the advantages of using WebDriver Waits?

WebDriver Waits are **useful in handling synchronization issues** in Selenium.
They **improve the reliability** of tests by waiting dynamically for certain conditions like element visibility or clickability, especially for dynamic web applications.
I **prefer using Explicit Wait over Thread**.sleep because it waits only as long as needed

18) **Explain the concepts of the FluentWait class in selenium and how you would implement it?**

FluentWait is an advanced version of WebDriverWait, allowing custom polling intervals and ignoring specific exceptions.

It's useful for handling scenarios where elements might appear/disappear over time.
To implement it, create a FluentWait instance, set up its conditions and polling intervals, and then use it to wait for an expected condition to be satisfied.

```
// FluentWait setup
Wait<WebDriver> wait = new FluentWait<>(driver)
                      .withTimeout(Duration.ofSeconds(30))
                      .pollingEvery(Duration.ofSeconds(5))
                      .ignoring(NoSuchElementException.class);
```

19) How to execute JavaScript code in Selenium?

JavaScript code can be executed in Selenium using **JavaScriptExecuter**
This is **useful when Selenium commands don't work** as expected
Ex: Clicking hidden elements,Getting inner text,Handling complex UI elements

```
WebDriver driver = new FirefoxDriver();

if (driver instanceof JavascriptExecutor)
{
  JavascriptExecutor js = (JavascriptExecutor) driver;
  js.executeScript("JavaScript code here");
}
```

20) What are some advantages of TestNG?

- **Annotations Support**:Provides powerful and flexible annotations like @Test,@BeforeMethod
- **Parallel Execution**:Allows running tests in multiple threads for faster execution.
- **Grouping of Tests**:you can group tests using tags (e.g., smoke, regression, sanity).
- **Prioritization of Tests**:You can define the order of test execution using the priority attribute.
- **Data-Driven Testing:**Supports @DataProvider to run the same test with different sets of data.
- **HTML Reports Generation**:Automatically generates HTML reports after execution

21) What is the use of @Listener annotation in TestNG?

@Listeners in TestNG are used to **track test execution events** like **test start, pass, fail, skip**, etc

Usually, testNG listeners are **used for configuring reports and logging**.

One of the **most widely used listeners** in testNG is the **ITestListener** interface.
It has methods like **onTestSuccess**, **onTestFailure**, **onTestSkipped**, etc.
We need to implement this interface by creating a listener class of our own.
After that using the @Listener annotation, we can specify that for a particular test class, a customized listener class should be used.

Use of Listeners:
-Logging test results
-Taking screenshots on failure
-Generating custom reports
Ex:

```
@Listeners(PackageName.CustomizedListenerClassName.class)
public class TestClass
{
WebDriver driver= new FirefoxDriver();
 @Test public void testMethod()
{
//test logic
}
}
```

22) What is the role of the WebDriverManager library in Selenium automation, and why is it useful?

WebDriverManager is a library that **automates the download and setup of WebDriver binaries** for different browsers and versions.
It simplifies WebDriver setup and ensures that the correct driver version is used, reducing compatibility issues and maintenance efforts.
Ex:

```
WebDriverManager.chromedriver().setup();
WebDriver driver = new ChromeDriver();
```

Why It's Useful (Advantages):
- Eliminates manual driver setup
- Auto-matches driver version
- Supports multiple browsers
- **Reduces boilerplate code:we don't need to use** System.setProperty(...).
- CI/CD friendly

23) Can you explain the difference between smoke testing and sanity testing?

Smoke testing:
Smoke testing is performed **after software release** to ensure that the critical functionalities are working correctly.
The purpose of smoke testing is to verify the important features are working as intended.

Sanity testing:
Sanity testing was performed **after receiving a software build** with **minor changes** in code or functionality.
The purpose of sanity testing is **to verify that the bugs are fixed and no further issues** observed due to these changes.

Compared to smoke testing,sanity testing hasscope. Specifically targets areas affected by recent changes.

24) What is the difference between severity and priority?

Severity:
Severity refers to **how a defect impacts the application** or functionality. It is usually decided by the **tester**.
High - Critical functionality is broken
Low - Minor issue, cosmetic or UI problem

Priority:
Priority refers to **how soon a defect should be fixed**. It is usually decided by the **developer or project manager** based on business needs.
High - Needs to be fixed immediately
Low - Can be fixed later

- Ex for "**low Severity & high Priority":**

Spelling mistake on homepage (e.g., "Welcom" instead of "Welcome")

- Ex for "**high Severity & low Priority":**

"Forgot Password" feature is completely broken,But this issue exists only on the **Admin Panel**, which is used **only once a year** for maintenance purposes

25) What is the defect life cycle?
Defect Life Cycle (also known as **Bug Life Cycle**) is the **journey** or **process that a defect goes through** from its initial discovery to its final closure in a software development life cycle.

