

## **Tell me about your roles and responsibilities in the Project:**

My roles and responsibilities include understanding the User Stories, writing manual test cases based on user story, and perform manual testing and log bugs if found, once developer has fixed all the bugs for that test case then we mark test case as Done in JIRA, after that we add labels in test cases that whether the test case is Automation Candidate or not, after that we start picking test cases that have automation label and automate them using Selenium Java and push code to github. Also I fix the test scripts in case of failure. Also I attend client calls on daily basis and understanding their requirements and act accordingly, also I am involved in Sprint Planning meeting, attending daily 15 min scrum call in morning. I help my team members in case of any blockers.

## **Have you created a framework from scratch? Explain the architecture:**

If you have, describe the framework's purpose, components (e.g., test data management, reporting, logging), and how they interact. Mention the technologies used (e.g., Selenium, TestNG, Maven). If you haven't built one from scratch, discuss your experience working with existing frameworks and your understanding of their architecture.

## **Which design pattern have you used in the framework?**

Common patterns in test automation frameworks include Page Object Model (POM), Factory, Singleton, and Observer. Explain the pattern you used and why it was appropriate.

## **What is 3 Amigos?**

In Agile development, the 3 Amigos refers to a meeting involving the Business Analyst (representing the business), the Developer, and the Tester to discuss user stories and acceptance criteria *before* development begins. This helps ensure a shared understanding and reduces ambiguity.

## **Difference between Scrum and Kanban:**

Scrum is a time-boxed, iterative framework with sprints, roles (Product Owner, Scrum Master, Development Team), and events (Sprint Planning, Daily Scrum, Sprint Review, Sprint Retrospective). Kanban is a flow-based system with a focus on visualizing work, limiting work in progress (WIP), and continuous improvement. Kanban is more flexible and less prescriptive than Scrum.

## **Smoke vs. Sanity Testing, Regression vs. Retesting:**

- **Smoke Testing:** A quick check to ensure the core functionality of the application is working after a build.
- **Sanity Testing:** A more focused test to verify that changes in a specific area haven't broken existing functionality.
- **Regression Testing:** Testing to ensure that new changes haven't negatively impacted existing features.
- **Retesting:** Testing to verify that defects that were previously found have been fixed

## What are the important validations you will put on an API?

Common API validations include:

- Status codes (e.g., 200 OK, 400 Bad Request, 500 Internal Server Error)
- Response body structure and data types
- Headers (e.g., Content-Type)
- Authentication and authorization
- Error handling

## Suppose you have 200 test cases and 49 tests failed; how would you collect the data of only failed test cases in TestNG? Best way?

TestNG provides reporting mechanisms. You can use listeners (like `ITestListener`) to capture test results. The `onTestFailure()` method of the listener will give you access to the failed test's information. You can then log this information to a file or generate a custom report.

## What is the Maven Surefire Plugin?

The Maven Surefire Plugin is used to execute unit tests as part of the Maven build lifecycle. It generates reports of the test results.

## Explain how you pick which test cases to automate?

Prioritize based on:

- Frequency of use
- Criticality to the business
- Stability of the feature
- Repetitive tasks

## In what cases do we need to use implicit/explicit waits?

- **Implicit Wait:** Tells `WebDriver` to poll the DOM for a certain amount of time when trying to find an element. Use it when elements take a relatively consistent amount of time to appear.
- **Explicit Wait:** Waits for a specific condition (e.g., element to be visible, clickable) before proceeding. Use it for elements that load dynamically or asynchronously.

## What is a `NullPointerException`?

A `NullPointerException` is thrown when you try to access a member (method or variable) of an object that is null (i.e., not referencing any object in memory).

## **Difference between driver.get() vs. driver.navigate().to():**

Both load a URL. driver.get() waits for the page to fully load before proceeding. driver.navigate().to() doesn't necessarily wait for full page load and provides additional navigation methods (e.g., back(), forward(), refresh()).

## **How to handle Location popup in Selenium?**

This depends on the popup. You might be able to use Alert handling (driver.switchTo().alert()) if it's a browser-level alert. If it's a webpage element, you'll need to locate and interact with it using standard Selenium methods.

## **How to handle dynamic dropdowns in Selenium?**

Use Select class if the dropdown is a <select> element. Otherwise, you'll need to locate the dropdown element, click it to open the options, and then locate and click the desired option.

## **Difference Between pom.xml and testng.xml**

-> To run test scripts in a Selenium framework, both pom.xml and testng.xml files play important roles, but for different purposes. The pom.xml file is used by Maven to manage dependencies, build configurations, and automate tasks, ensuring all necessary libraries (like Selenium and TestNG) are available. While you don't run the tests directly from pom.xml, it facilitates the build process and invokes TestNG when running tests through Maven (using commands like mvn clean test).

-> The testng.xml file, on the other hand, is used to define and configure test suites, classes, groups, and execution parameters for TestNG. It allows you to specify how tests should be executed and can be run directly from an IDE like Eclipse or IntelliJ. When running tests through Maven, the pom.xml will trigger the TestNG framework and execute the test scripts as defined in testng.xml.

### **TestNG Annotations Execution Order will be:-**

-> @BeforeSuite

-> @BeforeTest

-> @BeforeClass

-> @BeforeMethod

-> @Test

-> @AfterMethod

-> @AfterClass

-> @AfterTest

-> @AfterSuite

## How do you write XPath for radio buttons?

- For a specific radio button:

```
//input[@type='radio' and @value='desiredValue']
```

- Select a radio button by its label:

```
//label[text()='LabelName']/preceding-sibling::input[@type='radio']
```

## What is the difference between findElement and findElements in Selenium?

Feature	findElement	findElements
Returns	Single web element (WebElement).	List of web elements (List).
Exception on Failure	Throws NoSuchElementException.	Returns an empty list if no elements are found.
Use Case	When exactly one element is expected.	When multiple elements are expected.

Example:

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

```
List<WebElement> elements =
```

```
driver.findElements(By.className("multipleElements"));
```

## Explain the framework you have worked on.

**Example Answer:** I have worked on a **Hybrid Framework** in Selenium. This framework integrates:

- **Data-Driven Testing:** Reads test data from Excel using Apache POI.
- **Keyword-Driven Testing:** Uses keywords to map actions with web elements.
- **Page Object Model (POM):** Encapsulates locators and methods for each page in separate classes.
- **TestNG:** Manages test execution, annotations, and reporting.
- **Maven:** Handles dependency management.
- **Extent Reports:** Generates visually rich test reports.

## Provide a detailed explanation of your framework.

**Answer:** I have built an **Automation Framework** following these principles:

1. **POM** for maintainable locators and methods.
2. **Test Data Management** using Excel (Apache POI) or JSON.
3. **Centralized Configuration** in a config.properties file.
4. **Reusable Utilities** for common tasks (e.g., screenshots, waits).
5. **Parallel Execution** using TestNG with testng.xml.
6. **Reports:** Integrated Extent Reports for detailed logs and screenshots on failure.

## Write an XPath for a given scenario.

**Scenario:** Write an XPath to select a button with text "Submit" inside a form with an ID loginForm.

```
//form[@id='loginForm']//button[text()='Submit']
```

## Write an SQL query to retrieve the details of all employees from the emp table for a given scenario.

**Scenario:** Retrieve employees whose salary is greater than 5000 and belong to the "HR" department.

```
SELECT * FROM emp  
WHERE salary > 5000 AND department = 'HR';
```

## Explain how do you pick which test cases to automate?

To decide which test cases to automate, consider the following factors:

- **Repetitive Test Cases:** Frequently executed test cases (e.g., regression suites).
- **High Risk Areas:** Modules prone to failure or business-critical functionalities.
- **Time-Consuming Test Cases:** Manual execution takes too long (e.g., large data tests).
- **Stable Features:** Features with fewer changes and stable requirements.
- **Data-Driven Tests:** Tests requiring multiple inputs and combinations.
- **Cross-Browser or Platform Tests:** Validations across multiple environments.

Avoid automating:

- Tests that change frequently.
- Exploratory or usability tests.
- One-time test cases.

## How do you handle multiple frames?

You can switch to frames in Selenium using `driver.switchTo().frame()`. There are three ways:

- By **index**: `driver.switchTo().frame(0);`
- By **name or ID**: `driver.switchTo().frame("frameName");`
- By **WebElement**:  
    `WebElement frameElement = driver.findElement(By.id("frameID"));`  
    `driver.switchTo().frame(frameElement);`

## You have 10 links; how will you print the title of each link using Selenium? (Optimal approach)

```
List<WebElement> links = driver.findElements(By.tagName("a"));
for (WebElement link : links) {
    String url = link.getAttribute("href");
    if (url != null && !url.isEmpty()) {
        driver.navigate().to(url);
        System.out.println(driver.getTitle());
    }
}
```

## Write code to find Broken Links in Selenium

```
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.List;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;

public class BrokenLinks {
    public static void main(String[] args) throws Exception {
        WebDriver driver = new ChromeDriver();
        driver.get("https://example.com");

        List
```

## In what cases do we need to use implicit/explicit wait?

- **Implicit Wait:** Used for all web elements globally. Good for simple scenarios where you want Selenium to wait for a specific duration before throwing NoSuchElementException.

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

- **Explicit Wait:** Used for more complex conditions like waiting for an element to be clickable, visible, or present.
- `WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));`
- `wait.until(ExpectedConditions.elementToBeClickable(By.id("button")));`

## What is a NullPointerException?

- A **NullPointerException** (NPE) occurs in Java when you attempt to use a reference that points to null.
- Example:
- `String str = null;`
- `System.out.println(str.length());` // Throws `NullPointerException`

## How would you switch to the parent frame?

To switch to the parent frame:

```
driver.switchTo().parentFrame();
```

To switch to the top-level (default) content:

```
driver.switchTo().defaultContent();
```

## Difference between `driver.get()` vs `driver.navigate().to()`

Feature	<code>driver.get()</code>	<code>driver.navigate().to()</code>
<b>Primary Purpose</b>	Loads a web page.	Navigates to a URL (like a browser).
<b>Wait Behavior</b>	Waits until the page is fully loaded.	May not wait for the page to load fully.
<b>Chaining</b>	Cannot navigate back/forward.	Supports navigation (back, forward).

## How to handle location pop-ups in Selenium?

For location pop-ups (browser-based alerts), use:

1. **Handle via Browser Settings:**
  - Pass a profile with location access disabled.
2. **Alert Handling:**
3. `Alert alert = driver.switchTo().alert();`
4. `alert.accept();` // Click 'Allow'



## How to handle dynamic dropdowns in Selenium?

Dynamic dropdowns are handled by:

- Locating dropdown options based on visibility or dynamic updates.
- Example:

```
WebElement dropdown = driver.findElement(By.id("dropdownID"));
dropdown.click(); // Open dropdown
List<WebElement> options = driver.findElements(By.xpath("//ul[@id='dropdownOptions']/li"
));
for (WebElement option : options) {
    if (option.getText().equals("Desired Option")) {
        option.click();
        break;
    }
}
```

## default vs public Access Modifier

Feature	Default	public
<b>Access Level</b>	Accessible within the same package only.	Accessible from any class.
<b>Keyword</b>	No explicit keyword (default behavior).	Explicitly defined as public.

## What is method hiding in Java?

- **Method hiding** occurs when a **static method** in a child class has the same name as a static method in the parent class.
- Unlike method overriding, method hiding is resolved at **compile time** based on the reference type.

