Master List [Q&A]

3. **How do you achieve data-driven testing here?**
   Data-driven testing is achieved by separating test data from test scripts. This can be done using various methods such as:
   - Reading data from external sources like Excel, CSV files, or databases.
   - Using frameworks like Apache POI for Excel, or libraries like Jackson or Gson for JSON.
   - Utilizing TestNG's DataProvider annotation to pass different sets of data to test methods.

4. **When you talk about TestNG, how do you execute parallel execution?**
   Parallel execution in TestNG can be achieved by configuring the TestNG XML file. You can specify the parallel attribute and the thread-count attribute in the XML file to run test methods, classes, or suites in parallel. For example:
   ```xml
   <suite name="Suite" parallel="methods" thread-count="5">
     <test name="Test">
       <classes>
         <class name="com.example.TestClass"/>
       </classes>
     </test>
   </suite>
   ```

5. **Let's say you are performing regression testing. Imagine the framework you guys have already developed. How did you first form the regression test cases in TestNG, and how do you perform that?**
   Regression test cases are formed by identifying critical functionalities that need to be tested after any code changes. The process includes:
   - Identifying the core features and functionalities of the application.
   - Writing test cases that cover these features comprehensively.
   - Grouping the regression tests using TestNG’s @Test annotation and setting up groups in the TestNG XML file.
   - Running these tests using CI/CD tools to ensure they are executed regularly.

6. **Let's say there is a test method, and you have to run it two times. What are the annotations, and in the first case, you are logged in?**
   You can use the @Test annotation with the invocationCount attribute to run a test method multiple times. For handling the login scenario in the first run, you can use a boolean flag to manage the login state. For example:
   ```java
   @Test(invocationCount = 2)
   public void testMethod() {
   ```

```
    if (!isLoggedIn) {
        login();
        isLoggedIn = true;
    }
    // Test logic here
}
```

8. **While identifying a web element locator on a web page, it is not identifying it. What can be the scenarios?**
Several scenarios could cause this issue:
- The element is not yet rendered or is dynamically loaded (use explicit waits).
- The locator strategy is incorrect or outdated.
- The element is in an iframe or shadow DOM.
- The element is hidden or not visible.
- There are multiple elements with the same locator.

9. **Did you get a chance to work with frames?**
Yes, I have worked with frames. Handling frames involves switching to the frame using methods like driver.switchTo().frame(), performing actions within the frame, and then switching back to the default content using driver.switchTo().defaultContent().

10. **What is Fluent Wait?**
Fluent Wait is a type of explicit wait in Selenium that defines the maximum amount of time to wait for a condition, as well as the frequency with which to check the condition. It also allows ignoring specific exceptions while waiting. For example:
```java
Wait<WebDriver> wait = new FluentWait<>(driver)
    .withTimeout(Duration.ofSeconds(30))
    .pollingEvery(Duration.ofSeconds(5))
    .ignoring(NoSuchElementException.class);

WebElement element = wait.until(new Function<WebDriver, WebElement>() {
    public WebElement apply(WebDriver driver) {
        return driver.findElement(By.id("elementId"));
    }
});
```

11. **What kind of Action class methods have you worked with?**
I have worked with various Action class methods, including:
- `moveToElement()`: To hover over an element.
- `clickAndHold()`: To click and hold an element.

   - `dragAndDrop()`: To drag an element from one point and drop it at another.
   - `sendKeys()`: To send a sequence of keystrokes.
   - `release()`: To release a held mouse button.

12. **Do you know about POM (Page Object Model)? What OOP concept does it come under?**
   Yes, POM (Page Object Model) is a design pattern in Selenium that creates an object repository for web UI elements. It falls under the OOP concept of Encapsulation, as it encapsulates the page elements and actions within page classes.

13. **What is the objective of POM?**
   The objective of POM is to reduce code duplication and improve test maintenance. It helps to separate the test logic from the UI elements, making tests more readable, reusable, and easier to maintain.