Defect Life Cycle States:
New: Any recently located bugs are given a “new” status
Assigned: The new defect is assigned to a development team for correction
Open/in progress: Developer analyzes and starts working on the defect.
Fixed: All changes to the defect are completed
Pending retest: The developer reassigns the defect back to the tester
Verified Closed: With no additional flaws, the test cycle is verified and closed
Reopened: If the issue still exists after the fix, the tester reopens the defect.
Deferred: The **fix is postponed to a future release**. Business priority is low.
Rejected: Developer does not consider it a defect (e.g., expected behavior).

26) What are the key elements when reporting a bug?
Reporting a bug effectively is crucial for developers to **reproduce, understand, and fix** the issue quickly. A good bug report should be clear, concise, and complete.
Key elements:

- Bug ID
- Title
- Summary/Description
- **Steps to Reproduce**
- Actual Result
- Expected Result
- **Severity/Priority**
- Environment
- Attachments
- Assignee (optional)
- Build information & component

27) How do you handle multiple windowstabs in selenium webdriver?
In Selenium WebDriver, **handling multiple windows or tabs** is done using **window handles**. Each browser window or tab has a unique identifier known as a **Window Handle**.
Ex:

```
String mainWindow = driver.getWindowHandle();
Set<String> allWindows = driver.getWindowHandles();
for (String window : allWindows)
 {
            if (!window.equals(mainWindow))
          {
                driver.switchTo().window(window);
         }
}
 driver.switchTo().window(mainWindow);
```

28) Do you work in cucumber, can you tell me what all files are required in cucumber?
In cucumber we have Feature file, Step Definition file and Test Runner file.
- In the **feature file** we used to write a scenario in gherkin language which is most like in plain English language.
    - Here we use some of the keywords like feature,scenario, scenario outline, given, when, then, and, example, background keywords for writing our test scenarios steps.
- In the **Step Definition file** we write mapping code for all the scenarios of the feature file.

-The **test Runner file** provides the **address of the feature file**,step definition file, and all-important Tags, Plugin, Listeners in that. **@CucumberOptions**
-**Hooks File** Contains setup and teardown methods like @Before, @After. Can be used to **open/close browsers**.

29) How can we fetch a text written over an element?
Using the **getText**() method we can fetch the text over an element.
String text = driver.findElement("elementLocator").**getText**();

30) What are some expected conditions that can be used in explicit waits?
Some of the commonly used expected conditions of an element that can be used with explicit waits are-

-**elementToBeClickable**(By locator):Waits until the element is **clickable**

-**elementToBeSelected(By locator**): Waits until the element (checkbox/radio) is **selected**

-**visibilityOfElementLocated**(By locator):Waits until the element is **visible**

-**invisibilityOfElementLocated(By locator)**:Waits until the element is **no longer visible** or removed from DOM.

-**alertIsPresent():**Waits until an alert box is present.

-**textToBePresentInElementLocated**:Waits until the given text is present in the specified element

-**titleContains(String title):** Waits until the page title **contains** the given string.

-**titleIs(String title**): Waits until the **page title** is exactly the given string.

31) What are the different mouse actions that can be performed?

In Selenium, we use the **Actions class** to perform advance mouse & keyboard operations

Method	**Description**
`click()`	Performs a single left mouse click.
`doubleClick()`	Performs a double-click on an element
`contextClick()`	Performs a **right-click** on the target
`moveToElement(WebElement eleme`	Moves the mouse to the center of the element (hover).
`dragAndDrop(source, target)`	Clicks and holds the source element, the target, and releases.
`clickAndHold()`	Clicks (without releasing) on the cur location or a specific element.
`release()`	Releases the pressed mouse button.

32) How to handle dropdowns in selenium?

In Selenium, **dropdowns** (i.e., `<select>` HTML tags) are handled using the **Select class**

```
WebElement dropdown = driver.findElement(By.id("dropdownId"));

Select Se= new Select(dropdown);

Se.selectByVisibleText("SomethingWritten");
Se.selectByValue("opt1");
Se.selectByIndex(2);
```

*.Where do you used OOP's concept in your selenium project:

Inheritance: Where - In BaseTest / BasePage classes
Created BaseTest and BasePage classes so child classes reuse setup, teardown, and utilities

Polymorphism: Where - Method overloading & overriding in framework utilities.
Used method overloading for **waits** and overriding for **click** methods (like normal click vs JS click).
WebDriver(based upon the value we provided it will act as chromedriver,firefoxdriver,IEDriver..)

Encapsulation:Where -In Page Object Model (POM) classes.
In Page Object Model, I kept locators private and exposed only public action methods like login()

Abstraction: Where -Abstract classes and interfaces for framework design.
Used interfaces and abstract classes in driver factory to easily switch between browsers.This made framework reusable&easy to maintain

1.Inheritance Ex: - In BaseTest / BasePage classes.
Common code (browser setup, teardown, waits, logging, reporting) is kept in parent classes.
Test classes extend BaseTest and Page classes extend BasePage

```
public class BaseTest {
    protected WebDriver driver;

    @BeforeMethod
    public void setup() {
        WebDriverManager.chromedriver().setup();
        driver = new ChromeDriver();
        driver.manage().window().maximize(); }

    @AfterMethod
    public void tearDown() {
        driver.quit();
    }}

public class LoginTest extends BaseTest {
    @Test
    public void testLogin() {
        LoginPage loginPage = new LoginPage(driver);
        loginPage.enterUsername("admin");
        loginPage.enterPassword("password");
        loginPage.clickLogin();
    }
}
```
2.Polymorphism Ex: - Method overloading & overriding in framework utilities.
Overloading: Handling different types of waits or element interactions.
Overriding: Implementing custom click() or sendKeys() methods in a reusable BasePage.
WebDriver(based upon the value we provided it will act as chromedriver,firefoxdriver,IEDriver..)