Example:

```
class Parent {
    static void display() {
        System.out.println("Parent method");
    }
}

class Child extends Parent {
    static void display() {
        System.out.println("Child method");
    }
}

public class MethodHiding {
    public static void main(String[] args) {
        Parent obj = new Child();
        obj.display(); // Outputs: Parent method
    }
}
```

## What is Background in BDD Cucumber?

In Cucumber, Background is used to define steps that are common to all scenarios in a feature file. These steps are executed before each scenario. It helps avoid code duplication and improves readability.

### Example:

Feature: User Login

Background:

Given the user is on the login page

Scenario: Successful Login

When the user enters valid credentials

Then the user should see the homepage

Scenario: Failed Login

When the user enters invalid credentials

Then the user should see an error message

Here, the Background step Given the user is on the login page will execute before both scenarios.

## **Explain the Selenium Grid and how to set it up.**

### **What is Selenium Grid?**

- Selenium Grid allows you to run tests in parallel across multiple machines and browsers.
- It supports distributed testing, where you can execute tests on different OS and browser combinations.

### **How to Set It Up?**

1. **Install Selenium Grid:**
  - Download the Selenium server JAR file from the [official site](#).
2. **Start the Hub:**
  - Run the following command:
  - `java -jar selenium-server-<version>.jar hub`
  - Default Hub URL: `http://localhost:4444`
3. **Register Nodes:**
  - On a different machine or the same, run:
  - `java -jar selenium-server-<version>.jar node --hub http://<hub-ip>:4444`
4. **Configure Capabilities:**
  - Use desired capabilities in your code to specify the browser, version, and platform.
  - `DesiredCapabilities capabilities = new DesiredCapabilities();`
  - `capabilities.setBrowserName("chrome");`
  - `capabilities.setPlatform(Platform.WINDOWS);`
  - `WebDriver driver = new RemoteWebDriver(new URL("http://<hub-ip>:4444/wd/hub"), capabilities);`

### **How are you maintaining logs in your framework?**

- **Log4j/Logback:** Most commonly used for logging in Java-based frameworks.

**Explain the automation framework that you have designed.**

**Framework Type:** Hybrid (combines Data-Driven, Keyword-Driven, and POM).

**Key Components:**

1. **Page Object Model (POM):**
  - Encapsulates web element locators and actions in classes.
  - Improves maintainability.
2. **TestNG Integration:**
  - Manages test execution and reporting.
  - Supports parallel execution via testng.xml.
3. **Data-Driven Testing:**
  - Reads test data from external sources like Excel using Apache POI.
4. **Reusable Utilities:**
  - E.g., WaitHelper, ScreenshotHelper, ExcelReader.
5. **Logging and Reporting:**
  - Logs: Implemented with Log4j2.
  - Reports: Extent Reports for visual test reports.
6. **CI/CD Integration:**
  - Integrated with Jenkins for continuous testing.

## **Difference between Scenario and Scenario Outline**

<b>Feature</b>	<b>Scenario</b>	<b>Scenario Outline</b>
<b>Purpose</b>	Used for a single set of data.	Used for multiple sets of data.
<b>Keyword</b>	Scenario.	Scenario Outline.
<b>Data Input</b>	Hardcoded within the scenario steps.	Provided via Examples table.
<b>Example</b>	gherkin Scenario: Valid login When the user enters "username" and "password"	``gherkin Scenario Outline: Valid login When the user enters "" and "" Examples:

## **Write XPath for an element on a webpage**

If you share the website URL and element details, I can help craft an XPath. For now, here's an example:

**Scenario: XPath for a button with text "Submit":**

```
//button[text()='Submit']
```

## **Explain the most challenging scenario you came across during automation**

### **Example Answer:**

One challenging scenario was handling a dynamic table with AJAX-based updates. The table rows were generated dynamically with IDs changing after every refresh. To solve this:

1. Used XPath to identify rows based on visible text rather than dynamic attributes.
2. Introduced explicit wait to ensure rows were fully loaded before interaction.

## **Describe a scenario of Dev-QA tussle and how did you handle it?**

### **Example Answer:**

During testing, I reported a critical bug that the developer initially dismissed as "working as expected." To resolve:

1. Reproduced the issue with clear steps and evidence (e.g., screenshots, logs).
2. Demonstrated how the behavior deviated from the requirements.
3. Worked collaboratively to identify the root cause. Result: The issue was fixed, and trust was built through teamwork.

## **Among all the given locators, which locator is the fastest?**

### **Answer:**

- **ID** is the fastest locator because it is unique and directly uses the `getElementById()` method in the DOM.

## **If a method fails to find the element, which of the two methods throws exceptions?**

### **Answer:**

1. `findElement` throws a **NoSuchElementException** if the element is not found.
2. `findElements` does not throw an exception; instead, it returns an empty list if no elements are found.

## What is Selenium?

### Answer:

Selenium is an open-source automation testing tool for web applications. It supports multiple programming languages like Java, Python, C#, and JavaScript. Selenium consists of several components:

- **Selenium WebDriver:** Automates browser interactions.
- **Selenium IDE:** A record-and-playback tool.
- **Selenium Grid:** Allows parallel execution across multiple browsers and systems.

## How to execute multiple test cases at a time in TestNG?

### Answer:

You can execute multiple test cases by defining them in a testng.xml file.

### Example:

```
<suite name="Test Suite">
  <test name="Test Cases">
    <classes>
      <class name="tests.TestClass1"/>
      <class name="tests.TestClass2"/>
    </classes>
  </test>
</suite>
```

## Which method allows you to change control from one window to another?

### Answer:

```
driver.switchTo().window(windowHandle);
```

## Which annotations are executed first?

### Answer:

In TestNG, @BeforeSuite is executed first, followed by @BeforeTest, @BeforeClass, and @BeforeMethod.

## How many types of XPath are there in Selenium?

- **Absolute XPath:** Specifies the complete path from the root node.  
Example: /html/body/div[1]/div[2]/button.
- **Relative XPath:** Finds an element relative to another element or node.  
Example: //button[text()='Submit'].

**In Selenium 4, which method allows us to take a screenshot of a specific web element?**

**Answer:**

getScreenshotAs() method from the TakesScreenshot interface.

**Example:**

```
WebElement element = driver.findElement(By.id("example"));
File screenshot = element.getScreenshotAs(OutputType.FILE);
```

**Which language is used in Gherkin?**

**Answer:**

Gherkin uses plain English (or other supported natural languages) to write test scenarios.

**When you only want to access a single element on a webpage, which method will you use?**

**Answer:**

findElement() method.

**Which class is used for dropdowns in Selenium?**

**Answer:**

Select class is used for handling dropdowns.

**Example:**

```
Select dropdown = new Select(driver.findElement(By.id("dropdownId")));
dropdown.selectByVisibleText("Option 1");
```

**Which component of Selenium is the most important?**

**Answer:**

**Selenium WebDriver** is the most important component as it directly interacts with the browser for automation.

**Which keywords are used in Gherkin?**

**Answer:**

- Feature, Scenario, Scenario Outline, Given, When, Then, And, But, Examples, and Background.

## How many locators does Selenium have?

### Answer:

Selenium has 8 locators:

1. ID
2. Name
3. Class Name
4. Tag Name
5. Link Text
6. Partial Link Text
7. CSS Selector
8. XPath

## Which exception is shown in Selenium when there is a delay in loading elements?

TimeoutException.

## Write the correct syntax for an absolute path.

### Answer:

```
/html/body/div[1]/div[2]/button
```

## What is UI testing?

### Answer:

UI (User Interface) Testing ensures that the application interface meets the requirements and works as expected. It focuses on testing graphical elements like buttons, links, text fields, and layout.

## Which XPath searches dissimilar nodes in an XML document from the current node?

### Answer:

Axes like preceding, following, or ancestor are used to search dissimilar nodes.

### Example:

```
//div/following::span
```



## What do you mean by open-source software?

### Answer:

Open-source software is software with publicly accessible source code that allows anyone to view, modify, and distribute it.

## Which browsers support Selenium?

### Answer:

Selenium supports all major browsers, including:

- Google Chrome
- Mozilla Firefox
- Microsoft Edge
- Safari
- Opera

## Which methods are present in POM to locate WebElements?

### Answer:

In POM, we typically use:

- @FindBy annotation from Selenium's PageFactory.
- Regular driver.findElement() calls.

## Which type of wait command waits for a certain amount of time before throwing an exception?

### Answer:

**Explicit Wait** waits for a certain condition to occur before proceeding.

## Which tool will you use to manage and organize JARs and libraries in automation?

### Answer:

**Maven** is the most commonly used tool to manage JARs and dependencies in Java-based automation frameworks.

## Which design pattern have you used in your framework?

### Answer:

- **Singleton Pattern:** Ensures only one instance of WebDriver is created.
- **Factory Pattern:** Used for initializing page objects dynamically.
- **Page Object Model (POM):** Encapsulates web elements and actions in page-specific classes.

## What is 3 Amigos?

### Answer:

3 Amigos is a meeting where three key stakeholders (Business Analyst, Developer, and Tester) collaborate to:

- Discuss and clarify requirements.
- Define acceptance criteria.
- Ensure alignment between business goals, development, and testing.

## In TestNG, how will you ensure screenshots are taken only for failed test cases?

### Answer:

Use `ITestListener` and implement `onTestFailure()` to capture screenshots.

### Example:

```
@Override
public void onTestFailure(ITestResult result) {
    TakesScreenshot ts = (TakesScreenshot) driver;
    File file = ts.getScreenshotAs(OutputType.FILE);
    FileUtils.copyFile(file, new File("screenshots/" + result.getName() + ".png"));
}
```

## How to perform data-driven testing in Postman?

### Answer:

- Use **Collection Runner** in Postman with a data file (CSV/JSON).
- Add parameterized variables in the request body or headers.
- Load the data file and run the collection.

## What are the important validations you will put on an API?

### Answer:

- **Response Code:** Status codes like 200, 404, 500, etc.
- **Response Time:** Validate performance.
- **Response Body:** Schema validation, field values, and data types.
- **Headers:** Content-Type, Authorization, etc.
- **Error Handling:** Proper error messages and codes.

## Collect data of only failed test cases in TestNG.

### Answer:

Use ITestListener's onTestFailure() and save results into a log or report file.

## Print titles of 10 links in Selenium (Optimal approach).

### Answer:

```
List<WebElement> links = driver.findElements(By.tagName("a"));
for (int i = 0; i < Math.min(10, links.size()); i++) {
    System.out.println(links.get(i).getText());
}
```

## Handle Location Popup in Selenium

Use browser capabilities to disable location prompts.

### Example:

```
ChromeOptions options = new ChromeOptions();
options.addArguments("--disable-geolocation");
```

## What are different types of exceptions you faced in your framework and how you resolved them?

### Answer:

- **NoSuchElementException:** Element not found due to incorrect locator or timing. Resolved by improving locators and using waits.
- **StaleElementReferenceException:** Element is no longer attached to the DOM. Resolved by reinitializing the WebElement or using ExpectedConditions.refreshed().
- **TimeoutException:** Action took too long. Resolved by setting appropriate waits and checking application performance.
- **ElementClickInterceptedException:** Another element overlapped. Resolved by scrolling to the element or using JavaScriptExecutor.

## What is stale element exception? Why does it occur?

### Answer:

Occurs when the WebElement becomes invalid due to DOM changes after locating it.

**Resolution:** Reinitialize the WebElement or use dynamic locators.

## Use of text() in XPath?

### Answer:

text() is used to locate elements based on their visible text content.

### Example:

```
//button[text()='Submit']
```

## Why is WebDriver driver = new ChromeDriver() preferred?

### Answer:

- Follows polymorphism in Java.
- Allows flexibility to switch browser drivers without changing code (e.g., `WebDriver driver = new FirefoxDriver();`).