14. **How do you declare variables in POM?**
   Variables in POM are typically declared as private WebElements using locators. For example:
   ```java
   @FindBy(id = "username")
   private WebElement username;

   @FindBy(id = "password")
   private WebElement password;
   ```

15. **How do you handle try-catch?**
   Try-catch is used to handle exceptions in Java. It allows the program to continue executing even if an exception occurs. For example:
   ```java
   try {
       // Code that might throw an exception
   } catch (Exception e) {
       // Handle the exception
       e.printStackTrace();
   }
   ```

16. **In Selenium, how is retry possible? What is the syntax?**
   Retry logic in Selenium can be implemented using TestNG’s IRetryAnalyzer. Here is an example:
   ```java
   public class RetryAnalyzer implements IRetryAnalyzer {
       private int retryCount = 0;
       private static final int maxRetryCount = 3;
   ```

```java
        @Override
        public boolean retry(ITestResult result) {
            if (retryCount < maxRetryCount) {
                retryCount++;
                return true;
            }
            return false;
        }
    }

    @Test(retryAnalyzer = RetryAnalyzer.class)
    public void testMethod() {
        // Test logic
    }
```

17. **For retrieving the status code, how is Rest Assured used? How do you validate that and by which method?**
   Rest Assured is used for testing RESTful web services. To retrieve and validate the status code, you can use:
```java
    Response response = given()
        .when()
        .get("http://example.com/api");

    int statusCode = response.getStatusCode();
    Assert.assertEquals(statusCode, 200);
```

18. **How do you achieve deserialization? What is meant by deserialization?**
   Deserialization is the process of converting JSON or XML data into a Java object. This can be achieved using libraries like Gson or Jackson. For example, with Gson:
```java
    Gson gson = new Gson();
    MyClass myObject = gson.fromJson(jsonString, MyClass.class);
```

19. **How good are you with CI/CD?**
   I have a solid understanding and practical experience with CI/CD pipelines. I have worked with tools like Jenkins, GitLab CI, and CircleCI to automate the build, test, and deployment processes, ensuring faster and more reliable software releases.

Selenium

Difference between close() and quit():
- `close()`: This method is used to close the current browser window that the WebDriver is focused on.
- `quit()`: This method closes all the browser windows opened by the WebDriver and ends the WebDriver session.

Different types of locators used to identify web elements:
1. **ID**: `driver.findElement(By.id("elementID"))`
2. **Name**: `driver.findElement(By.name("elementName"))`
3. **Class Name**: `driver.findElement(By.className("elementClassName"))`
4. **Tag Name**: `driver.findElement(By.tagName("elementTagName"))`
5. **Link Text**: `driver.findElement(By.linkText("Link Text"))`
6. **Partial Link Text**: `driver.findElement(By.partialLinkText("Partial Link Text"))`
7. **CSS Selector**: `driver.findElement(By.cssSelector("cssSelector"))`
8. **XPath**: `driver.findElement(By.xpath("xpathExpression"))`

Difference between findElement and findElements:
- `findElement`: Returns the first web element that matches the specified locator. If no element is found, it throws a `NoSuchElementException`.
- `findElements`: Returns a list of web elements that match the specified locator. If no elements are found, it returns an empty list.

Handle frames and windows in Selenium:
- **Frames**: Switch to a frame using `driver.switchTo().frame("frameName")`, perform actions, and switch back using `driver.switchTo().defaultContent()`.
- **Windows**: Handle multiple windows by switching between window handles. Use `driver.getWindowHandles()` to get all window handles and `driver.switchTo().window(windowHandle)` to switch.

Handle synchronization issues in Selenium tests:
- **Implicit Wait**: `driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(10))`
- **Explicit Wait**: Use WebDriverWait for specific conditions.
  ```java
  WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
  WebElement element =
wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("elementID")));
  ```
- **Fluent Wait**: Customize the wait with polling intervals and exception handling.

Limitations of Selenium:
- Cannot test desktop applications.
- Limited support for handling mobile applications.
- No built-in reporting capabilities.
- Requires programming knowledge.
- Browser-specific issues and inconsistencies.

What is WebDriverWait:
WebDriverWait is a class in Selenium used to wait for a certain condition to be met before proceeding with the execution. It is a type of explicit wait.

Java

Explain concepts like OOPs:
- **Object-Oriented Programming (OOP)**: A paradigm based on the concept of "objects" which can contain data and methods.
  - **Encapsulation**: Wrapping data and methods into a single unit (class).
  - **Inheritance**: Mechanism by which one class inherits the properties and behavior of another class.
  - **Polymorphism**: Ability of a variable, function, or object to take on multiple forms.
  - **Abstraction**: Hiding complex implementation details and showing only the necessary features.

Collections Framework:
Java Collections Framework provides architecture to store and manipulate a group of objects. Includes interfaces like List, Set, and Map, and classes like ArrayList, HashSet, and HashMap.