Overloading ex code:
```
public class WaitHelper {
    public void waitForElement(WebElement element) {
        new WebDriverWait(driver, Duration.ofSeconds(10))
            .until(ExpectedConditions.visibilityOf(element));
    }

    public void waitForElement(By locator) {
        new WebDriverWait(driver, Duration.ofSeconds(10))
            .until(ExpectedConditions.visibilityOfElementLocated(locator));
    }
}
```

Overriding ex code:
```
class BasePage {
    WebDriver driver;
    public void clickElement(WebElement element) {
        element.click();
    }
}

class JavaScriptPage extends BasePage {
    @Override
    public void clickElement(WebElement element) {
        ((JavascriptExecutor)driver).executeScript("arguments[0].click();", element);
    }
}
```

3.Encapsulation ex: - In Page Object Model (POM) classes.

We make web elements private and provide public getter methods (or action methods like login(), clickButton()).

This hides implementation details and exposes only necessary functionality.

```
public class LoginPage {
    private WebDriver driver;

    // Encapsulated locators
    private By username = By.id("username");
    private By password = By.id("password");
    private By loginBtn = By.id("login");

    public LoginPage(WebDriver driver) {
        this.driver = driver;
    }

    // Encapsulated action methods
    public void enterUsername(String user) {
        driver.findElement(username).sendKeys(user);
    }
    public void enterPassword(String pass) {
        driver.findElement(password).sendKeys(pass);
    }
    public void clickLogin() {
        driver.findElement(loginBtn).click();
    }
}
```

4.Abstraction Ex: - Abstract classes and interfaces for framework design.
Interfaces (ITestDataReader, IDriverManager) to define contracts.
Abstract classes for reusable test utilities or drivers

```
public interface DriverManager {
    WebDriver getDriver();
}

public class ChromeDriverManager implements DriverManager {
    public WebDriver getDriver() {
        WebDriverManager.chromedriver().setup();
        return new ChromeDriver();
    }}
public class FirefoxDriverManager implements DriverManager {
    public WebDriver getDriver() {
```

```
    WebDriverManager.firefoxdriver().setup();
    return new FirefoxDriver();
}}
```

Where do you used OOPs in your project

OOPS: Object oriented programming language:

Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules.

1. ***Inheritance***   -acquiring the properties of super class to sub class is called as Inheritance(achieved by "extends" keyword)

Created **BaseTest** and BasePage classes so child classes reuse setup, teardown, and utilities

Real time:Think of a "**Car**" class. A "**SportsCar**" or "**ElectricCar**" can inherit basic properties like **wheels**, **engine**, and methods like start/stop from the generic Car, while also adding their specific features like turbo mode or battery

2. ***polymorphism***   - One entity showing different behavior is called as polymorphism

WebDriver(based upon the value we provided it will act as chromedriver,firefoxdriver,IEDriver..)**Real time EX**: human relations

3. ***Encapsulation*** - The process of wrapping the data members(Global variables) and memberfunction(methods) into a single unit(class) is called Encapsulation.

all the page object models are examples of Encapsulation(every class is also enc example)

Real time: Think of a capsule—it hides the medicine inside. Similarly, a class hides the data and provides controlled access using methods (getters/setters).

4. ***Abstraction***   - process of **hiding the internal implementation** and showing the necessary data to the end user

page object model page ,we can **see the locator name** but not the initialization/implementation of locators

Real time:Using a TV remote, you don't need to know how it works internally,just to operate it

*What is a Constructor& its uses in Java?

Constructor is a special type of method which **gets executed whenever we created an object**

The main purpose of constructor is to **initialise a non static variable**

Types:

Default Constructor – Provided by Java if you don't define one.

Parameterized Constructor – Allows passing values at the time of object creation.

*.Difference between throw and throws?

throw: is used to **explicitly** throw an exception.

throws: is used in method signature to **declare** exceptions.

Throw:

• Used to **create and throw** an exception object **explicitly**.

• Used **inside a method or block**.

• Syntax:`throw new ExceptionName("Message");`

• Mainly used for **user-defined exceptions** (runtime).

• Using `throw`, we can **throw only one exception** at a time.

throws:

• throws keyword is used to **declare or report a checked Exception.**

• throws keyword is used **with method declaration.**

• **Syntax:**`public void methodName() throws Exception1, Exception2 { }`

• Mainly used for **checked exceptions**.

• Using `throws`, we can **declare multiple exceptions** at a time.

When we use throws keyword, we are indicating that current method will not handle exception rather calling method or its caller will handle
the exception

*: **What's the difference between for, while, and do-while loops?**

• **for**: Used when the number of iterations is known.