## API Question: Tell me Bad Request response code.

### Answer:

**400:** Indicates the request could not be understood due to malformed syntax.

## Difference between PUT and PATCH?

### Answer:

#### PUT

Replaces the entire resource.

Idempotent.

#### PATCH

Updates partial data of a resource.

Non-idempotent.

## How will you take a full-page screenshot in Selenium?

### Answer:

Using Selenium 4:

```
File screenshot = ((TakesScreenshot) driver).getFullPageScreenshotAs(OutputType.FILE);
FileUtils.copyFile(screenshot, new File("fullpage.png"));
```

## What is the use of dynamic XPath? Write dynamic XPath for the "Check Availability" button on Rediffmail Create Account Page.

### Answer:

Dynamic XPath is used when attributes or structure changes frequently.

### Example XPath:

```
//button[contains(text(),'Check Availability')]
```

## XPath axes use and functions you have used.

### Answer:

**XPath Axes:** Navigate through nodes relative to the current node (e.g., child, parent, ancestor).

### Functions:

- `contains()`: For partial matches.
- `text()`: For text-based locators.
- `starts-with()`: Locates elements with attributes starting with specific text.

## Rest Assured Question: What is the use of Request Specification and Response Specification?

### Answer:

- **Request Specification:** Define common configurations like base URI, headers, and authentication for all requests.
- **Response Specification:** Define validations for responses like status codes or response schemas.

**What is the full form of REST?**

**Answer:**

**Representational State Transfer.**

**What is JavaScriptExecutor? Write code.**

**Answer:**

**JavaScriptExecutor** executes JavaScript code within the browser.

**Example:**

```
JavaScriptExecutor js = (JavaScriptExecutor) driver;  
js.executeScript("arguments[0].click();", element);
```

**Different ways to click on an element in Selenium (3 ways).**

**Answer:**

1. **Standard click:**

```
element.click();
```

2. **JavaScriptExecutor:**

```
js.executeScript("arguments[0].click();", element);
```

3. **Actions class:**

```
Actions actions = new Actions(driver);  
actions.moveToElement(element).click().perform();
```

**How to handle multiple windows in Selenium? Write code.**

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

**Write code to read data from an Excel file.**

**Answer:**

```
FileInputStream fis = new FileInputStream("data.xlsx");
Workbook workbook = new XSSFWorkbook(fis);
Sheet sheet = workbook.getSheetAt(0);
for (Row row : sheet) {
    for (Cell cell : row) {
        System.out.println(cell.getStringCellValue());
    }
}
```

**Are you comfortable working with manual testing if needed?**

**Answer:**

Yes, I am comfortable with manual testing and use it for exploratory testing or validating edge cases.

**Write an XPath for a given scenario.**

**Answer:**

Scenario: Locate a button with text "Submit".

```
//button[text()='Submit']
```

Scenario: Locate the first name field based on placeholder "First Name".

```
//input[@placeholder='First Name']
```

**Write an SQL query to retrieve the details of all employees from the emp table for a given scenario.**

**Answer:**

Scenario: Retrieve details of employees with salary > 50000.

```
SELECT * FROM emp WHERE salary > 50000;
```

Scenario: Retrieve employees whose department is "HR".

```
SELECT * FROM emp WHERE department = 'HR';
```

## What is invocationCount used for in TestNG?

### Answer:

- **Purpose:** Used to execute a test method multiple times.
- **Example:**
- `@Test(invocationCount = 5)`
- `public void runMultipleTimes() {`
- `System.out.println("Test executed");`
- `}`

## How would you wait for the visibility of an element in Selenium?

### Answer:

Using **WebDriverWait**:

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

## How would you use AutoIT to upload a file?

### Answer:

1. Create an AutoIT script to handle the OS file dialog:

```
ControlFocus("File Upload", "", "Edit1")  
ControlSetText("File Upload", "", "Edit1", "C:\\file.txt")  
ControlClick("File Upload", "", "Button1")
```

2. Compile the script into an .exe file.
3. Call it in Selenium using:

```
Runtime.getRuntime().exec("path_to_autoit_script.exe");
```



## What is the ElementClickInterceptedException? How to resolve it?

**Answer:**

- **Cause:** Another element is overlapping the element you're trying to click.
- **Solutions:**

- - Use JavaScriptExecutor to click:

```
((JavascriptExecutor)driver).executeScript("arguments[0].click();", element);
```

Ensure element is visible or scroll into view:

- Actions actions = new Actions(driver);
- actions.moveToElement(element).click().perform();

## What are the challenges you faced while working with a framework?

**Answer:**

- **Dynamic elements:** Resolved using dynamic XPath and waits.
- **Test data management:** Integrated Apache POI for Excel.
- **Cross-browser compatibility:** Used Selenium Grid for parallel execution.
- **Reporting:** Enhanced with Extent Reports for custom logs.

## What is the normalize-space function in XPath?

**Answer:**

- Removes leading and trailing whitespaces and replaces consecutive spaces with a single space.
- **Use Case:** For text-based locators where extra spaces may exist.
- `//button[normalize-space(text()='Submit']`

## How do you disable images in Selenium?

**Answer:**

Disable images by setting browser preferences.

**Example for Chrome**

```
ChromeOptions options = new ChromeOptions();
Map<String, Object> prefs = new HashMap<>();
prefs.put("profile.managed_default_content_settings.images", 2);
options.setExperimentalOption("prefs", prefs);
WebDriver driver = new ChromeDriver(options);
|
```

## Difference between Action vs Actions in Selenium?

Action	Actions
Represents a single user action (e.g., click).	A class used to build a chain of actions.
Example: Action clickAction = actions.click(element).build();	Example: Actions actions = new Actions(driver);

## How do you deal with dynamically changing element attributes in Selenium?

**Answer:**

- Use dynamic XPath with partial match:
- `//div[contains(@id, 'dynamic_part')]`
- Leverage relative locators (Selenium 4).
- Introduce waits for stability.

## Purpose of using the ThreadLocal class in Selenium?

**Answer:**

- Maintains separate WebDriver instances for parallel execution in threads.
- Prevents thread interference.

**Example:**

```
private static ThreadLocal<WebDriver> driver = new ThreadLocal<>();
public static WebDriver getDriver() {
    return driver.get();
}
```

## API Status Codes: 200, 400, 410, 403 Differences

- **200 (OK):** Request succeeded.
- **400 (Bad Request):** Client-side error, invalid request syntax.
- **410 (Gone):** Resource no longer available.
- **403 (Forbidden):** Access is denied

## 7. How do you write a test case in Postman to validate the status code?

**Postman test script:**

```
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});
```

## How would you perform data-driven testing in Postman?

- Use **Postman Runner** with a data file (CSV/JSON).

Link variables in the collection:

```
{
  "username": "{{username}}",
  "password": "{{password}}"
}
```

## Difference between HEAD and OPTIONS API methods?

### HEAD

Returns headers without the body.

### OPTIONS

Returns allowed HTTP methods for a resource.

## Have you worked with JMeter? Basics related to ThreadGroup, Listeners

**ThreadGroup:** Defines the number of users and requests.

**Listeners:** Capture and visualize test results (e.g., View Results Tree, Summary Report).

## How would you handle dynamic elements in Selenium that do not have stable locators?

Handling dynamic elements in Selenium that do not have stable locators requires employing various strategies. Here are several approaches to tackle this issue:

### 1. Use of Dynamic XPath:

For dynamic elements, the XPath can be written in a way that accommodates changes in attributes or parts of the element. For example, using the contains() or starts-with() functions helps when the attributes of the elements change slightly.

#### Example:

```
//button[contains(@id, 'submit')]
```

This XPath is more flexible than one that matches the exact ID, as it will find the button even if part of the ID changes dynamically.

### 2. Use of Relative Locators (Selenium 4):

Relative locators are a powerful feature in Selenium 4, where you can find elements based on their proximity to other elements.

#### **Example:**

```
WebElement element =  
driver.findElement(RelativeLocator.with(By.tagName("button")).toRightOf(By.id("submit"))  
);
```

This locates the button relative to another element with the ID "submit".

### **3. Explicit Waits with Conditions:**

Dynamic elements can appear at any time, and using an **explicit wait** helps ensure that the test waits for the element to appear, be clickable, or meet some other condition.

#### **Example:**

```
WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));  
WebElement element =  
wait.until(ExpectedConditions.elementToBeClickable(By.xpath("//button[@class='dynamic']  
")));
```

### **4. CSS Selectors with Partial Matches:**

Like XPath, **CSS selectors** can also be used with partial matches for dynamically changing attributes.

#### **Example:**

```
button[class*='submit']
```

This selector will find buttons whose class contains 'submit' anywhere in the string.

### **5. Use JavaScriptExecutor:**

In cases where locators are unstable or unreliable, using **JavaScriptExecutor** to directly interact with elements can be a good alternative. For example, you can simulate a click or get the element's properties through JavaScript.

#### **Example:**

```
JavascriptExecutor js = (JavascriptExecutor) driver;  
js.executeScript("arguments[0].click();", element);
```

### **6. Handling IFrames:**

If dynamic elements are embedded inside **iFrames**, you need to switch to the iframe before interacting with the elements inside.

### Example:

```
driver.switchTo().frame("iframeName");  
WebElement element = driver.findElement(By.id("dynamicElement"));
```

## 7. Retry Mechanism:

In case of temporary element instability (e.g., elements loading asynchronously), implementing a retry mechanism can be helpful. If an element is not found on the first attempt, the script retries after a short delay.

### What strategies would you implement to ensure the stability and reliability of your automated tests?

Ensuring the stability and reliability of automated tests is crucial for maintaining a robust testing process. Below are strategies you can implement:

#### 1. Use Robust and Stable Locators:

Always prefer stable attributes for locating elements, such as IDs, name attributes, or unique data-\* attributes. Avoid relying on fragile or frequently-changing attributes like classes or styles.

- **Example:** Use data-id attributes or unique aria-labels where possible.

#### 2. Implement Explicit Waits:

Using **explicit waits** ensures that your tests wait for an element to become visible, clickable, or meet other necessary conditions before interacting with it. This avoids issues related to timing or elements that take time to load.

- **Example:** Wait for an element to be present or visible before interacting with it.

#### 3. Leverage Page Object Model (POM):

The **Page Object Model** is a design pattern that promotes the creation of classes for each page, encapsulating elements and actions. This improves test maintenance, reusability, and makes the tests more readable.

- **Example:** A page class that contains methods like login(), addItemToCart(), etc., keeps the locators and actions separate from test scripts.

#### 4. Keep Tests Independent:

Each test should be independent of others, ensuring that failure in one test does not affect others. You can achieve this by **isolating test data**, resetting states, and avoiding dependencies between tests.

#### 5. Implement Retry Logic for Flaky Tests:

For tests that are occasionally unstable due to network or timing issues, implementing a **retry mechanism** can help. Tools like **TestNG's retry analyzer** can help rerun failed tests a specified number of times before marking them as failed.

## **6. Use Data-Driven Testing:**

**Data-driven testing** allows running the same test with different sets of input data, improving test coverage without duplicating code. Tools like **Apache POI** (for Excel) or **CSV files** are commonly used to pass test data.

## **7. Integrate Continuous Integration (CI):**

Integrating your tests with a **CI/CD pipeline** (such as Jenkins, GitLab CI) ensures that tests run automatically on each commit. This helps in detecting issues early and provides immediate feedback to developers.

## **8. Version Control for Test Scripts:**

Store your test scripts in a **version control system** (such as Git) to manage changes, track revisions, and collaborate efficiently with team members. It also helps in keeping track of which versions of tests work with which versions of the application.