Inheritance & Polymorphism:
- **Inheritance**: Allows a class to inherit fields and methods from another class.
  ```java
  class Animal { void eat() { System.out.println("eating"); } }
  class Dog extends Animal { void bark() { System.out.println("barking"); } }
  ```
- **Polymorphism**: Allows methods to do different things based on the object it is acting upon.
  ```java
  Animal a = new Dog(); a.eat();  // Output: eating
  ```

Exception Handling:
Mechanism to handle runtime errors to maintain normal application flow using try, catch, finally, throw, and throws.
```java
try {
    // code that may throw exception
```

} catch (ExceptionType e) {
    // code to handle exception
} finally {
    // code that executes regardless of exception
}

Lambda Expressions:
Introduced in Java 8, lambda expressions provide a way to express instances of single-method interfaces (functional interfaces) more compactly.
```java
Runnable r = () -> System.out.println("Lambda expression example");
```

Difference between HashMap & HashTable:
- **HashMap**: Non-synchronized, allows one null key and multiple null values, faster.
- **HashTable**: Synchronized, doesn’t allow null keys or values, slower.

Program to sort an int array:
```java
int[] arr = {5, 2, 8, 1, 3};
Arrays.sort(arr);
System.out.println(Arrays.toString(arr));
```

Wrapper class in Java:
Wrapper classes provide a way to use primitive data types (int, boolean, etc.) as objects. Example: Integer, Boolean.
```java
int a = 5;
Integer aObj = Integer.valueOf(a);  // Boxing
int b = aObj.intValue();  // Unboxing
```

Differentiate between instance and local variables:
- **Instance variables**: Declared in a class, but outside a method, constructor, or block. They are associated with an instance of the class.
- **Local variables**: Declared within a method, constructor, or block. They are only accessible within that method or block.

Differences between String and StringBuffer with Example:
- **String**: Immutable, stored in the string pool.
- **StringBuffer**: Mutable, used for concatenation and modification of strings.
  ```java
  ```

  String str = "Hello";
  str.concat(" World"); // str remains "Hello"

  StringBuffer sb = new StringBuffer("Hello");
  sb.append(" World"); // sb becomes "Hello World"

Access Specifiers in Java:
- **Private**: Accessible only within the class.
- **Default**: Accessible within the same package.
- **Protected**: Accessible within the same package and subclasses.
- **Public**: Accessible from anywhere.

Types of constructors used in Java with Example:
- **Default constructor**: No arguments.
  ```java
  class Example {
      Example() { System.out.println("Default Constructor"); }
  }
  ```
- **Parameterized constructor**: Takes arguments.
  ```java
  class Example {
      Example(int a) { System.out.println("Parameterized Constructor"); }
  }
  ```

Differences between abstract class and interface with Example:
- **Abstract Class**: Can have both abstract and concrete methods. Can have instance variables.
  ```java
  abstract class Animal {
      abstract void sound();
      void eat() { System.out.println("eating"); }
  }
  ```
- **Interface**: Can only have abstract methods (Java 8 onwards can have default and static methods). Cannot have instance variables.
  ```java
  interface Animal {
      void sound();
  }
  ```

Explain System.out.println:
- **System**: A final class in java.lang package.
- **out**: A static member of the System class, which is an instance of PrintStream.
- **println**: A method of PrintStream class that prints the argument passed and a newline.

Cucumber

Keywords in Cucumber:
- **Feature**: Describes the feature under test.
- **Scenario**: Describes a particular test scenario.
- **Given**: Describes the initial context of the system.
- **When**: Describes an event or action.
- **Then**: Describes the expected outcome.
- **And**: Used to add additional steps.
- **But**: Used to add a step that introduces a contradiction.

What is Scenario Outline:
Scenario Outline is used to run the same Scenario multiple times with different sets of data. It is followed by Examples, which provide the data.
```gherkin
Scenario Outline: Test with multiple data
  Given I have a user with "<username>" and "<password>"
  Examples:
    | username | password |
    | user1    | pass1    |
    | user2    | pass2    |
```

Annotations in Cucumber:
- **@Given**: Marks a step as Given.
- **@When**: Marks a step as When.
- **@Then**: Marks a step as Then.
- **@And**: Marks a step as And.
- **@But**: Marks a step as But.