• **while**: Used when the **condition must be checked** before executing.

• **do-while**: Executes at least once, then checks the condition

*. **What is the difference between break,continue and return?**

a) **break**: Exits the loop or switch immediately.

b) **continue**:Skips current **iteration** and goes to next.
c) **return**: It exits the method and optionally returns a value.

*. **What are the different types of waits in Selenium?**

1.Implicit Wait – Wait for a set amount of time for all elements.
Ex:
```
driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(10));
```

2.Explicit Wait – Wait for a specific condition to be met.
Ex:
```
WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
WebElement element =
wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("username")));
```

3.Fluent Wait – FluentWait is an advanced version of WebDriverWait, allowing custom polling intervals and ignoring specific exceptions.
Ex:
```
        Wait<WebDriver> wait = new FluentWait<>(driver)
                            .withTimeout(Duration.ofSeconds(30))
                            .pollingEvery(Duration.ofSeconds(5))
                            .ignoring(NoSuchElementException.class);
```

*.**How do you take a screenshot in Selenium WebDriver?**

```
TakesScreenshot ts = (TakesScreenshot) driver;
File src = ts.getScreenshotAs(OutputType.FILE);
FileUtils.copyFile(src, new File("path/to/screenshot.png"));
```

***. How do you take a screenshot on test failure in Cucumber?**

```
@After
public void takeScreenshot(Scenario scenario) {
    if (scenario.isFailed())
   {
     TakesScreenshot ts = (TakesScreenshot) driver;
     byte[] screenshot = ts .getScreenshotAs(OutputType.BYTES);
        scenario.attach(screenshot, "image/png", "failed_scenario");
    }
}
```

Exceptions in Selenium Java:

1️⃣ **NoSuchElementException:**
- Trying to locate an element that does not exist in the DOM.
Fix: Check locator (XPath, CSS, ID), wait until element is present.

2️⃣ **TimeoutException:**
- Element not found within the given **explicit wait** time.
Fix: Increase wait time or improve locator strategy.

3️⃣ **StaleElementReferenceException:**
Element reference is lost (page refreshed / DOM updated).
Fix: Re-locate element after DOM refresh.

4️⃣ **ElementNotInteractableException:**
- Trying to click or sendKeys on an element which is **not visible / disabled**.
Fix: Wait for element to be clickable using ExpectedConditions.

5️⃣ **ElementClickInterceptedException:**
- Another element (like popup/banner) is blocking the element you are trying to click.
Fix: Scroll into view or use JavascriptExecutor click.

6️⃣ **InvalidSelectorException:**
- Using invalid XPath / CSS selector.(broken xpath)

7️⃣ **NoSuchWindowException & 8️⃣ NoSuchFrameException:**
- Switching to a window handle or Frame that doesn’t exist.

9️⃣ **NoAlertPresentException:**
- Trying to switch to an alert which is not present.

🔟 **JavascriptException:**
- Executing invalid JavaScript via JavascriptExecutor.

1️⃣1️⃣ **WebDriverException:**
- Generic Selenium error, e.g. driver version mismatch with browser.

1️⃣2️⃣ **MoveTargetOutOfBoundsException:**
- Trying to move to an element that is outside the page view.

Cucumber Exceptions in Java:

1️⃣ **AmbiguousStepDefinitionsException:**
- Two or more step definitions are mapped to the same Gherkin step.
Fix: Keep only one unique step definition per step.

2️⃣ **UndefinedStepException:**
- You wrote a step in the **feature file** but forgot to create a **matching step definition**
Fix: Create matching step definition.

3️⃣ **PendingException:**
- Step definition exists, but you marked it as **pending** or left it incomplete.
Fix: Implement actual code instead of throwing PendingException.

4️⃣ **DuplicateStepDefinitionException:**
- When two identical step definitions are present **within the same project**.
Similar to ambiguous, but here cucumber clearly identifies exact duplicates.
Fix: Remove duplicate mappings.

5️⃣ **CucumberException:**
- Generic exception for **syntax errors in feature files** (e.g., missing Scenario or Feature keyword,spelling mistakes..).
Fix: Correct the Gherkin syntax.

6️⃣ **ClassNotFoundException / NoClassDefFoundError:**
- Missing cucumber-java / cucumber-core /cucumber-testing jars or mismatched versions.
Fix: Check pom.xml dependencies.

7️⃣ **IllegalArgumentException:**
- Your step definition regex or parameter type **doesn't match** the step in feature file.
Fix: Change to {int} instead of {String} or update feature file.

8️⃣ **NullPointerException:**
- Driver object is not initialized in step definition / Hooks.
Fix: Initialize driver in @Before hook or constructor.

*. Checked vs Unchecked Exceptions in Java

1️⃣ **Checked Exceptions:**
- Exceptions that are **checked at compile-time** by the compiler.
- You must **handle** them using **try-catch** or declare with **throws** keyword.
- They represent conditions that are **outside program control** (like File not found, network errors).
Ex:
IOException,SQLException,FileNotFoundException,InterruptedException
In Selenium:

- Thread.sleep() → Checked Exception.