## **9. Log and Report Test Results:**

Using tools like **ExtentReports** or **Allure Reports**, log detailed information about test execution (including steps, screenshots, and failure reasons). This helps in debugging and identifying root causes quickly.

## **10. Handle External Dependencies:**

In case your tests interact with external systems (e.g., APIs, databases, third-party services), make sure to mock these dependencies during testing or use controlled test environments to ensure reliability.

## **11. Perform Cross-Browser Testing:**

Ensure your tests are executed across different browsers (e.g., Chrome, Firefox, Edge) to identify browser-specific issues early. Selenium Grid can be used for parallel execution.

## **12. Regular Test Maintenance:**

Review and update tests regularly to ensure that they remain valid, particularly when the application undergoes changes. This includes adjusting locators, test logic, and data-driven configurations.

## What is the advantage of Selenium WebDriver?

Selenium WebDriver offers several advantages:

- **Cross-Browser Support:** It supports multiple browsers like Chrome, Firefox, Safari, Edge, etc.
- **Programming Language Support:** It supports several programming languages such as Java, Python, C#, Ruby, and JavaScript.
- **Ease of Use:** Selenium WebDriver offers simple and intuitive APIs for automating web browsers.
- **Browser-Specific Features:** It interacts directly with browsers (without the need for a proxy) for faster and more accurate results.
- **Cross-Platform Compatibility:** It can be run on Windows, Mac OS, and Linux environments.

## What are the different types of locators supported by Selenium WebDriver?

The different types of locators supported by Selenium are:

- **ID:** `driver.findElement(By.id("element_id"))`
- **Name:** `driver.findElement(By.name("element_name"))`
- **Class Name:** `driver.findElement(By.className("element_class"))`
- **Tag Name:** `driver.findElement(By.tagName("tag_name"))`
- **Link Text:** `driver.findElement(By.linkText("link_text"))`
- **Partial Link Text:** `driver.findElement(By.partialLinkText("partial_link_text"))`
- **XPath:** `driver.findElement(By.xpath("xpath_expression"))`
- **CSS Selector:** `driver.findElement(By.cssSelector("css_selector"))`

## Explain the concept of TestNG and how it is used with Selenium for test automation?

TestNG is a testing framework inspired by JUnit but with more powerful features, such as annotations, parameterization, parallel test execution, and test configuration. In Selenium, TestNG is used for:

- **Running test cases:** TestNG enables you to run multiple test cases in parallel or sequentially.
- **Annotations:** `@Test`, `@BeforeMethod`, `@AfterMethod`, etc., define test flow.
- **Data-Driven Testing:** You can use `@DataProvider` to feed different test data to tests.

## How do you perform mouse and keyboard actions using Selenium WebDriver?

Selenium WebDriver provides the **Actions** class to perform advanced user interactions like mouse movements, clicks, drag-and-drop, etc., and keyboard actions like typing and pressing keys.

**Example:**

```
Actions actions = new Actions(driver);
actions.moveToElement(element).click().build().perform();
```

### **Keyboard actions:**

```
actions.sendKeys(Keys.ENTER).perform();
```

## **What are the advantages and limitations of Selenium for test automation?**

### **Advantages:**

- Open-source and widely used.
- Supports multiple browsers and programming languages.
- Integration with other testing tools and frameworks (TestNG, Jenkins, Maven).

### **Limitations:**

- Does not support mobile app automation directly (but can be used with Appium).
- Limited support for capturing screenshots in non-browser environments.
- Requires a stable internet connection for remote grid execution.

## **How do you handle SSL certificates and security-related issues in Selenium?**

Selenium WebDriver provides options to handle SSL certificates:

- In Chrome, you can disable SSL verification using:
- `ChromeOptions options = new ChromeOptions();`
- `options.addArguments("--ignore-certificate-errors");`
- `WebDriver driver = new ChromeDriver(options);`
- Similarly, in Firefox, you can configure Profile to accept SSL certificates.

## **Can you automate testing for mobile applications using Selenium? If yes, how?**

Yes, mobile application testing can be automated using **Appium** (which is built on top of Selenium). Appium allows you to automate both Android and iOS applications.

## **How do you manage test data and test configurations in Selenium tests?**

You can manage test data and configurations by:

- Using external files such as **Excel**, **CSV**, or **JSON** to store test data and configurations.
- Using **TestNG parameters** or **@DataProvider** for data-driven testing.
- For configurations, you can use properties files or environment variables.



## What is Page Object Model (POM), and why is it used in Selenium automation?

Page Object Model (POM) is a design pattern that promotes the creation of separate classes for each web page in the application. Each page class contains the elements and actions (methods) associated with that page. POM is used to:

- Improve maintainability by separating test logic from UI actions.
- Reduce code duplication by reusing page classes.
- Make tests more readable and manageable.

## How do you handle exceptions and errors in Selenium WebDriver scripts?

You can handle exceptions by using try-catch blocks. Selenium provides various exceptions, such as NoSuchElementException, TimeoutException, ElementNotVisibleException, etc., which you can catch and handle accordingly.

### Example:

```
try {
    WebElement element = driver.findElement(By.id("elementId"));
} catch (NoSuchElementException e) {
    System.out.println("Element not found");
}
```

## How to take a screenshot in Selenium?

You can take a screenshot using the TakesScreenshot interface in WebDriver.

### Example:

```
File screenshot=((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);
FileUtils.copyFile(screenshot, new File("screenshot.png"));
```

## Data provider in TestNG?

@DataProvider in TestNG is used to pass different sets of data to a test method. It allows the execution of the same test with multiple sets of data.

### Example:

```
@DataProvider(name = "testData")
public Object[][] data() {
    return new Object[][] {
        {"testdata1", "expectedResult1"},
        {"testdata2", "expectedResult2"}
    };
}
```

```
@Test(dataProvider = "testData")
public void testMethod(String inputData, String expectedResult) {
    // Test logic
}
```

## How to validate if links are valid or not on the webpage?

You can validate links by:

1. Extracting all links using XPath or CSS selectors.
2. Sending HTTP requests to check if the link returns a valid response code (e.g., 200 OK).

### Example:

```
List<WebElement> links = driver.findElements(By.tagName("a"));
for (WebElement link : links) {
    String url = link.getAttribute("href");
    HttpURLConnection connection = (HttpURLConnection) new URL(url).openConnection();
    connection.setRequestMethod("HEAD");
    connection.connect();
    int responseCode = connection.getResponseCode();
    System.out.println(url + " - " + responseCode);
}
```

## How do you manage drag and drop activity in Selenium?

You can use the Actions class to perform drag-and-drop actions.

### Example:

```
WebElement source = driver.findElement(By.id("source"));
WebElement target = driver.findElement(By.id("target"));
Actions actions = new Actions(driver);
actions.dragAndDrop(source, target).build().perform();
```

## What is the use of the testng.xml file?

The testng.xml file is used to configure the test execution:

- It defines the test suites, test cases, groups, and parameters.
- It allows for parallel execution of tests.
- It helps in managing test execution order, exclusions, and grouping tests.

## **.What is StaleElementException? Why does it occur with the use of text() in XPath?**

- **StaleElementException** occurs when an element is no longer attached to the DOM. This often happens if the page is refreshed or dynamically modified, and the element reference is no longer valid.
- When using text() in XPath, if the DOM is updated and the element is modified or replaced, the reference to that element may become stale. To resolve this, we can relocate the element again before interacting with it.

## **Why is WebDriver driver = new ChromeDriver() more preferred?**

- **WebDriver driver = new ChromeDriver();** is preferred because it is a simple, clean way of initializing a new instance of the Chrome browser to automate tests. The ChromeDriver is part of the Selenium WebDriver package and is specifically built to interact with Chrome browser, providing better compatibility and performance for automation tasks.

## **Do you know about Cucumber and TestNG? Explain about it?**

- **Cucumber:** Cucumber is a tool used for behavior-driven development (BDD). It uses Gherkin language to write test scenarios in natural language (Given-When-Then) that are easy to understand for both developers and non-developers. It helps automate acceptance criteria by allowing collaboration between developers, testers, and business stakeholders.
- **TestNG:** TestNG is a testing framework inspired by JUnit but designed to cover a broader range of testing categories, including unit testing, integration testing, and end-to-end testing. TestNG provides features such as:
  - Annotations (@Test, @BeforeClass, @AfterMethod, etc.)
  - Grouping of test cases
  - Data-driven testing with @DataProvider
  - Parallel test execution
  - Configurable reports.

## **If we want to execute one functionality in common, what is the best way to use?**

The best way to execute one functionality in common is to use **Page Object Model (POM)**, where each page of the application is represented as a class with methods that interact with the elements on that page. This allows you to reuse functionality in different test cases without duplication, and changes to a page's structure will only require updates in the POM class, not across the entire test suite.

## Usage of scenario outline in Cucumber:

The **Scenario Outline** in Cucumber allows you to run the same test scenario with multiple sets of data. It uses the **Examples** table to supply different input values and expected outputs for the same scenario.

Scenario Outline: Add two numbers

Given I have two numbers <num1> and <num2>

When I add the numbers

Then the result should be <sum>

Examples:

num1	num2	sum
5	3	8
10	15	25

## How to run the specific test cases in Cucumber that need to be run on a daily basis?

You can use **tags** in Cucumber to specify which tests should be run. For example:

@DailyTests

Scenario: Some daily test case

Given ...

When ...

Then ...

You can then run the tests with the tag @DailyTests by specifying it in the command:

```
cucumber -t @DailyTests
```

## Page Object Model for a Login Page

Here's an implementation of the **Page Object Model** (POM) for a login page with a username, password, and login button. It includes methods for both valid and invalid login scenarios.

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;

public class LoginPage {
    WebDriver driver;

    // Locators for elements on the page
    By usernameField = By.id("username");
    By passwordField = By.id("password");
    By loginButton = By.id("loginBtn");
    By errorMessage = By.id("errorMsg");

    // Constructor to initialize the WebDriver
    public LoginPage(WebDriver driver) {
        this.driver = driver;
    }

    // Method to enter username
    public void enterUsername(String username) {
        driver.findElement(usernameField).sendKeys(username);
    }

    // Method to enter password
    public void enterPassword(String password) {
        driver.findElement(passwordField).sendKeys(password);
    }

    // Method to click the login button
    public void clickLoginButton() {
        driver.findElement(loginButton).click();
    }

    // Method to get the error message
    public String getErrorMessage() {
        return driver.findElement(errorMessage).getText();
    }

    // Valid login method
    public HomePage validLogin(String username, String password) {
        enterUsername(username);
        enterPassword(password);
        clickLoginButton();
        return new HomePage(driver); // Redirects to HomePage after successful login
    }
}
```

```

    }

    // Invalid login method
    public void invalidLogin(String username, String password) {
        enterUsername(username);
        enterPassword(password);
        clickLoginButton();
    }
}

```

### Explanation:

- The LoginPage class contains methods to interact with the username field, password field, and login button.
- The validLogin() method performs a successful login and redirects to the HomePage.
- The invalidLogin() method simulates an unsuccessful login attempt.
- A method getErrorMessage() is used to capture error messages in case of invalid login.

### Do you create 100 page objects for 100 pages?

While it's ideal to create a separate page object for each page in your application (1:1 mapping), it's not always necessary to have exactly 100 page objects. If multiple pages share common elements, you can create a generic page object for those pages and reuse it.

For instance:

- If several pages share the same login form, you can have a single LoginPage object.
- You can also create **BasePage** classes with common methods like click(), sendKeys(), etc.

Thus, while creating 100 page objects is possible, **page object reuse** and **abstracting common actions** can help keep the framework manageable.

## Implement a TestNG listener to take a screenshot of a failed test case

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.OutputType;
import org.openqa.selenium.TakesScreenshot;
import org.testng.ITestListener;
import org.testng.ITestResult;
import org.apache.commons.io.FileUtils;
import java.io.File;
import java.io.IOException;

public class TestListener implements ITestListener {
    WebDriver driver;

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

    @Override
    public void onTestFailure(ITestResult result) {
        TakesScreenshot screenshot = (TakesScreenshot) driver;
        File srcFile = screenshot.getScreenshotAs(OutputType.FILE);
        try {
            FileUtils.copyFile(srcFile, new File("screenshots/" + result.getName() + ".png"));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

## Cucumber feature file for a search functionality in an e-commerce application

Feature: Search functionality in E-commerce website

Scenario: Searching for a product by category

Given I am on the homepage of the e-commerce website  
When I select the "Electronics" category from the filter  
And I enter "Laptop" in the search box  
And I click the "Search" button  
Then the search results should display products in the "Laptop" category  
And the category filter should be "Electronics"

## Handling shared test data between multiple Cucumber scenarios

You can use **Cucumber hooks** (like @Before and @After) or **Scenario-level variables** to share test data between scenarios. If the test data needs to be shared, you can store it in a **static variable** or a **singleton class** for access across scenarios.