Hooks in Cucumber:
Hooks are blocks of code that run before or after each scenario. Annotations like @Before and @After are used.
```java
@Before
public void beforeScenario() {
    // Code to be executed before each scenario
}
```

@After
public void afterScenario() {
    // Code to be executed after each scenario
}

What are tags in Cucumber:
Tags are used to filter which tests to run. They are specified in the feature file and used with Cucumber options.

```gherkin
@smoke
Scenario: Test scenario
  Given ...
```
```java
@RunWith(Cucumber.class)
@CucumberOptions(tags = "@smoke")
public class TestRunner {
    // Code to run the tests
}
```

Cucumber Dry Run:
Dry Run is used to check the mapping between feature file and step definitions without actually executing the tests. It is specified using the `dryRun` option.
```java
@RunWith(Cucumber.class)
@CucumberOptions(dryRun = true)
public class TestRunner {
    // Code to run the tests
}
```

API Testing

Differences between API and Web Service:
- **API**: Application Programming Interface, a set of rules and protocols for building and interacting with software applications.
- **Web Service**: A type of API that operates over a network, usually HTTP, providing functionality to other systems.

Common protocols used in API testing:

- **HTTP/HTTPS**
- **SOAP**
- **REST**
- **gRPC**

Principles of an API test design:
- **Validation of endpoints**: Ensuring endpoints are accessible and return correct responses.
- **Schema validation**: Checking if the response adheres to a predefined schema.
- **CRUD operations**: Testing Create, Read, Update, Delete operations.
- **Error handling**: Ensuring the API handles errors correctly.
- **Performance testing**: Checking the performance and load handling of the API.

Agile

Important parts of the Agile process:
- **User Stories**: Short descriptions of a feature from an end-user perspective.
- **Sprint**: A time-boxed period for completing a set of user stories.
- **Scrum**: A framework for managing Agile projects.
- **Daily Stand-up**: A short daily meeting to discuss progress and impediments.
- **Retrospective**: A meeting held at the end of a sprint to discuss what went well and what can be improved.

Tools used in scrum planning:
- **JIRA**
- **Trello**
- **Asana**
- **VersionOne**
- **Monday.com**

DB Testing

What is a sub-query?
A sub-query is a query nested inside another query. It is used to perform operations in multiple steps.
```sql
SELECT * FROM employees WHERE salary > (SELECT AVG(salary) FROM employees);
```

Why do we use a GROUP clause?
The GROUP BY clause groups rows that have the same values in specified columns into summary rows. It is often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG).
```sql
SELECT department, COUNT(*) FROM employees GROUP BY department;
```

Difference between DELETE, DROP, and TRUNCATE commands:
- **DELETE**: Removes rows from a table based on a condition. Can be rolled back.
  ```sql
  DELETE FROM employees WHERE department = 'Sales';
  ```
- **DROP**: Removes an entire table or database. Cannot be rolled back.
  ```sql
  DROP TABLE employees;
  ```
- **TRUNCATE**: Removes all rows from a table without logging individual row deletions. Cannot be rolled back.
  ```sql
  TRUNCATE TABLE employees;
  ```

SQL Query to fetch 5th Highest Salary:
```sql
SELECT salary FROM (
    SELECT salary, DENSE_RANK() OVER (ORDER BY salary DESC) AS rank
    FROM employees
) WHERE rank = 5;
```

What framework(s) have you used in REST Assured?

In REST Assured, I've used frameworks that integrate with TestNG and JUnit for writing and managing test cases. I've also used Maven for project management and dependency handling. The framework is structured to include base classes for setup and teardown, utility classes for reusable methods, and specific test classes for different API endpoints.

Have you used Postman?

Yes, I have used Postman extensively for manual API testing. It helps in creating and running HTTP requests, validating responses, and automating tests using the Postman Collection Runner. I also use Postman for generating test scripts and documenting APIs.

In your previous positions, how did you receive your requirements in API testing?

Requirements for API testing are typically received from business analysts or product owners in the form of user stories or requirements documents. These requirements include details about the endpoints, request and response formats, expected behavior, and edge cases. Sometimes, API specifications are provided using Swagger or OpenAPI documentation.

Can you tell me about your project structure?

The project structure typically follows a modular approach:
- **src/main/java**: Contains the application source code.
- **src/test/java**: Contains test code, including unit tests, integration tests, and end-to-end tests.
- **src/test/resources**: Contains test data, configuration files, and test resources.
- **pom.xml**: Maven configuration file for managing project dependencies and build configuration.

Can you tell me how the Selenium framework works?

The Selenium framework works by automating web browsers to perform specific actions on web applications. It involves:
1. **WebDriver**: An interface to interact with different browsers.
2. **Locators**: Methods to locate web elements (ID, Name, XPath, CSS Selector).
3. **Page Object Model (POM)**: Design pattern to create an object repository for web elements.
4. **TestNG**: Framework for running test cases, managing test execution, and generating reports.
5. **Setup and Teardown**: Methods for initializing and closing browser sessions.
6. **Assertions**: Verifying test results using assertions.