2️⃣ **Unchecked Exceptions:**
- Exceptions that occur at **runtime only** (not checked at compile time).
- They extend **RuntimeException**.
- Usually due to programming mistakes (like invalid index, null object).
Ex:
NullPointerException,ArrayIndexOutOfBoundsException,
ArithmeticException (like divide by zero),NumberFormatException
In Selenium:

- NoSuchElementException → Unchecked Exception

*.How to Read Data from Excel in Selenium (Java)

We use **Apache POI** library to read/write Excel files (.xlsx or .xls).

```
  FileInputStream fis = new FileInputStream(filePath);
      Workbook workbook = WorkbookFactory.create(fis);

 1.Sheet sheet = workbook.getSheet(sheetName);
      Row row = sheet.getRow(rowNum);
      Cell cell = row.getCell(colNum);
      cellData = cell.toString();
      workbook.close();
         (or)
2.workbook.getSheet(sheetName).getRow(rowNum).getCell(colNum).toString();
```

***.What challenges did you face in Selenium automation?**

Dynamic IDs: Resolved using regex-based XPath.

Slow loading pages: Handled using FluentWait.

Multiple popups: Managed with WindowHandles & conditional logic.

Parallel execution issues: Solved using ThreadLocal drivers.

***. Gherkin key words:**

Keyword	Use / Purpose
`Feature:`	Describes the overall test/business goal
`Scenario:`	Describes one test case or flow
`Given`	Sets up preconditions (e.g., open browser)
`When`	Describes the main action (e.g., click, login)
`Then`	Verifies outcomes (e.g., check title, success)
`And`	Used to combine multiple `Given/When/Then`
`Background:`	Runs common steps before each scenario
`Scenario Outline:`	Used for data-driven testing (with Examples)
`Examples:`	Provides data table for Scenario Outline

Runners.class:

```
@CucumberOptions(

   features = "src/test/resources/features",  // path to .feature files

   glue = "stepdefinitions",             //path to step definition classes

   tags ="@visainivite",                    //run specific scenarios

   plugin = {"pretty", "html:target/cucumber-reports.html"},

//Console output in clean, readable format & Generates HTML report at given
path

   monochrome = true         //Removes weird characters from console output

   dryRun = true

//checks if every step in the .feature file has a matching step definition
without running the test.

)
```

***.How to Rerun Failed Scenarios in Cucumber?**

In real-time projects, some test scenarios may fail due to network issues, environment slowness, or minor glitches. Instead of running the whole suite again, we **rerun only failed scenarios**.

Step 1: Add rerun Plugin in Runner

```
@CucumberOptions(

plugin = {

        "pretty",

        "html:target/cucumber-reports.html",

        "rerun:target/failed_scenarios.txt"   // Save failed scenarios in
text file
```

Step 2: Create a Separate Rerun Runner

```
@CucumberOptions(features = "@target/failed_scenarios.txt",  //rerun failed
scenarios

public class RerunTestRunner extends AbstractTestNGCucumberTests {}
```

***. Data Table:**

Data Table is used to **pass a set of values** (rows/columns) to a **single step**.

```
Feature: Login feature

  Scenario: Login with valid credentials

    Given user enters login details

      | username | password  |

      | ashok    | ashok@123 |
```

***.Scenario Outline:**

Scenario Outline is used to execute the **same scenario multiple times** with different sets of input data.

data is passed using the **Examples** section.

```
Feature: Login feature

  Scenario Outline: Login with different users

    Given user logs in with "<username>" and "<password>"

    Then login should be successful

    Examples:

      | username | password  |

      | ashok    | ashok@123 |

      | reddy    | reddy@456 |
```

***. Define error bug Failure?**

Error : is a human mistake mostly committed by dev in developing stage

Bug/defect :when actual behavior doesn't matched with the expected behavior (we can find this in testing environment)

Failure : if something is not working as expected in customer environment is called failure (it is related to the UAT)

✅ String vs StringBuffer?

String:

- Immutable (cannot be changed once created)
- Not synchronized (not thread-safe)
- **Slower for modifications** (creates new object every time)
- Used when data is fixed (e.g., constants, messages)
- Methods like `concat()`, `substring()`, `replace()`
- `java.lang.String` package

StringBuffer:

- Mutable (can be modified without creating new object)
- Synchronized (thread-safe, slower in multithreading)
- **Faster for modifications** (modifies same object)
- Used when frequent modifications are required
- Methods like `append()`, `insert()`, `delete()`, `reverse()`
- `java.lang.StringBuffer` package

*. What is headless browser testing in Selenium?

Headless browser testing means running your Selenium test scripts **without opening a visible browser window** (UI).

The tests still run in the background, but you don't see the browser on the screen.

```
ChromeOptions options = new ChromeOptions();

options.addArguments("--headless");

WebDriver driver = new ChromeDriver(options);
```

***.What are the challenges you face in Selenium automation?**

1. Dynamic Web Elements

- **Challenge:** Many web applications had elements with **dynamic IDs or attributes** (changing every refresh).

