1. Explain Selenium Framework?

Selenium Framework is a suite of automation testing tools. It provides  a structured approach to test automation, making execution more efficient and maintainable.

* It could run the tests directly on the target browser, drive the interactions on the required web page and rerun them without any manual input.
* It eliminates repetitive manual testing that consumes lots of time and effort.
* **Selenium Remote Control (RC) (2004)**: The first Selenium tool that enabled automated web testing using a server to communicate with browsers.
* **Selenium IDE (2006)**: A browser extension for Chrome and Firefox that records and plays back test cases and is used for quick test creation and debugging.
* **Selenium Grid (2008)**: A tool that enables parallel test execution across multiple browsers and environments to reduce overall test execution time.
* **Selenium WebDriver (2011)**: It succeeded Selenium RC and allowed direct browser automation without a server to make tests faster and more reliable.

1. How to handle dynamic dropdowns in selenium?

To handle dynamic dropdowns in Selenium, you can use the Select class. You can then interact with the dropdown by using the methods provided in the Select class, such as selectByVisibleText(), selectByIndex(), or selectByValue().

Select select = new Select(dropdownElement)

selectByVisibleText(String text): Selects the option with the specified visible text.

selectByIndex(int index): Selects the option at the specified index (starting from 0).

selectByValue(String value): Selects the option with the specified value attribute.

If the dropdown's options are loaded dynamically (e.g., after a search or interaction), you might need to add waits to ensure the options are available before interacting with them. You can use WebDriverWait to wait for specific conditions before proceeding.

 5 different methods that can be used to select value in dropdown in Selenium without using Select Class. These methods are:

1. By storing all the options in List and iterating through it
2. By creating Custom Locator and without iterating the List
3. By using [JavaScriptExecutor](https://www.browserstack.com/guide/javascriptexecutor-in-selenium" \o "How to use JavascriptExecutor in Selenium" \t "_blank) class
4. By using [sendKeys](https://www.browserstack.com/guide/sendkeys-in-selenium" \o "SendKeys in Selenium WebDriver" \t "_blank)
5. By using [Actions Class](https://www.browserstack.com/guide/action-class-in-selenium)

Link :-> https://www.browserstack.com/guide/handling-dropdown-in-selenium-without-select-class

1. Can we automate CAPTCHA?

Selenium cannot directly automate CAPTCHAs

three efficient ways of **handling Captcha in Selenium**:

* By disabling the Captcha in the testing environment
* Adding a hook to click the Captcha checkbox
* By adding a delay to the Webdriver and manually solve Captcha while testing

1. Explain Exception hierarchy in java

The exception hierarchy in Java is rooted in the Throwable class, which serves as the superclass for all exceptions and errors. It branches into two main subclasses: Exception and Error.

* **Throwable:** The root class for all exceptions and errors.
* **Error:** Represents serious problems that a reasonable application should not attempt to catch. Examples include OutOfMemoryError and StackOverflowError. These typically indicate issues with the Java Virtual Machine (JVM) or system-level failures.
* **Exception:** Represents conditions that a program might try to catch and recover from. Exceptions are further divided into:
  + **Checked Exceptions:** These exceptions are subclasses of Exception (but not RuntimeException) and must be either caught using a try-catch block or declared in the throws clause of a method. Examples include IOException and SQLException. The compiler enforces the handling of checked exceptions.
  + **Unchecked Exceptions (Runtime Exceptions):** These exceptions are subclasses of RuntimeException and do not need to be declared or caught. They typically result from programming errors, such as NullPointerException or ArrayIndexOutOfBoundsException. While not mandatory to handle, it is good practice to address them to prevent unexpected program termination.

1. Difference between driver.get() and driver.navigateTo() methods

| **Factors** | **get()** | **navigate()** |
| --- | --- | --- |
| **Interface Part** | The get function is a part of the Webdriver interface. | The navigate function is a part of the Navigation interface. |
| **Wait for webpage load** | The get function waits until the webpage is loaded properly and available to return control. | The navigate function does not wait for the webpage to be loaded properly. |
| **History tracking feature** | The get function does not have the history tracking feature, thus enabling the user not to go back. | The navigate function has a history tracking feature. Thus, the user can go back to the previous web page too. |
| **Refresh feature** | The get function does not support the refresh feature. | The navigate function supports the refresh feature. |
| **Forward feature** | The get function does not support the forward function once the user has moved to the earlier webpage. | The navigate function supports the forward function and lets the user move to the next page once the user has moved to the earlier webpage. |

1. Parameterization in TestNG?

Parameterization in TestNG allows you to pass data into your test methods, enabling data-driven testing and avoiding code duplication. This can be achieved by using the @Parameters annotation and the TestNG XML file or the @DataProvider annotation.