What are the constructors in Java?

Constructors in Java are special methods used to initialize objects. They have the same name as the class and do not have a return type. Types of constructors:
- **Default Constructor**: No arguments.
  ```java
  class Example {
      Example() {
          System.out.println("Default Constructor");
      }
  }
  ```
- **Parameterized Constructor**: Takes parameters to initialize object properties.
  ```java
  class Example {
      int x;
      Example(int x) {
          this.x = x;
      }
  }
  ```

What is an interface? Why is it used?

An interface in Java is an abstract type that is used to specify a set of methods that a class must implement. It is used for:
- Achieving abstraction.
- Multiple inheritance in Java.
- Loose coupling by allowing classes to interact through the interface rather than concrete implementations.

```java
interface Animal {
    void sound();
}
class Dog implements Animal {
    public void sound() {
        System.out.println("Bark");
    }
}
```

Write a program to check if a number is prime or not.

```java
public class PrimeCheck {
    public static boolean isPrime(int num) {
        if (num <= 1) return false;
        for (int i = 2; i <= Math.sqrt(num); i++) {
            if (num % i == 0) return false;
        }
        return true;
    }

    public static void main(String[] args) {
        int num = 29;
        System.out.println(isPrime(num) ? num + " is a prime number" : num + " is not a prime
number");
    }
}
```

Input/Output logic for A, B, C:

```java
```

```
public class ColorPrint {
    public static void main(String[] args) {
        char input = 'A'; // This can be changed to 'A', 'B', or other values

        switch (input) {
            case 'A':
                System.out.println("RED");
                break;
            case 'B':
                System.out.println("GREEN");
                break;
            default:
                System.out.println("BLACK");
                break;
        }
    }
}
```

How do you handle hidden elements in Selenium?

To handle hidden elements in Selenium, you can use JavaScript to interact with them or wait until they become visible.
```java
WebElement hiddenElement = driver.findElement(By.id("hiddenElement"));
((JavascriptExecutor) driver).executeScript("arguments[0].click();", hiddenElement);
```

What is the biggest challenge you have faced while writing Selenium scripts?

The biggest challenge is handling dynamic elements and synchronization issues. Elements that change frequently or load asynchronously can cause tests to fail intermittently. Using explicit waits and writing robust locators helps mitigate these challenges.

When is a repository used?

A repository is used to manage code versions, facilitate collaboration among team members, and maintain a history of changes. Tools like Git and Bitbucket are commonly used for this purpose.

Say you have a few files on your machine that you want to change. How will you do it?

To change files on a machine, you can:
- Open the files in an IDE or text editor.

- Make the necessary changes.
- Save the changes.
- If using version control, commit and push the changes to the repository.

Is there any way to avoid any kind of conflicts?

To avoid conflicts:
- Frequently pull changes from the main branch.
- Communicate with team members about ongoing changes.
- Resolve conflicts promptly by merging or rebasing.
- Use feature branches for isolated development.

Which regression suite do you currently have?

The current regression suite includes automated test cases for critical functionalities of the application. These tests cover login, user management, data processing, and reporting modules, ensuring that core features work as expected after code changes.

Do you have any experience with Sanity Suite?

Yes, I have experience with sanity testing. A sanity suite includes a subset of regression tests focusing on verifying specific functionalities after minor changes or bug fixes, ensuring the application is stable for further testing.

Have you ever been involved in code reviewing?

Yes, I have participated in code reviews, which involve reviewing code changes submitted by peers to ensure quality, consistency, and adherence to coding standards.

What are the items you generally focus on while reviewing?

While reviewing code, I focus on:
- Code readability and maintainability.
- Adherence to coding standards and best practices.
- Proper use of design patterns.
- Efficient and optimized code.
- Adequate test coverage and documentation.

How do you know if the test cases have passed or failed?

Test case results are determined by assertions in the test scripts. In automated tests, frameworks like TestNG generate reports indicating pass or fail status based on these assertions.

What are the failure codes you have had?

Common failure codes include:
- `NoSuchElementException`: Element not found.
- `TimeoutException`: Operation timed out.
- `AssertionError`: Test assertion failed.

Have you ever used Parasoft or Katalon?

I have experience with Katalon Studio, a tool for automated testing of web and mobile applications. It provides a user-friendly interface and integrates well with various test automation frameworks.