- **Solution:** Used **robust locators** like XPath with `contains()`, `starts-with()`, `text()`, and **Relative Locators** (Selenium 4). In POM, I created reusable locator strategies.

✅ **2. Synchronization Issues (Timing Problems)**

- **Challenge:** Elements sometimes loaded late → `ElementNotVisibleException`.

- **Solution:** Used **Explicit Waits (WebDriverWait)** instead of Thread.sleep(). Created a **Reusable Wait Utility** method to handle multiple conditions (`elementToBeClickable`, `visibilityOf`).

✅ **3. Cross-Browser Compatibility**

- **Challenge:** Scripts worked in Chrome but failed in Firefox/Edge.

- **Solution:** Used **WebDriverManager** to manage drivers and executed tests in **parallel with TestNG** on multiple browsers. Reported browser-specific defects to dev team.

✅ **4. Handling Popups & Windows**

- **Challenge:** Application had multiple **alerts, file uploads, and child windows**.

- **Solution:** Used `driver.switchTo().alert()`, `driver.switchTo().window()`, and for file upload → used **Robotclass / AutoIT / sendKeys()** depending on the situation.

✅ **5. Test Data Management**

- **Challenge:** Hardcoding test data made scripts less reusable.

- **Solution:** Implemented **Data-Driven Testing** with Excel (Apache POI) and JSON. Later, integrated **Cucumber DataTables & Scenario Outline** to handle test data neatly.

✅ **6. Reports & Debugging Failures**

- **Challenge:** No clear visibility of failed scenarios.
- **Solution:** Integrated **Extent Reports + Cucumber Reports** → added screenshots on failure using `scenario.attach()` in Hooks. This helped devs debug quickly.

✅ **7. CI/CD Integration**

- **Challenge:** Manual execution was time-consuming.
- **Solution:** Integrated the Selenium suite with **Jenkins pipeline** so tests ran automatically on every build → faster feedback to the team.

*. *What is the difference between Assert and Verify?*

Assert- it is used to verify the result. If the test case fails then it will stop the execution of the test case there itself and move the control to another test case.

Verify- it is also used to verify the result. If the test case fails then it will not stop the execution of that test case.

*. Explain Selenium Architecture (Complete Suite):

Selenium has four components: IDE, RC (deprecated), WebDriver, and Grid.

1. **Selenium IDE (Integrated Development Environment):**

A **record-and-playback tool**, primarily a browser extension, that allows users to record interactions with a web application and then replay them as automated tests. It simplifies test creation for less technical users.

Example: If you write code in **Java**, you use **Selenium Java Client Library**.

2. Selenium RC (Remote Control):

Deprecated in Selenium 3 and removed in Selenium 4,RC is replaced by WebDriver.

3. Selenium WebDriver (Core Component):

- The **most widely used component**.
- Directly interacts with the browser without needing an intermediate server.
- Uses **Browser Drivers (ChromeDriver, GeckoDriver, EdgeDriver, etc.)**.
- Supports **multiple programming languages** (Java, Python, C#, etc.).
- Can be integrated with **Cucumber, TestNG, JUnit, Jenkins, Maven, Docker**.

4. Selenium Grid

- Used for **distributed test execution**.
- Allows running tests on **multiple machines, browsers, and OS in parallel**.
  Architecture:
  - **Hub** ⟶ Central server that controls test execution.
  - **Nodes** ⟶ Machines where actual tests run (different browsers/OS).

✅ Explain Selenium WebDriver Architecture:

Selenium WebDriver follows a **3-tier architecture**:

1. Selenium Client Libraries

- These are language-specific bindings (APIs) provided by Selenium.
- Available for **Java, Python, C#, Ruby, JavaScript, Kotlin, etc.**

2. JSON Wire Protocol / W3C Protocol

- Acts as a **communication medium** between client libraries and browser drivers.
- Converts Selenium commands (Java/Python code) into **JSON format**.

3. Browser Drivers

- Each browser has its own **driver** (binary) that understands Selenium commands:
  chromedriver.exe ⟶ for Chrome,**geckodriver.exe** ⟶ for Firefox,**msedgedriver.exe** ⟶ for Edge
- Drivers receive commands from JSON Wire/W3C and **translate them into native browser actions**.

4. Browsers: The actual browsers (Chrome, Firefox, Safari, Edge, etc.) where tests execute

Client Round (Project Allocation) Interview Questions & Answers

---

Project & Framework Related (Core Selenium)

1. **Q: Can you explain your automation framework?**
   A: *I worked on a Hybrid BDD framework using Selenium WebDriver with Java, Cucumber, TestNG, Maven, and Jenkins. It had reusable utility classes, Page Object Model for maintainability, Cucumber feature files for scenarios, and Extent/Allure reports for reporting. We integrated it with Jenkins for CI/CD.*

2. **Q: Why did you choose BDD (Cucumber) for your project?**
   A: *BDD improves collaboration between QA, Dev, and Business teams. Feature files are readable in Gherkin syntax so even non-technical stakeholders can understand scenarios. It bridges the gap between business requirements and automation.*