Methods for Parameterization:

1. **1.**Using @Parameters annotation:
   * Define the parameter names within the @Parameters annotation in your test class.
   * Pass the corresponding values through the TestNG XML file or by using system properties.
   * Test methods can then access these parameters and use them in their logic.
2. **2.**Using @DataProvider annotation:
   * The @DataProvider annotation is used to provide a stream of data to a test method.
   * This is particularly useful for data-driven testing, where you want to run the same test with different inputs.
   * The data provider can return an array of objects, each representing a different set of test data.
3. How to connect Maven and TestNG?

To integrate Maven and TestNG, you first create a Maven project and then add TestNG dependencies and a configuration file to the project. This setup allows you to run TestNG tests as part of your Maven build process.

Here's a more detailed breakdown:

1. Create a Maven Project:

* Use your IDE (like IntelliJ IDEA, Eclipse, or VS Code) to create a new Maven project.
* Select the "Maven Archetype Quickstart" option (or equivalent) to get a basic project structure.
* Provide a Group ID, Artifact ID, and other project details.

2. Add Dependencies:

* Open the pom.xml file (Project Object Model).
* Add the necessary dependencies for TestNG and any other libraries your tests need (e.g., Selenium WebDriver).
* Save the pom.xml file; Maven will automatically download the required JAR files.

3. Create TestNG Tests:

* Create Java classes in the src/test/java directory to write your TestNG tests.
* Use TestNG annotations like @Test, @BeforeClass, @AfterClass, etc., to define your test methods and setup/teardown logic.

4. Create testng.xml Configuration:

* Create a testng.xml file in the src/test/java directory (or another appropriate location).
* This file specifies which TestNG test suites and classes to run, along with other configuration options.

5. Execute Tests:

* **Using Maven:** Right-click on the pom.xml file and select "Run As" -> "Maven Test" (or similar).
* **Alternatively, use the command line:** Navigate to your project directory and run mvn test.
* Maven will execute the TestNG tests based on the testng.xml configuration.

1. What are streams in java and their usage?

In Java, streams represent a sequence of elements that support sequential and parallel aggregate operations. Introduced in Java 8, streams provide a functional approach to process collections of objects. They do not store data but operate on the source data structure, producing a pipelined data flow.

Usage

Streams are used to perform various operations on collections, arrays, or I/O channels, including:

* **Filtering**: Selecting elements based on a condition.
* **Mapping**: Transforming elements.
* **Reducing**: Aggregating elements into a single result.
* **Collecting**: Gathering elements into a collection.
* **Sorting**: Arranging elements in a specific order.
* **Iterating**: Looping through elements.

1. How to work with configuration files in selenium framework?
2. Difference between Page Object Model (POM) and Page Factory? How Page factory implementing lazy initialization and how we can do same in POM?

Page Factory is a more advanced implementation of the Page Object Model (POM) that uses @FindBy annotations for element locators and automatically initializes web elements with PageFactory.initElements(). While POM uses manual initialization and does not inherently support lazy initialization, Page Factory leverages lazy initialization, where elements are only located when accessed.

Key Differences:

* **Initialization:**

POM uses manual initialization, while Page Factory uses @FindBy annotations and automatic initialization via PageFactory.initElements().

* **Lazy Initialization:**

POM initializes elements on class instantiation, while Page Factory initializes them only when accessed (lazy initialization).

* **Syntax:**

POM is more verbose, requiring more manual code. Page Factory uses annotations, making code cleaner and easier to read.

How Page Factory implements lazy initialization:

* @FindBy Annotations:

The @FindBy annotation is used to define locators for web elements. These annotations are applied to the class-level variables that represent web elements, according to a Medium article.

* PageFactory.initElements():

This method automatically initializes the web elements defined with @FindBy annotations when it's called.

* **Lazy Loading:**

When a web element is first accessed (e.g., when a method that uses the element is called), Page Factory locates the element in the DOM and stores it for future use.

Implementing Lazy Initialization in POM:

To achieve lazy initialization in a traditional POM, you can use a custom strategy:

1. **1. Create a Locator Object:**

Define a class (e.g., Locator) to store locators and a method to get the element.