```
public class SharedTestData {
    public static String username = "testuser";
    public static String password = "password123";
}
```

## ElementClickInterceptedException - Explanation and Resolution

### What is it?

- This exception occurs when an element is **obscured by another element** (like a modal, overlay, or popup) and cannot be clicked.

### How to resolve:

- Wait for the element to become clickable using explicit waits.
- Scroll the element into view.

```
WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
WebElement element = wait.until(ExpectedConditions.elementToBeClickable(By.id("button")));
element.click();
```

## Handle Stale Element Exceptions in Selenium

### What is it?

- This occurs when a reference to an element is no longer valid, typically due to the DOM being refreshed or updated after finding the element.

### How to handle:

- Re-locate the element after encountering a stale element exception.

```
WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
WebElement element = wait.until(ExpectedConditions.presenceOfElementLocated(By.id("elementId")));
```

## Purpose of the Actions Class in Selenium and Double-click Action

The **Actions class** is used for performing complex user gestures like mouse movements, drag-and-drop, clicking multiple elements, and more.

### Example for double-click:

```
Actions actions = new Actions(driver);
WebElement element = driver.findElement(By.id("someElement"));
actions.doubleClick(element).perform();
```



## Handling file uploads and downloads in Selenium WebDriver

- **File Upload:** Use the `sendKeys()` method to simulate file uploads.

```
WebElement uploadElement = driver.findElement(By.id("upload"));
uploadElement.sendKeys("C:\\path\\to\\file.jpg");
```

- **File Download:** Use browser settings to automatically handle file downloads or use a tool like **AutoIT** for more complex scenarios.

## Fluent Waits in Selenium

**What is it?** A **FluentWait** allows you to define the frequency with which Selenium will check for the condition, as well as ignoring specific exceptions (like `NoSuchElementException`) while waiting for an element to appear.

```
FluentWait<WebDriver> wait = new FluentWait<>(driver)
    .withTimeout(Duration.ofSeconds(30))
    .pollingEvery(Duration.ofSeconds(5))
    .ignoring(NoSuchElementException.class);
WebElement element = wait.until(driver -> driver.findElement(By.id("someElement")));
```

## JMeter Test Plan with 50 Threads to Hit a Sample API Endpoint

1. **Create a Test Plan** with 50 threads (virtual users).
2. **Add HTTP Request** for the API endpoint.
3. **Add an Assertion** to check response time is under 2 seconds.
4. Add **Listeners** to capture results like View Results Tree and Summary Report.

## Configure an Azure Pipeline to Trigger Automated Tests

1. In **Azure DevOps**, go to **Pipelines** and create a new pipeline.
2. **Add tasks** to install dependencies, run tests (e.g., Selenium scripts), and publish results.
3. Use the `mvn test` command to run the tests in your framework.
4. Set up **Triggers** for the pipeline to run on code commit.

## Integrating Azure DevOps with Test Framework

- Create a service connection to your repository in **Azure DevOps**.
- Use **Maven** or **Gradle** build tasks to trigger test execution.
- Publish the test results and logs as pipeline artifacts.

## Purpose of Azure Pipelines and How They Can Be Configured to Run Test Suites Automatically

Azure Pipelines is a cloud-based continuous integration and continuous deployment (CI/CD) service that helps automate the building, testing, and deployment of applications. It can be configured to run automated test suites after every code commit, ensuring that new code does not break existing functionality.

### How to configure it to run test suites automatically:

1. **Create a Pipeline:** In Azure DevOps, create a new pipeline that connects to your version control system (GitHub, Azure Repos, etc.).
2. **Define Tasks:**
  - **Install dependencies:** Use tasks like Maven or npm to install dependencies for your framework.
  - **Run Tests:** Add a task to execute your tests (e.g., mvn test for Maven projects).
  - **Publish Test Results:** Add a task to publish the test results, such as using the Publish Test Results task for JUnit or NUnit results.
3. **Set Up Triggers:** Configure triggers to run the pipeline automatically whenever there is a code change (e.g., on push to the main branch).
4. **Post-test Actions:** You can configure the pipeline to send notifications, generate reports, or deploy the application based on the results.

### What is OAuth2.0:

This is an Authorization mechanism, not an Authentication mechanism.

OAuth stands for Open Authorization as this is an open standard.

OAuth2.0 is a protocol, or we can call it a framework as well.

Purpose: OAuth2.0 is used to authorize 3rd party applications to access resources of a user on behalf of the user.

Roles in OAuth2.0:

1. **Resource Owner:** Person who owns the resources.
2. **Client:** 3rd party application that wants to access Resource owner resources. So, it should have the Access token of the Resource owner.
3. **Authorization Server:** Authorization server helps the client in obtaining the access token of the Resource owner.
4. **Resource Server:** Once the client has the Access Token, it can use the access token to access Resource owner's protected resources.

How OAuth2.0 flow works?

- First, the Client should be registered against the OAuth server to obtain a client id and client secret.

- Now, the client will make an Authorization Request to the Authorization server with the client id and client secret.
4. Resource Server: Once the client has Access Token it can use the access token to access Resource owner's protected resources.

How OAuth2.0 flow works?

- First Client should be registered against OAuth server to obtain client id and client secret
- Now client will make Authorization Request to Authorization server with client id and client secret
- Now Resource owner needs to enter credentials to Authorize 3rd party applications to access Resource Owner resources
- Once the Resource owner grants permission, the Authorization server will issue an Authorization code
- Now, the client needs to hit the Token endpoint to obtain an Access Token by passing the Authorization Code received in the previous step
- With the Access Token, the client can now access the protected Resources of the Resource owner

## How will you parse complex JSON files using REST Assured?

In REST Assured, parsing complex JSON can be done using **JsonPath**. You can extract values from the JSON response and then process them as needed.

Example:

```
import io.restassured.path.json.JsonPath;
import io.restassured.response.Response;

public class ParseJson {
    public static void main(String[] args) {
        // Assume we have a complex JSON response from an API
        Response response = RestAssured.get("https://api.example.com/getDetails");

        JsonPath jsonPath = response.jsonPath();

        // Parsing a complex JSON structure
        String value = jsonPath.get("parentObject.childObject[0].subChildField");
        System.out.println("Parsed Value: " + value);
    }
}
```

You can navigate through nested JSON objects using JsonPath expressions like `parentObject.childObject[0].subChildField`

## Explain Authentication and Authorization in API testing.

- **Authentication:** It is the process of verifying the identity of a user or system. Common methods include Basic Authentication, OAuth, and API keys.
- **Authorization:** It determines whether the authenticated user has permission to access a resource. It ensures that users can only perform actions within their level of access.

In REST Assured:

- **Basic Authentication:**
- RestAssured.given()
- .auth().basic("username", "password")
- .get("https://api.example.com/protected");
- **Bearer Token (OAuth):**
- RestAssured.given()
- .header("Authorization", "Bearer yourAccessToken")
- .get("https://api.example.com/protected");

## What is Object mapper in REST Assured?

Object Mapper is used for converting Java objects to JSON and vice versa. In REST Assured, you can use **Jackson ObjectMapper** or **Gson** to map complex Java objects to JSON and JSON back to Java.

Example:

```
import com.fasterxml.jackson.databind.ObjectMapper;

public class ObjectMapperExample {
    public static void main(String[] args) throws Exception {
        MyClass obj = new MyClass("Test", 25);
        ObjectMapper objectMapper = new ObjectMapper();

        // Convert Java Object to JSON
        String json = objectMapper.writeValueAsString(obj);
        System.out.println(json);

        // Convert JSON back to Java Object
        MyClass newObj = objectMapper.readValue(json, MyClass.class);
        System.out.println(newObj.getName());
    }
}
```

## Explain PoJo concept.

POJO (Plain Old Java Object) is a simple Java object with properties (fields), getters, setters, and a constructor, used for mapping data between JSON and Java objects.

## What are listeners in TestNG?

Listeners in TestNG allow you to intercept and customize test execution behavior. They can be used to perform actions before/after tests, such as logging, reporting, or taking screenshots.

Common listeners:

- **ITestListener**: Listens for test execution events (onStart, onFinish, onTestSuccess, onTestFailure).
- **ISuiteListener**: Listens for suite start and finish events.
- **ITestResult**: Used to analyze the result of the tests.

Example:

```
public class MyTestListener implements ITestListener {  
    public void onTestFailure(ITestResult result) {  
        System.out.println("Test Failed: " + result.getName());  
    }  
}
```

## How will you resolve the conflicts on GitHub?

To resolve conflicts in GitHub:

1. **Fetch the latest changes** from the remote repository using `git fetch`.
2. **Switch to your branch** and try to merge the main branch into it: `git merge main`.
3. Resolve conflicts manually by editing the files marked as conflicted.
4. After resolving, **commit the changes**: `git commit -m "Resolved merge conflicts"`.
5. **Push the changes** back to the repository: `git push`.

## What is the Git Stash command used for?

The `git stash` command temporarily saves your uncommitted changes (modifications, additions, and deletions) so that you can work on something else without committing or losing your work. You can retrieve it later using `git stash apply`.

## What Challenges did you face while designing your automation framework?

Challenges faced while designing an automation framework might include:

- **Selecting appropriate tools** (e.g., Selenium, TestNG, Rest Assured, etc.).
- **Managing dynamic and complex elements** with robust locators.
- **Handling dependencies** like waiting for elements to load.
- **Ensuring cross-browser compatibility**.
- **Integrating with CI/CD pipelines** for continuous testing.
- **Handling large volumes of test data** in data-driven testing.

## What is Robot Framework, and why is it used?

### Answer:

- Robot Framework is an open-source, keyword-driven test automation framework.
- It is primarily used for acceptance testing and acceptance test-driven development (ATDD).
- It supports both **Data-Driven Testing** and **Keyword-Driven Testing**.
- It is built on top of Python and uses libraries like **SeleniumLibrary** for browser automation.
- It is popular because of its easy-to-read syntax and extensibility with custom libraries.

## How do you organize a test suite in Robot Framework?

### Answer:

- In Robot Framework, tests are organized into **Test Suites**, which can be a single file or a directory containing multiple test files.
- A test suite can include:
  - **Settings:** Metadata and configurations like library imports.
  - **Variables:** To store static or dynamic data.
  - **Test Cases:** Individual test steps.
  - **Keywords:** Reusable actions or functions.