3. **Q: What are the key components of your framework?**
   A: *Page Object Model, Utilities (Excel, DB, API), Cucumber Feature files, Step Definitions, Hooks, TestNG runner, WebDriverManager for browser setup, Extent reports for reporting, Log4j for logging.*

4. **Q: How do you handle test data in your project?**
   A: *We used Excel files for test data with Apache POI and also parameterized values with Cucumber* `Scenario Outline` *+* `Examples`*. For dynamic test data, we generated data using Java Faker library.*

5. **Q: How do you manage locators in your framework?**
   A: *All locators are kept in Page Object classes (POM). Each page class has WebElements with @FindBy annotations (PageFactory). This improves reusability and reduces duplication.*

Execution & Reporting

6. **Q: How do you run your test cases?**
   A: *We use TestNG runner and Maven command (*`mvn test`*). For CI/CD we use Jenkins pipeline to trigger tests on every code commit.*

7. **Q: How do you run tests in parallel?**
   A: *We configured TestNG `parallel=methods` and also used Cucumber parallel plugin (`cucumber-jvm-parallel-plugin`). This reduced execution time significantly.*

8. **Q: How do you generate reports?**
   A: *We used Extent Reports for HTML reports, Cucumber JSON for detailed scenario reports, and Jenkins also published reports after execution.*

9. **Q: How do you capture screenshots?**
   A: *We used `TakesScreenshot` in `@After` hook. If a scenario fails, we attach the screenshot to the Cucumber report.*

10. **Q: How do you rerun failed test cases?**
    A: *We used Cucumber `rerun` plugin which generates a txt file of failed scenarios, then configured a separate runner to rerun only those failed tests.*

Challenges & Solutions

11. **Q: What challenges did you face in Selenium automation?**
    A: *Dynamic elements, sync issues, browser compatibility. I overcame them using explicit waits, retry logic, WebDriverManager for browser handling, and reusable methods.*

12. **Q: How do you handle synchronization issues?**
    A: *I prefer Explicit Wait (`WebDriverWait` + `ExpectedConditions`) instead of Thread.sleep(). I also use FluentWait for polling and handling exceptions.*

13. **Q: How do you handle dynamic elements in Selenium?**
    A: *I used dynamic XPath with `contains()`, `starts-with()`, and parameterized locators. Sometimes used `JavascriptExecutor` if element was hidden.*

14. **Q: What do you do when your automation script fails intermittently?**
    A: *I analyze logs, recheck locators, add synchronization, and implement retry mechanism (TestNG IRetryAnalyzer).*

15. **Q: How do you handle pop-ups, alerts, or iframes?**
    A: *Alerts – `driver.switchTo().alert()`. Iframes – `driver.switchTo().frame()`. Popups – handled using window handles or by disabling notifications using ChromeOptions.*

OOPs & Java Concepts in Automation

16. **Q: Where did you use OOPs concepts in your project?**
 A: *Encapsulation in POM (locators + methods inside classes), Inheritance for base classes (Browser setup, Hooks), Polymorphism for method overloading/overriding in utilities, Abstraction for interfaces like WebDriver.*

17. **Q: What is your understanding of inheritance in your framework?**
 A: *We created a BaseTest class with common setup/teardown methods, and all test classes extended it to reuse code.*

18. **Q: Did you create reusable methods in your framework?**
 A: *Yes, we created reusable methods for click, type, waitForElement, dropdown selection, screenshot capture, Excel read/write.*

19. **Q: How do you handle exceptions in your project?**
 A: *We used try-catch blocks with custom exception classes. For Selenium-specific issues like NoSuchElementException, TimeoutException, StaleElementException, we wrote retry logic.*

20. **Q: Have you implemented custom utilities?**
 A: *Yes, Excel utility (Apache POI), Logger utility (Log4j2), Config reader (Properties), Screenshot utility.*

Team & Agile Process

21. **Q: How do you plan your test automation in Agile?**
 A: *We participate in sprint planning, identify automatable stories, write feature files along with developers, and commit scripts within the same sprint.*

22. **Q: How do you prioritize test cases for automation?**
 A: *We automate regression, repetitive, high-risk, and high-priority scenarios. Exploratory and one-time scenarios are tested manually.*

23. **Q: How do you maintain test scripts when UI changes frequently?**
 A: *By using Page Object Model, centralizing locators, and updating only in one place.*

24. **Q: How do you communicate defects to the client/team?**
    A: *We log defects in JIRA with screenshots, logs, and steps to reproduce. For critical issues, we immediately notify the client in stand-ups or Slack/MS Teams.*

25. **Q: How do you ensure quality of automation scripts?**
    A: *By code reviews, using coding standards, and peer reviewing feature files and step definitions.*

CI/CD & Integration

26. **Q: How do you integrate Selenium with Jenkins?**
    A: *By creating Maven project jobs in Jenkins, configuring Git repository, running* `mvn clean test`*, and publishing reports.*

27. **Q: Have you integrated Selenium with any version control?**
    A: *Yes, GitHub. We maintained feature files, step defs, and runners in Git. We used branching strategy (develop, feature branches) for parallel work.*