How do you ensure there is traceability in Jira?

Traceability in Jira is ensured by:
- Linking test cases to user stories or requirements.
- Using issue types like Epics, Stories, and Tasks.
- Maintaining a clear workflow and status for each issue.
- Generating traceability reports and matrices.

Assume you are at the beginning of a sprint. Once you have identified test cases for automation, how do you shortlist?

To shortlist test cases for automation:
- Prioritize tests based on criticality and frequency of use.
- Identify repetitive and time-consuming manual tests.
- Consider the stability of the application area.
- Ensure test cases have clear expected results.

Suppose that while running a regression test, 50% of the test cases fail. How do you do the failure analysis?

For failure analysis:
- Review failure logs and screenshots.
- Identify patterns or common causes of failures.
- Determine if failures are due to application bugs, test script issues, or environment problems.
- Collaborate with developers to fix bugs.
- Update and improve test scripts as needed.

What is your approach to automating testing?

My approach to automating testing includes:
- Analyzing the application to identify test automation opportunities.

- Selecting the right tools and frameworks.
- Designing a scalable and maintainable test automation framework.
- Writing reusable and modular test scripts.
- Integrating with CI/CD pipelines for continuous testing.
- Regularly updating and maintaining automated tests.

Have you handled both functional and automation testing? How do you design in priority?

Yes, I have handled both functional and automation testing. Prioritization is based on:
- Business criticality of the feature.
- Frequency of use.
- Impact on end-users.
- Complexity and feasibility of automation.
- Return on investment for automation.

Assumptions and Branch Maintenance

Assume there are 3 resources, one resource submits the PR and one is on leave. At that point, you are not able to merge. What would you do in that situation?
- **Action Plan**:
  1. **Review the PR**: Check if the PR meets the merging criteria and if any additional changes are required.
  2. **Communication**: Communicate with the resource who is on leave and see if an urgent merge can be done.
  3. **Backup Reviewer**: If the above two steps do not resolve the issue, seek assistance from another team member who can review and merge the PR.
  4. **Documentation**: Document the reasons for the delay and any steps taken to resolve the situation for future reference.

Do you maintain the branches in test cases?
- **Yes**: Maintaining branches in test cases ensures that different versions of the test cases align with the corresponding application code versions. This practice supports parallel development and helps in testing specific features or bug fixes in isolation.

Feedback and Usability in Automation Scripts

What are some recent comments you have given to your team members as feedback on their automation scripts?
- **Code Readability**: "Ensure variable names are descriptive enough to make the code more readable."
- **Reusability**: "Refactor this method to make it reusable across multiple test cases."
- **Exception Handling**: "Add proper exception handling to avoid test failures due to unforeseen issues."

- **Logging**: "Include more logging to make debugging easier."

How do you ensure the usability of the methods in review?
- **Code Review Checklists**: Use checklists to ensure the methods are reusable, modular, and adhere to coding standards.
- **Code Reusability**: Verify that the methods are designed to be reused across different test scenarios.
- **Documentation**: Ensure methods are well-documented with comments explaining their purpose and usage.

Do you ensure the validation in the test cases in automation scripts?
- **Yes**: Validation is ensured by using assertions in the test scripts to verify the expected outcomes. Each test case includes checks to validate that the application behaves as expected under various conditions.

Framework Creation and Cucumber

You have some exposure to framework creation. Assume that you are tasked with creating a framework for a web application like Amazon. How will you break down a course of action?
1. **Requirement Analysis**: Understand the application's key features and user flows.
2. **Tool Selection**: Choose tools and technologies (e.g., Selenium, TestNG, Maven).
3. **Framework Design**: Design a scalable architecture using design patterns like Page Object Model (POM).
4. **Setup Environment**: Set up the development environment with necessary dependencies and configurations.
5. **Implement Core Components**: Develop core components like base classes, utility functions, and common actions.
6. **Write Test Cases**: Start writing test cases for key functionalities.
7. **Integration with CI/CD**: Integrate the framework with Jenkins or other CI/CD tools for automated test execution.
8. **Reporting and Logging**: Implement reporting and logging mechanisms to track test results and issues.
9. **Review and Refine**: Continuously review and refine the framework based on feedback and new requirements.

Let's consider the Cucumber framework. In the existing framework, what would be needed to accommodate Cucumber?
1. **Add Dependencies**: Include Cucumber dependencies in the build file (e.g., Maven or Gradle).
2. **Feature Files**: Create feature files with scenarios written in Gherkin language.
3. **Step Definitions**: Implement step definitions to link Gherkin steps with test code.
4. **Runner Class**: Create a test runner class using JUnit or TestNG to run Cucumber tests.
5. **Configuration**: Configure Cucumber options for reporting, tags, and glue paths.