```
*** Settings ***
Library    SeleniumLibrary
Suite Setup    Open Browser    https://example.com    chrome
Suite Teardown    Close Browser

*** Variables ***
${USERNAME}    user1
${PASSWORD}    pass123

*** Test Cases ***
Login with valid credentials
    Input Text    id=username    ${USERNAME}
    Input Text    id=password    ${PASSWORD}
    Click Button    id=login
    Page Should Contain    Welcome, ${USERNAME}

*** Keywords ***
Open Browser
    Open Browser    ${URL}    ${BROWSER}
```

## How do you handle data-driven testing in Robot Framework?

Answer:

- **Data-driven testing** can be done using the `Test Template` keyword.
- It allows running the same test case with multiple data sets.

Example:

```
robot
CopyEdit
*** Settings ***
Library      SeleniumLibrary

*** Variables ***
${URL}       https://example.com
${BROWSER}   chrome

*** Test Cases ***
Login Test
    [Template]    Check Login
    user1    pass123    Welcome, user1
    user2    pass456    Welcome, user2

*** Keywords ***
Check Login
    Input Text    id=username    ${USERNAME}
    Input Text    id=password    ${PASSWORD}
    Click Button    id=login
    Page Should Contain    ${WELCOME_MESSAGE}
```

## How do you integrate Robot Framework with CI/CD tools like Jenkins?

Answer:

- Install the **Robot Framework plugin** in Jenkins.
- Create a Jenkins pipeline or freestyle project.
- Configure the build step to execute tests using the command:

```
bash
CopyEdit
robot -d results tests/
```

- Use the **Post-build Actions** to publish test results.

## How do you create custom keywords in Robot Framework?

### Answer:

- Custom keywords can be created in two ways:
  1. **User-Defined Keywords** in the Robot file itself.
  2. **Python-defined Keywords** by writing a Python function and importing it as a library.

### Example of User-Defined Keyword:

```
robot
CopyEdit
*** Keywords ***
Login To Application
    Input Text    id=username    ${USERNAME}
    Input Text    id=password    ${PASSWORD}
    Click Button  id=login
```

### Example of Python-Defined Keyword:

```
python
CopyEdit
# custom_library.py
from selenium import webdriver

class CustomLibrary:
    def open_browser(self, url, browser='chrome'):
        if browser == 'chrome':
            self.driver = webdriver.Chrome()
        elif browser == 'firefox':
            self.driver = webdriver.Firefox()
        self.driver.get(url)
```

### Usage in Robot File:

```
robot
CopyEdit
*** Settings ***
Library    custom_library.py

*** Test Cases ***
Open Custom Browser
    Open Browser    https://example.com    chrome
```

.



## What is TestNG, and why is it used?

### Answer:

- TestNG is a testing framework inspired by JUnit but designed to cover a wider range of test categories, including unit tests, functional tests, and end-to-end tests.
- **Advantages:**
  - Annotations like `@Test`, `@BeforeClass`, `@AfterClass`, etc., to manage test lifecycle.
  - Support for parallel testing.
  - Flexible test configurations using XML.
  - Built-in reporting.

## Explain the TestNG annotations and their execution order.

### Answer:

- `@BeforeSuite` → `@BeforeTest` → `@BeforeClass` → `@BeforeMethod` → `@Test` → `@AfterMethod` → `@AfterClass` → `@AfterTest` → `@AfterSuite`
- `@BeforeSuite` and `@AfterSuite`: Execute before and after the entire test suite.
- `@BeforeTest` and `@AfterTest`: Execute before and after all methods in a test tag in XML.
- `@BeforeClass` and `@AfterClass`: Execute before and after each class.
- `@BeforeMethod` and `@AfterMethod`: Execute before and after each test method.

### Example:

```

public class TestNGExample {

    @BeforeSuite
    public void beforeSuite() {
        System.out.println("Before Suite");
    }

    @BeforeClass
    public void beforeClass() {
        System.out.println("Before Class");
    }

    @BeforeMethod
    public void beforeMethod() {
        System.out.println("Before Method");
    }

    @Test
    public void testMethod1() {
        System.out.println("Test Method 1");
    }

    @Test
    public void testMethod2() {
        System.out.println("Test Method 2");
    }

    @AfterMethod
    public void afterMethod() {
        System.out.println("After Method");
    }

    @AfterClass
    public void afterClass() {
        System.out.println("After Class");
    }

    @AfterSuite
    public void afterSuite() {
        System.out.println("After Suite");
    }

}

```

## Output:

```

pgsql
CopyEdit
Before Suite
Before Class
Before Method
Test Method 1
After Method
Before Method
Test Method 2
After Method
After Class
After Suite

```

## What is Functional Testing, and how is it different from Non-Functional Testing?

### Answer:

- **Functional Testing** checks the functionality of the software system. It ensures that the application behaves as expected according to the requirements.
- It involves testing user interactions, APIs, databases, security, and other application components.
- **Non-Functional Testing** focuses on aspects like performance, usability, reliability, and security.

### Examples:

- **Functional Testing:** Checking login functionality, form submissions, CRUD operations, etc.
- **Non-Functional Testing:** Checking response time, load capacity, scalability, etc.

## What are the different types of Functional Testing?

### Answer:

1. **Unit Testing:** Testing individual components or modules.
2. **Integration Testing:** Testing interactions between integrated modules.
3. **System Testing:** End-to-end testing of the complete system.
4. **User Acceptance Testing (UAT):** Validation from an end-user perspective.
5. **Smoke Testing:** Basic testing to ensure the application is stable enough for further testing.
6. **Sanity Testing:** Verifying specific functionalities after minor changes.
7. **Regression Testing:** Ensuring that new changes don't affect existing functionalities.

## How do you prioritize test cases?

### Answer:

Test case prioritization depends on:

- **Business Impact:** High-impact functionalities are tested first.
- **Critical Path:** Core application workflows that are frequently used.
- **Risk and Complexity:** Features prone to defects or complex integrations.
- **New and Modified Features:** Newly developed or recently modified features.
- **Customer Requirements:** Based on customer priority or critical needs.

### Example Prioritization Matrix:

Priority Level	Criteria	Example
High	Business-critical, core workflows	Payment gateway, user authentication
Medium	Important but not critical	User profile update, notifications
Low	Minor features, less used modules	Footer links, social media sharing

## What is the Defect Life Cycle?

### Answer:

The defect life cycle represents the stages a defect goes through during its lifetime.

### Typical Defect Life Cycle:

1. **New:** When a defect is initially found and logged.
2. **Assigned:** Assigned to a developer for analysis.
3. **Open:** Developer starts working on the fix.
4. **Fixed:** The defect is resolved by the developer.
5. **Retest:** Tester retests the fix.
6. **Verified:** The defect is confirmed as fixed.
7. **Closed:** If the defect is no longer present.
8. **Reopened:** If the defect persists after the fix.
9. **Deferred:** If the defect is postponed to future releases.
10. **Rejected:** If it is not considered a defect or is a duplicate.

## How do you ensure good test coverage?

### Answer:

1. **Requirement Traceability Matrix (RTM):** Ensures all requirements are covered by test cases.
2. **Test Design Techniques:**
  - **Equivalence Partitioning:** Grouping input data with similar behavior.
  - **Boundary Value Analysis:** Testing at the boundaries of input ranges.
  - **Decision Table Testing:** Covering all possible input combinations.
  - **State Transition Testing:** For systems with state-based behavior.
3. **Code Coverage Tools:** To check statement, branch, and path coverage.
4. **Peer Review and Walkthroughs:** Collaborative review of test cases.
5. **Exploratory Testing:** Unscripted testing to discover edge cases.

## Scenario: Automating a Complex Web Form

**Question:** How would you automate a complex web form with dynamic fields that appear based on user input?

### Answer:

1. **Identify Dynamic Elements:** Use locators like XPath or CSS Selectors that can adapt to dynamic elements.
2. **Use Explicit Waits:** Use Selenium's `WebDriverWait` to wait for dynamic fields to appear.
3. **Handle Conditional Logic:** Implement conditions to check for field visibility before interacting.
4. **Data-Driven Approach:** Use data sets to test multiple input combinations.
5. **Modular Design:** Create reusable methods for interacting with form fields.

## Scenario: Failing Test Cases Due to Timing Issues

**Question:** Your Selenium tests fail intermittently due to timing issues. How do you resolve this?

**Answer:**

1. **Use Explicit Waits:** Instead of fixed sleep times, use `WebDriverWait` to wait for conditions.
2. **Implement Fluent Waits:** To check for conditions at regular intervals.
3. **Optimize Page Loads:** Verify if slow page loads are causing the issue.
4. **Debugging:** Use screenshots and logs to identify where the failure occurs.
5. **Review Dependencies:** Check if third-party services are causing delays.

**Question:** How do you handle test case maintenance when the application UI changes frequently?

**Answer:**

1. **Use Robust Locators:** Prefer `XPath` and `CSS Selectors` with minimal dependency on attributes that frequently change.
2. **Page Object Model (POM):** Centralize locators in page classes to update them in one place.
3. **Custom Keywords in Robot Framework:** Create custom keywords to abstract away locator changes.
4. **Version Control:** Maintain test scripts in Git to track changes.
5. **Regular Refactoring:** Schedule periodic reviews to refactor and optimize test scripts.

**How do you handle tight deadlines when delivering test automation scripts?**

**Answer:**

- Prioritize test cases based on business impact and critical paths.
- Focus on automating high-priority and repetitive test cases first.
- Communicate transparently with stakeholders about feasible deliverables.
- Collaborate with developers to identify and resolve blockers early.
- If necessary, execute a mix of manual and automated testing to meet deadlines.

**Describe a challenging bug you found and how you resolved it.**

**Answer:**

- Clearly describe the bug, including the impact on functionality.
- Explain the investigation steps, such as analyzing logs, debugging scripts, or reproducing the issue.
- Detail the root cause analysis and how you collaborated with the development team.
- Mention the verification steps after the fix and how you ensured it didn't affect other functionalities (regression testing).

## How do you handle dynamic dropdowns in Selenium?

### Answer:

- Dynamic dropdowns populate options based on user actions like clicking or typing.
- Use `WebDriverWait` with `ExpectedConditions` to wait for options to appear.

### Example in Python:

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

driver = webdriver.Chrome()
driver.get("https://example.com")

# Click on the dropdown to display options
dropdown = driver.find_element(By.ID, "dropdownMenu")
dropdown.click()

# Wait for options to load and then select an option
option = WebDriverWait(driver, 10).until(
    EC.visibility_of_element_located((By.XPATH, "//ul[@id='dropdownOptions']/li[text()='Option 2']"))
)
option.click()
```

## How do you create custom keywords in Robot Framework using Python?

### Answer:

- Custom keywords are created by writing Python functions and importing them as libraries in Robot Framework.
- This allows for more complex logic and reuse of code.

### Example:

```
python
CopyEdit
# File: custom_keywords.py
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

def wait_for_element(driver, locator, timeout=10):
    WebDriverWait(driver,
        timeout).until(EC.visibility_of_element_located((By.XPATH, locator)))

# In the Robot Framework test case
*** Settings ***
Library    custom_keywords.py
```

```

*** Test Cases ***
Test Custom Keyword
    Open Browser    https://example.com    Chrome
    Wait For Element    //input[@id='username']
    Close Browser

```

- Save the Python file and import it as a library in the Robot Framework script.
- This approach is useful for complex scenarios not covered by built-in keywords.

## How do you handle dynamic web elements in Robot Framework?

### Answer:

- Dynamic web elements require flexible locators and waits.
- Use `Wait Until Element Is Visible` or `Wait Until Element Contains` keywords for dynamic content.