28. **Q: Have you integrated Selenium with any other tools?**
    A: *Yes, integrated with Jenkins (CI/CD), JIRA (defect tracking), and TestNG Listeners (logging). For API automation, we used RestAssured along with Selenium for end-to-end testing.*

29. **Q: How do you trigger Selenium tests in Jenkins after code deployment?**
    A: *We configured Jenkins pipeline jobs triggered by webhooks (post-commit hook from Git).*

30. **Q: How do you manage test execution across different environments (QA, UAT, Prod)?**
    A: *By using a config.properties file and Maven profiles. Based on profile, we load environment-specific URLs and credentials.*

Real-Time Scenarios

31. **Q: How do you handle captcha in automation?**
    A: *We usually bypass it using test environment flags, disable captcha for automation users, or use API stubs. For OCR, we use third-party libs like Tesseract.*

32. **Q: How do you test file upload in Selenium?**
 A: *By sending file path to `<input type='file'>` using `sendKeys()`. If hidden, we use Robot class or AutoIT.*

33. **Q: How do you test file download in Selenium?**
 A: *By setting browser preferences in **ChromeOptions**/FirefoxProfile to auto-download without prompt.*

34. **Q: How do you handle browser cookies in Selenium?**
 A: *Using `driver.manage().getCookies()`, `addCookie()`, `deleteCookieNamed()`. Useful in login sessions.*

35. **Q: How do you test responsive design with Selenium?**
 A: *By setting Chrome window size with `driver.manage().window().setSize(new Dimension(375,812))` for mobile view.*

Client-Facing Questions

36. **Q: How do you estimate automation effort?**
 A: *We estimate based on complexity, number of test cases, reusability, and environment setup. We use story points in Agile.*

37. **Q: How do you ensure your scripts are maintainable?**
 A: *By following POM, keeping reusable utilities, meaningful naming conventions, and avoiding hard-coded values.*

38. **Q: How do you ensure scripts are robust?**
 A: *By using proper waits, exception handling, logging, and retry mechanism for flaky tests.*

39. **Q: How do you communicate automation progress to clients?**
 A: *We share daily/weekly status reports, test execution summary, and defect metrics. In sprint demos, we show automated scenarios.*

40. **Q: If a client asks to add automation for a new feature within 2 days, what will you do?**
 A: *I will prioritize by identifying critical test cases, automate the high-value ones first, and communicate the timeline clearly to clients.*

Advanced / Tricky

41. **Q: Difference between @Before, @BeforeClass, @BeforeSuite in TestNG?**
    A: *@Before – before each test method; @BeforeClass – before first method in a class; @BeforeSuite – before entire test suite execution.*

42. **Q: What is the difference between Scenario Outline and Data Table?**
    A: *Scenario Outline is for running the same scenario multiple times with different data (Examples). Data Table is for passing multiple sets of data within a single scenario.*

43. **Q: What is stale element exception and how do you handle it?**
    A: *It occurs when DOM changes and old element reference becomes invalid. I re-locate element, or use ExpectedConditions `stalenessOf()`.*

44. **Q: What is the difference between findElement and findElements?**
    A: *findElement returns first matching element or throws NoSuchElementException; findElements returns list (empty list if nothing found).*

45. **Q: How do you handle SSL certificate errors?**
    A: *By setting ChromeOptions (`--ignore-certificate-errors`) or DesiredCapabilities in Firefox.*

Client/Behavioral

46. **Q: Tell me about a time when your automation scripts caught a major defect.**
    A: *In one sprint, my **login automation detected session timeout issues** in **certain browsers**. This saved **production downtime,** and the client appreciated the QA team.*

47. **Q: How do you handle pressure when deadlines are tight?**
    A: *I prioritize critical scenarios, automate the most business-impacting flows first, and communicate challenges early with clients.*

48. **Q: Have you trained/manual testers in automation?**
    A: *Yes, I conducted KT sessions for junior QA, explained framework usage, created documentation, and helped them write the first Cucumber feature file.*

49. **Q: How do you keep yourself updated in automation?**
 A: *By following Selenium, TestNG, and Cucumber* ***documentation****,* ***blogs****, and practicing on demo sites. Recently I explored Playwright with Java.*

50. **Q: Why should we allocate you to this project?**
 A: *I bring 4 years of Selenium automation experience, worked on BDD frameworks, integrated with CI/CD, solved real-time challenges, and ensured faster delivery with quality.* ***I can take ownership and contribute from day one.***

*. **How do you handle browser notifications or SSL certificates?**

I use **ChromeOptions** or **FirefoxOptions** to disable notifications and bypass certificate errors using `--ignore-certificate-errors`

***.What was the most challenging bug you found through automation?**
My login automation found that the **password reset email link** was broken in staging. Manual testers missed it because **they tested with a cached session.** Automation ensured fresh validation.

***. Tell me about a time you solved a client escalation.**
A critical regression failed in Jenkins before UAT. I debugged quickly, found the issue was **due to expired test data**, fixed it with new data, reran, and provided green reports within 2 hours. Client appreciated the quick resolution.