Test Case Execution and API Testing

How do you access test case execution? You need to capture the screenshot when it fails. How can you accommodate that?
- **TestNG Listeners**: Implement TestNG listeners (e.g., ITestListener) to capture screenshots on test failure.
  ```java
  public class TestListener implements ITestListener {
      public void onTestFailure(ITestResult result) {
          File screenshot = ((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);
          FileUtils.copyFile(screenshot, new File("path/to/screenshot.png"));
      }
  }
  ```

What do you generally validate in your API testing?
- **Status Codes**: Ensure the API returns the correct HTTP status codes (e.g., 200, 404).
- **Response Body**: Validate the structure and content of the response body.
- **Headers**: Check if the response headers contain the expected values.
- **Performance**: Measure response time and ensure it meets performance criteria.
- **Error Handling**: Verify the API handles errors correctly and returns meaningful error messages.

Can you explain the “failed API test” status code? How would you handle a “failed API test” status code?
- **Failed API Test Status Code**: Common status codes indicating a failure might include 400 (Bad Request), 401 (Unauthorized), 403 (Forbidden), 404 (Not Found), and 500 (Internal Server Error).
- **Handling**:
  1. **Log the Failure**: Capture detailed logs of the request and response.
  2. **Analyze the Issue**: Determine if the failure is due to an issue with the test data, environment, or a bug in the API.
  3. **Report the Bug**: If it’s a bug, report it with detailed information to the development team.
  4. **Retry Logic**: Implement retry logic for transient issues.

Which tool do you use for API Automation?
- **REST Assured**: For Java-based API testing.
- **Postman/Newman**: For manual and automated API testing.

Suppose you are running a test that fails. Do you consider using Manual testing?
- **Yes**: If automated tests fail due to script issues or environmental factors, manual testing can help verify the issue and provide additional context for debugging.

What are the authentication test cases when you do manual testing?

- **Valid Login**: Test with correct credentials.
- **Invalid Login**: Test with incorrect credentials.
- **Expired Token**: Test with an expired authentication token.
- **No Token**: Test without providing an authentication token.
- **Role-Based Access**: Test different user roles and permissions.

Mobile Automation

Please give us some more background on your experience with Mobile testing automation.
- **Tools Used**: Experience with Appium for automating mobile applications on both Android and iOS platforms.
- **Frameworks**: Developed automation frameworks using Appium, integrated with TestNG for test management and Maven for dependency management.
- **Testing Types**: Performed functional, regression, and performance testing on mobile apps.
- **CI/CD**: Integrated mobile tests into Jenkins pipelines for continuous testing.

Assume you have partial test cases run in mobile and browser. Will it work in both frameworks?
- **Yes**: By designing the framework with reusable components and abstractions, test cases can be shared across mobile and web platforms. This involves creating common utility functions and page objects.

Assume validation for a native application, will it work?
- **Yes**: Validation for a native application involves checking UI elements, interactions, and backend responses, similar to web automation but using mobile-specific locators and actions.

Do you have any experience in Katalon?
- **Yes**: Katalon Studio experience includes using it for web and API automation, leveraging its built-in keywords, and customizing scripts for advanced testing scenarios.

Tell me the steps/code snippet of an application like Amazon, where you search for a particular product and add it to the cart. How do you validate in the cart?
```java
// Initialize WebDriver and navigate to Amazon
WebDriver driver = new ChromeDriver();
driver.get("https://www.amazon.com");

// Search for a product
WebElement searchBox = driver.findElement(By.id("twotabsearchtextbox"));
searchBox.sendKeys("Laptop");
searchBox.submit();

// Select a product from the results
WebElement productLink = driver.findElement(By.xpath("//span[contains(text(), 'Laptop')]"));
```

```
productLink.click();

// Add the product to the cart
WebElement addToCartButton = driver.findElement(By.id("add-to-cart-button"));
addToCartButton.click();

// Validate the product in the cart
WebElement cartButton = driver.findElement(By.id("nav-cart"));
cartButton.click();
WebElement cartItem = driver.findElement(By.xpath("//span[contains(text(), 'Laptop')]"));
assert cartItem.isDisplayed();

// Close the browser
driver.quit();
```

Jenkins and Agile Testing

In your experience with recent projects, what is your strength with the regression suite? How frequently do you run it?
- **Strength**: Comprehensive coverage of critical functionalities, robust reporting, and integration with CI/CD for automated execution.
- **Frequency**: Regression suite is typically run nightly or with every major code change.