### Example:

```

robot
CopyEdit
*** Settings ***
Library    SeleniumLibrary

*** Variables ***
${DYNAMIC_LOCATOR}    //div[contains(text(),'Dynamic Text')]

*** Test Cases ***
Handle Dynamic Element
    Open Browser    https://example.com    Chrome
    Wait Until Element Is Visible    ${DYNAMIC_LOCATOR}    timeout=10
    Click Element    ${DYNAMIC_LOCATOR}
    Close Browser

```

- Use XPath functions like `contains()` or `starts-with()` for flexible locators.
- Adjust the timeout to handle dynamic loading times.

## How do you integrate Robot Framework with Jenkins for CI/CD?

### Answer:

1. **Install Robot Framework Plugin:** In Jenkins, install the Robot Framework plugin for report generation.
2. **Create a Jenkins Job:**
  - Choose **Freestyle Project** or **Pipeline**.
  - In the **Build** section, use a shell or batch command to run Robot tests.

### Example Shell Script:

```

sh
CopyEdit
pip install -r requirements.txt
robot -d Results TestCases/

```

### 3. Post-Build Actions:

- Add **Publish Robot Framework Report** to visualize the test results.

### 4. Trigger Options:

- Set up triggers for SCM commits, scheduled builds, or after a successful deployment.

### 5. Pipeline Script Example:

```
groovy
CopyEdit
pipeline {
    agent any
    stages {
        stage('Install Dependencies') {
            steps {
                sh 'pip install -r requirements.txt'
            }
        }
        stage('Run Robot Tests') {
            steps {
                sh 'robot -d Results TestCases/'
            }
        }
    }
    post {
        always {
            robot outputPath: 'Results'
        }
    }
}
```

## How do you handle data-driven testing in Robot Framework?

### Answer:

- Use the `DataDriver` library for parameterized testing.
- Test data can be provided via CSV, Excel, or JSON files.

### Example with CSV:

```
robot
CopyEdit
*** Settings ***
Library           SeleniumLibrary
Library           DataDriver      file=TestData.csv

*** Variables ***
${URL}            https://example.com

*** Test Cases ***
Login Test
    Open Browser    ${URL}        Chrome
    Input Text      id=username    ${username}
    Input Text      id=password    ${password}
    Click Button     id=login
    Page Should Contain    ${expected_message}
    Close Browser
```



### TestData.csv:

```
pgsql
CopyEdit
username,password,expected_message
user1,pass1>Welcome User 1
user2,pass2>Welcome User 2
```

- Each row of the CSV file is used as a set of test data.
- This reduces repetition and improves maintainability.

### How do you debug failing test cases in Robot Framework?

#### Answer:

1. **Screenshots on Failure:**
  - Use `Capture Page Screenshot` in the teardown to capture screenshots on failure.
2. **Logs and Reports:**
  - Review `log.html` and `report.html` for detailed execution logs.
3. **Run Tests in Debug Mode:**
  - Use `--loglevel DEBUG` to get more detailed logs.
4. **Breakpoints:**
  - Use `BuiltIn.Pause` to pause execution and investigate the state of the browser.
5. **Verbose Output:**
  - Use `--console verbose` to print detailed console output.

#### Example:

```
robot
CopyEdit
*** Test Cases ***
Debugging Example
    Open Browser    https://example.com    Chrome
    Pause          # Pauses execution for manual inspection
    Close Browser
```

- This is useful for troubleshooting locators or dynamic content issues.

### What is Dependency Testing in TestNG?

#### Answer:

- TestNG allows you to specify dependencies between test methods using the `dependsOnMethods` attribute.
- If a dependent method fails, the dependent test is skipped.

#### Example:

```
java
CopyEdit
```

```

@Test
public void openBrowser() {
    System.out.println("Browser Opened");
}

@Test(dependsOnMethods = { "openBrowser" })
public void login() {
    System.out.println("User Logged In");
}

```

### Output:

```

pgsql
CopyEdit
Browser Opened
User Logged In

```

- If `openBrowser()` fails, `login()` is skipped.

## How do you perform Data-Driven Testing in TestNG using DataProvider?

### Answer:

- `@DataProvider` is used to provide multiple sets of data to a test method.
- It allows parameterization without external data files.

### Example:

```

java
CopyEdit
@DataProvider(name = "loginData")
public Object[][] loginData() {
    return new Object[][] {
        { "user1", "pass1" },
        { "user2", "pass2" }
    };
}

@Test(dataProvider = "loginData")
public void loginTest(String username, String password) {
    System.out.println("Username: " + username + ", Password: " +
password);
}

```

### Output:

```

yaml
CopyEdit
Username: user1, Password: pass1
Username: user2, Password: pass2

```

- This enables data-driven testing with minimal code duplication.

## How do you control test execution order in TestNG?

### Answer:

1. **Priority:** Using `priority` attribute. Lower priority values are executed first.
2. **DependsOnMethods:** Test order based on dependencies.
3. **TestNG XML:** Specify the order in the `<test>` and `<classes>` tags.

### Example:

```
java
CopyEdit
@Test(priority = 1)
public void testA() {
    System.out.println("Test A");
}

@Test(priority = 2)
public void testB() {
    System.out.println("Test B");
}

@Test(priority = 3)
public void testC() {
    System.out.println("Test C");
}
```

### Output:

```
css
CopyEdit
Test A
Test B
Test C
```

- Tests execute in the order of `priority`.

## What is the difference between Functional and Non-Functional Testing?

### Answer:

- **Functional Testing:**
  - Focuses on verifying that the application behaves according to the requirements and specifications.
  - Checks "what" the system does.
  - Examples: Unit Testing, Integration Testing, System Testing, User Acceptance Testing.
- **Non-Functional Testing:**
  - Evaluates aspects like performance, usability, reliability, and security.
  - Checks "how well" the system performs.
  - Examples: Performance Testing, Load Testing, Security Testing, Usability Testing.

### Comparison Table:

Aspect	Functional Testing	Non-Functional Testing
Purpose	Verifies the correctness of features	Evaluates performance and usability
Focus	User actions and system responses	System behavior under various conditions
Testing Types	Unit, Integration, System, UAT	Performance, Security, Usability
Tools	Selenium, Robot Framework	JMeter, LoadRunner, Burp Suite

## What is Smoke Testing vs. Sanity Testing?

**Answer:**

- **Smoke Testing:**
  - Performed on initial builds to check the critical functionalities.
  - A high-level test to ensure the application is stable enough for further testing.
  - Also known as **Build Verification Testing**.
- **Sanity Testing:**
  - Conducted after minor changes or bug fixes.
  - Checks specific functionalities to confirm they work as expected.
  - Ensures that new changes did not introduce new defects.

**Example Scenario:**

- **Smoke Test:** Verify if the application launches, login works, and main navigation is functional.
- **Sanity Test:** After a bug fix in the cart module, check if adding/removing items works without testing the entire application.

## What is Regression Testing? When is it performed?

**Answer:**

- Regression Testing verifies that recent code changes did not adversely affect the existing functionalities.
- It ensures the stability of the application after enhancements, bug fixes, or configuration changes.
- **When to Perform:**
  - After bug fixes.
  - After enhancements or new features.
  - During continuous integration cycles.

**Types of Regression Testing:**

1. **Corrective Regression Testing:** No changes in the product's specification.
2. **Retest-all Regression Testing:** Checks the entire application. Time-consuming but thorough.

3. **Selective Regression Testing:** Tests the impacted areas using dependency mapping.
4. **Progressive Regression Testing:** Done when the product's specifications are updated.
5. **Complete Regression Testing:** Done after major changes in the application.

## What is the difference between Verification and Validation?

**Answer:**

- **Verification:**
  - Evaluates work-products like design, code, requirements, and specifications.
  - Ensures the product is being built correctly.
  - **"Are we building the product right?"**
  - Techniques: Reviews, Walkthroughs, Inspections.
- **Validation:**
  - Evaluates the final product to check if it meets user requirements.
  - Ensures the right product is built.
  - **"Are we building the right product?"**
  - Techniques: Testing (Unit, Integration, System, UAT).

**Example:**

- **Verification:** Checking the design documents for consistency.
- **Validation:** Testing the application to see if it meets the end-user needs.

## What is Exploratory Testing? When is it used?

**Answer:**

- Exploratory Testing is an informal testing approach where the tester actively explores the application without predefined test cases.
- It combines test design and execution on the fly.
- Testers simultaneously learn the application, design test cases, and execute them.
- **When to Use:**
  - When there is insufficient documentation.
  - When time is limited.
  - In Agile environments to quickly identify defects.
  - During ad-hoc testing sessions.

**Example:**

- Testing a new feature by randomly navigating, inputting data, and observing the behavior.
- Useful in finding UI/UX inconsistencies and edge cases.

## How would you test an e-commerce checkout process?

**Answer:**

- **Positive Test Cases:**
  - Successful login and checkout with valid credentials.
  - Adding items to the cart and updating quantities.
  - Applying valid coupons and checking discounts.
  - Selecting different payment methods (Credit Card, PayPal, etc.).
  - Completing the checkout and receiving a confirmation email.
- **Negative Test Cases:**
  - Invalid payment details (e.g., expired card, wrong CVV).
  - Empty cart checkout attempt.
  - Applying expired or invalid coupons.
  - Network failure during payment.
  - Invalid shipping address format.
- **Edge Cases:**
  - Maximum allowed quantity per product.
  - Special characters in the shipping address.
  - Multiple currency conversions.
  - Session timeout during checkout.
- **Non-Functional Testing:**
  - Performance under peak loads.
  - Security testing for payment data.
  - Usability testing for the checkout flow.

## How would you approach testing a REST API?

**Answer:**

- **Functionality Testing:**
  - Validate endpoints with different HTTP methods (GET, POST, PUT, DELETE).
  - Check response status codes (200, 400, 401, 404, 500).
  - Validate request and response headers.
  - Test for mandatory and optional fields.
- **Negative Testing:**
  - Invalid input parameters.
  - Unauthorized access (without tokens).
  - Invalid request formats (XML instead of JSON).
- **Performance Testing:**
  - Check response time under normal and peak loads.
  - Test API throttling and rate limits.
- **Security Testing:**
  - Check for SQL injection and XSS vulnerabilities.
  - Ensure sensitive data is encrypted.
  - Test authentication (OAuth, JWT).
- **Automation Tools:**
  - Postman for manual and automated testing.
  - RestAssured for Java-based automation.
  - Robot Framework with `RequestsLibrary`.

## Describe a challenging bug you encountered. How did you resolve it?

**Answer:**

- **Situation:** Encountered a critical bug in the checkout flow during a release cycle.
- **Task:** Ensure a seamless user experience without delaying the release.
- **Action:**
  - Analyzed logs and identified a race condition in API calls.
  - Collaborated with developers to reproduce and debug the issue.
  - Suggested an asynchronous retry mechanism to handle the race condition.
  - Tested the fix thoroughly with edge cases and regression tests.
- **Result:**
  - The issue was resolved without impacting the release timeline.
  - Improved the checkout flow's reliability and user satisfaction.

## How do you prioritize test cases when time is limited?

**Answer:**

- **Critical Path Analysis:** Identify high-priority functionalities critical for the business.
- **Risk-Based Testing:** Focus on areas with high risk and impact on users.
- **Customer-Facing Features:** Prioritize features directly affecting end-users.
- **Defect Prone Areas:** Test modules with a history of frequent bugs.
- **Compliance Requirements:** Ensure compliance-related functionalities are verified.

## How would you test a login page?

**Answer:**

- **Positive Test Cases:**
  - Valid username and valid password.
  - Remember me option functionality.
  - Login with different user roles (Admin, User, Guest).
- **Negative Test Cases:**
  - Invalid username and password.
  - Empty username and password fields.
  - Special characters or SQL injection attempts.
  - Exceeding maximum allowed characters.
  - Incorrect password multiple times (account lockout scenario).
- **Security Testing:**
  - Check for SQL Injection.
  - Validate HTTPS and secure cookies.
  - Test for Brute Force attack protection.
- **Usability Testing:**
  - Check error messages for accuracy and clarity.
  - Validate the alignment of UI elements.

- Test tab order and keyboard accessibility.
- **Performance Testing:**
  - Response time after entering credentials.
  - Concurrent login attempts with load testing tools.
- **Compatibility Testing:**
  - Across different browsers (Chrome, Firefox, Safari).
  - On mobile and desktop devices.

## How would you test a search functionality?

**Answer:**

- **Positive Test Cases:**
  - Search with exact keywords.
  - Search with partial keywords or auto-suggest.
  - Case-insensitive searches.
  - Search with special characters and numbers.
  - Sorting and filtering of search results.
- **Negative Test Cases:**
  - Search with no input (blank search).
  - Invalid or random characters.
  - SQL injection attempts.
- **Edge Cases:**
  - Single character search.
  - Maximum allowed characters.
  - Search with leading and trailing spaces.
  - Search with special characters (e.g., #, @, &, %).
- **Performance Testing:**
  - Response time with a large dataset.
  - Load testing with multiple concurrent users.
- **Usability Testing:**
  - Relevance of search results.
  - Auto-suggestions and error messages.
  - Pagination and navigation.
- **Security Testing:**
  - Prevention of SQL injection.
  - Data integrity and confidentiality.

## How would you test a REST API for a payment gateway?

**Answer:**

- **Functional Testing:**
  - Positive scenarios with valid card details.
  - Negative scenarios with invalid details (e.g., expired card, insufficient funds).
  - Mandatory fields validation (card number, CVV, expiration date).
  - Validations for optional fields (e.g., promo codes).
- **Error Handling:**
  - Proper status codes (200, 400, 401, 403, 500).
  - Error messages in the response body.
- **Security Testing:**



- Ensure data encryption (HTTPS).
  - Validate token-based authentication (JWT, OAuth).
  - Prevent SQL injection and XSS.
- **Performance Testing:**
  - Response time and processing time.
  - Load testing with multiple payment requests.
- **Edge Cases:**
  - Maximum and minimum payment amounts.
  - Currency conversion accuracy.
- **Integration Testing:**
  - Verify communication between frontend and backend.
  - Test integration with third-party payment providers.
- **Tools:**
  - Postman, RestAssured (Java), or Robot Framework with `RequestsLibrary`.

## Describe a situation where you missed a bug. How did you handle it?

### Answer:

- **Situation:** During a release, a critical bug related to payment processing was found in production.
- **Task:** Investigate the issue, fix it urgently, and ensure it doesn't recur.
- **Action:**
  - Conducted a root cause analysis and found a missing edge case in the test scenarios.
  - Collaborated with the development team to reproduce the issue.
  - Created new test cases to cover the missed scenarios.
  - Implemented regression tests to verify the fix.
  - Documented the incident and improved the review process.
- **Result:**
  - The issue was resolved quickly, minimizing impact on users.
  - Enhanced the overall test coverage and test review process.

### Key Takeaways:

- Acknowledge the mistake without blaming others.
- Demonstrate accountability and a proactive approach to resolve the issue.
- Show how you improved the process to prevent future occurrences.

## How do you handle conflicts within your team?

### Answer:

- **Stay Calm and Listen:** Allow all parties to express their views without interruption.
- **Understand the Root Cause:** Identify the underlying issue by asking open-ended questions.
- **Seek Common Ground:** Focus on shared goals and objectives.
- **Collaborate on Solutions:** Encourage a brainstorming session for a mutually beneficial solution.
- **Communicate Clearly:** Maintain clear and respectful communication throughout the discussion.
- **Involve a Mediator (if needed):** If the conflict escalates, involve a team lead or manager.

### Example:

- A conflict arose when a developer disagreed with the severity of a defect.
- I scheduled a meeting to understand the developer's perspective.
- We reviewed the impact on the user experience and agreed on the priority.
- Documented the decision and shared it with the team for transparency.

### Key Takeaways:

- Show emotional intelligence by managing emotions and maintaining a positive attitude.
- Demonstrate conflict resolution skills with a focus on collaboration.

## How do you stay updated with the latest testing tools and technologies?

### Answer:

- **Online Learning Platforms:** Regularly take courses on platforms like Udemy, Coursera, or Pluralsight.
- **Webinars and Conferences:** Attend industry webinars and conferences like Selenium Conference, TestCon, etc.
- **Communities and Forums:** Participate in QA communities like Stack Overflow, Reddit, or Test Automation University.
- **Blogs and Newsletters:** Subscribe to industry-leading blogs like Ministry of Testing, Test Automation Blog.
- **Hands-on Practice:** Experiment with new tools in personal projects or Proof of Concepts (POCs).
- **Networking:** Connect with other QA professionals on LinkedIn or through local meetups.

## What are the different levels of testing?

### Answer:

1. **Unit Testing:**
  - Tests individual units or components.
  - Performed by developers using frameworks like JUnit, NUnit, or PyTest.
2. **Integration Testing:**
  - Tests interactions between integrated modules.
  - Ensures modules work together as expected.
3. **System Testing:**
  - Tests the complete application as a whole.
  - Ensures the system meets functional and non-functional requirements.
4. **User Acceptance Testing (UAT):**
  - Performed by end-users or clients.
  - Ensures the application meets business requirements and is ready for production.

## What is the Test Life Cycle?

### Answer:

1. **Requirement Analysis:**
  - Understand and analyze requirements.
  - Identify testable requirements.
2. **Test Planning:**
  - Create a test plan, define scope, objectives, resources, and schedule.
  - Identify test tools and environment setup.
3. **Test Design:**
  - Create test cases, test scripts, and test data.
  - Review and baseline test cases.
4. **Test Environment Setup:**
  - Prepare the testing environment.
  - Install software and configure the system for testing.
5. **Test Execution:**
  - Execute test cases and report defects.
  - Track and retest defects once fixed.
6. **Test Closure:**
  - Evaluate exit criteria and generate test closure reports.
  - Document lessons learned and archive test artifacts.

## What is a Test Case, and what are its key components?

### Answer:

- **Test Case:**
  - A set of actions executed to verify a particular feature or functionality of an application.
  - It includes preconditions, inputs, actions, expected results, and postconditions.

### Key Components of a Test Case:

1. **Test Case ID:** Unique identifier for the test case.
2. **Test Description:** A brief explanation of the test case objective.
3. **Preconditions:** Any prerequisite conditions that must be met before test execution.
4. **Test Steps:** Detailed steps to execute the test.
5. **Test Data:** Data required for test execution.
6. **Expected Result:** The anticipated outcome of the test.
7. **Actual Result:** The actual outcome after execution.
8. **Status:** Pass or Fail status based on comparison of expected and actual results.
9. **Priority:** Indicates the importance (High, Medium, Low).
10. **Attachments:** Screenshots or logs (if applicable).

### Example:

- **Test Case ID:** TC\_LOGIN\_001
- **Description:** Verify successful login with valid credentials.

- **Precondition:** User is on the login page.
- **Steps:**
  1. Enter valid username.
  2. Enter valid password.
  3. Click the login button.
- **Expected Result:** User should be redirected to the dashboard.
- **Actual Result:** As expected.
- **Status:** Pass

## What are Test Scenarios, and how do they differ from Test Cases?

### Answer:

- **Test Scenario:**
  - High-level documentation of what to test.
  - Focuses on the functionality or feature to be tested.
  - Example: "Verify the login functionality with different user roles."
- **Test Case:**
  - Detailed documentation of how to test a scenario.
  - Contains steps, inputs, and expected outcomes.
  - Example: "Test login with a valid admin account."

### Key Differences:

Aspect	Test Scenario	Test Case
Level of Detail	High-level, broad functionality	Detailed steps and expected results
Focus	What to test	How to test
Granularity	Coarse-grained	Fine-grained
Example	"Test login functionality"	"Test login with valid username and password"

## What is Defect Life Cycle?

### Answer:

- The **Defect Life Cycle** (also known as Bug Life Cycle) is the progression of a defect from its discovery to closure.
- It involves several stages:

### Stages of Defect Life Cycle:

1. **New:** When a defect is logged for the first time.
2. **Assigned:** Assigned to a developer for fixing.
3. **Open:** Developer starts analyzing and fixing the defect.

4. **Fixed:** Defect is resolved and ready for retesting.
5. **Retest:** QA retests the defect with the fix.
6. **Verified:** If the defect is fixed, it is marked as verified.
7. **Reopen:** If the defect persists, it is reopened.
8. **Closed:** If the defect is verified and fixed, it is closed.
9. **Deferred:** If the defect is low priority or will be fixed in a future release.
10. **Rejected:** If the defect is invalid or not reproducible.

### Example Flow:

- New → Assigned → Open → Fixed → Retest → Verified → Closed

## What are Test Metrics? Why are they important?

### Answer:

- **Test Metrics** are quantitative measures that help to assess the quality and effectiveness of testing activities.
- They provide insights into the testing process, test coverage, defect density, and overall project health.

### Types of Test Metrics:

1. **Product Metrics:** Measure the product's quality (e.g., Defect Density, Defect Severity).
2. **Process Metrics:** Evaluate the efficiency of the testing process (e.g., Test Execution Rate).
3. **Project Metrics:** Monitor the project's progress (e.g., Test Case Productivity).

### Examples:

- **Defect Density:** Number of defects per KLOC (thousand lines of code).
- **Test Execution Rate:** Percentage of test cases executed in a cycle.
- **Defect Leakage:** Number of defects found in production after release.

### Why Important:

- Measure testing efficiency and productivity.
- Identify areas of improvement.
- Track project health and quality.
- Help in decision-making and process optimization.

## What is a Test Automation Framework? Why is it important?

### Answer:

- **Test Automation Framework** is a set of guidelines, rules, and tools to create and maintain test scripts efficiently.
- It provides a standardized approach for designing, implementing, and executing automated tests.

### Key Benefits:

- Reusability of code.
- Maintainability and scalability of test scripts.
- Improved test accuracy and efficiency.
- Better test reporting and logging.
- Integration with CI/CD pipelines.

### **Types of Automation Frameworks:**

1. **Linear Scripting Framework:** Record and playback approach. Simple but not reusable.
2. **Modular Testing Framework:** Breaks tests into reusable modules.
3. **Data-Driven Framework:** Test data is externalized (e.g., Excel, CSV, DB).
4. **Keyword-Driven Framework:** Actions are driven by keywords in external files.
5. **Hybrid Framework:** Combines multiple frameworks for flexibility.
6. **Behavior-Driven Development (BDD) Framework:** Uses Gherkin syntax for test scenarios (e.g., Cucumber, Robot Framework).

### **What is Page Object Model (POM) in Selenium?**

#### **Answer:**

- **Page Object Model (POM)** is a design pattern that creates an object repository for web elements.
- It separates the UI elements and actions from test scripts.
- Enhances maintainability and reduces code duplication.

#### **Key Concepts:**

- **Page Class:** Contains web elements and methods to perform actions.
- **Test Class:** Contains test scripts that call methods from Page Classes.

#### **Advantages:**

- Promotes code reusability and maintainability.
- Enhances readability and reduces duplication.
- Isolates UI changes to Page Classes without affecting Test Classes.