What is the strength of the Sanity suite? How frequently do you run it?
- **Strength**: Quick checks to ensure the basic functionality of the application is intact after minor changes.
- **Frequency**: Sanity suite is run before every release and after each build.

What is the duration of the sprint?
- **Duration**: Typically 2 weeks.

How do you categorize when you do functional testing?
- **Categories**: Critical functionality, major workflows, integration points, edge cases, and user interface.

**When do we need

to run a regression?**
- **Regression**: Before major releases, after significant code changes, or after bug fixes to ensure existing functionality is unaffected.

How do you prioritize a test case in functional testing?

- **Priority**: Based on the criticality of the feature, usage frequency, impact on users, and the likelihood of defects.

Have you ever been involved in test planning and test strategy?
- **Yes**: Involved in creating test plans and strategies, defining scope, objectives, resources, timelines, and risk management.

How frequently have you been in release?
- **Release Frequency**: Typically, releases are scheduled bi-weekly or monthly, depending on the project requirements.

What are the details you will be giving in a bug report?
- **Bug Report Details**:
  - Summary
  - Steps to reproduce
  - Expected and actual results
  - Screenshots or logs
  - Severity and priority
  - Environment details

Assume a bug coming from UAT. What will you generally do?
- **Action**:
  1. Reproduce the bug in the test environment.
  2. Analyze the root cause.
  3. Log the bug with detailed information.
  4. Communicate with the development team for a fix.
  5. Verify the fix and perform regression testing.

Do you ever perform RCA?
- **Yes**: Root Cause Analysis (RCA) is performed to identify the underlying cause of defects, prevent recurrence, and improve processes.

What is traceability?
- **Traceability**: Ensuring that every requirement is covered by test cases and every test case is linked back to a requirement. This helps in ensuring complete test coverage.

JIRA

You ran a test case in 1 iteration, and then rerun it in a second test case. How do you handle the duplicates in JIRA?
- **Handling Duplicates**: Link the duplicate test cases to the original issue, add comments to provide context, and update the status to indicate the test has been rerun.

Imagine a situation where you are in a test run in release. As a lead, what will be your response to them?
- **Response**: Monitor the test run closely, address any critical issues immediately, communicate with the team and stakeholders, and ensure all testing activities are completed within the timeline.

Have you worked in Swagger? How is it different from a database? Please explain the high-level differences between them.
- **Swagger**: Used for designing, building, documenting, and consuming RESTful APIs. It provides interactive API documentation and supports API testing.
- **Database**: A structured collection of data stored electronically. Databases are used for data storage, retrieval, and management, often accessed via SQL queries.
- **Differences**:
  - **Swagger**: Focuses on API endpoints, request/response formats, and interaction with APIs.
  - **Database**: Focuses on data storage, retrieval, and relationships between data entities.

How do you validate two data in the backend in Selenium? Do you need anything special to validate the data?
- **Validation**: To validate backend data, Selenium can be integrated with database connectors (e.g., JDBC) to execute SQL queries and compare results with UI data.
  ```java
  Connection conn = DriverManager.getConnection(dbURL, username, password);
  Statement stmt = conn.createStatement();
  ResultSet rs = stmt.executeQuery("SELECT * FROM table WHERE id=1");
  while(rs.next()) {
      String dbData = rs.getString("column_name");
      String uiData = driver.findElement(By.id("element_id")).getText();
      assert dbData.equals(uiData);
  }
  ```

How do you configure Parallel Execution?
- **Configuration**: In TestNG, configure parallel execution in the XML file using the `parallel` attribute.
  ```xml
  <suite name="Suite" parallel="tests" thread-count="4">
    <test name="Test1">
      <classes>
        <class name="com.example.TestClass1"/>
      </classes>
    </test>
    <test name="Test2">
      <classes>
  ```

```
      <class name="com.example.TestClass2"/>
    </classes>
  </test>
</suite>
```

What are the parameters that you will need to configure parallel execution?
- **Parameters**:
  - **parallel**: Determines the scope of parallelism (methods, classes, or tests).
  - **thread-count**: Specifies the number of threads to use for execution.
  - **data-provider-thread-count**: For parallel execution of data provider methods.
  - **synchronized**: Ensures that the test methods run in a synchronized manner, if needed.