1. **2. Lazy Loading:**

In the Locator class, implement a method that returns the WebElement. This method will only find the element if it hasn't already been found.

1. **3. Use in Page Objects:**

In your Page Objects, instead of directly initializing web elements, use the Locator class to get them when needed.

1. **What is the hierarchy of TestNG annotations?**
2. **How to run failed tests again in TestNG?**

**Third Round: Duration 45 min (Selenium technical questions)**

1. **Open "Crick buzz" website and get the 6's count of player by giving player name in Xpath? Dynamic Xpath**
2. **Actions class questions like how to perform keyboard actions using Actions class?**
3. **How to perform control click operation using actions class?**
4. **How to perform file upload operation in selenium and write code for it?**
5. **Write code for taking screenshot?**
6. **What are listeners and write code for it?**
7. **Manual API testing questions like status codes and http methods and difference between put and patch?**

**1 Defect Management:**

1. **What are the different statuses a defect goes through in the defect log?**
2. **If a developer rejects a defect, how should a tester proceed?**
3. **What is the defect life cycle?**

**2 Java Concepts:**

1. **Explain the difference between final and static keywords in Java.**
2. **How do you optimize your code for better performance?**
3. **What is method overriding and overloading?**

**3 XPath:**

1. **What is the difference between relative and dynamic XPath?**
2. **Open Flipkart's website, Go to footer section, write an XPath to locate multiple elements like "Contact Us," "About Us," "Careers," etc.**

**4 Testing Methodologies:**

1. **What is regression testing?**
2. **What is sanity testing?**
3. **What is smoke testing?**
4. **Which testing is performed first: smoke or sanity?**

**5 Automation Framework:**

1. **Explain the framework you used in your project. How do you read a CSV file in the Serenity framework?**
2. **What is the difference between, findElement and findElements?**
3. **What is implicit wait, explicit wait, fluent wait ?**
4. **What is polling in fluent wait?**
5. **Explain how much you know about exceptions in selenium?**

**6 Bug Handling:**

1. **If you find a bug in development, what steps will you take?**

**7 Coding Challenges:**

1. **Write a Java program to find the occurrence of each letter in a given string.**
2. **Write a Java program to return the most frequent character in a string.**
3. **Write a Selenium script to find all broken links on a webpage.**

**8 Collections in Java:**

1. **What are Set and Map in Java? What are the key differences between them?**

**These 10 ChatGPT prompts help you to crack tough interview questions:**

1. **Why are you interested in this position?**
2. **Prompt: "Provide an example of how to express interest in a software engineering position at a tech startup."**
3. **Tell me about yourself.**
4. **Prompt: "How do I construct a concise and impactful 'tell me about yourself' answer for an interview?"**
5. **What is your greatest strength?**
6. **Prompt: "What's a good way to frame problem-solving as a strength in an interview?"**
7. **Can you describe your experience with \_?**
8. **Prompt: "How can I effectively communicate my experience with project management in an interview?"**
9. **Why are you looking to leave your current role?**
10. **Prompt: "Give an example of a professional and positive reason for leaving a current job."**
11. **What is your greatest weakness?**
12. **Prompt: "How can I discuss a weakness in a constructive way during an interview?"**
13. **Where do you see yourself in five years?**
14. **Prompt: "How can I answer the 'where do you see vourself in five years' question in a way that showe**
15. **How do you manage test data across different environments?**
16. **→ Use of factories, mock data, seed scripts, or dynamic data generation.**
17. **What's your test pyramid and how do you align with it?**
18. **→ Talk about ratios between unit, API, UI tests, and shift-left testing.**
19. **How do you validate logs and system metrics as part of your test validations?**
20. **→ Use of tools like ELK, CloudWatch, or custom log parsers in tests.**
21. **Tell me how you tested a CI/CD pipeline for production deployments.**
22. **→ Validations, smoke tests, canary testing, rollback strategies.**
23. **How do you parallelize your tests for faster feedback?**
24. **→ Test isolation, containerization, thread-safe designs, test splitting.**
25. **How do you handle test automation for dynamic elements and frequent Ul changes?**
26. **→ Use of stable locators, resilient selectors, visual testing.**
27. **How do you ensure traceability between requirements, test cases, and defects?**
28. **→ Linkage with tools like Jira, Zephyr, or custom traceability matrices.**
29. **How did your automation suite catch a regression that manual testing missed?**
30. **→ Share a story, the defect, and the impact it had.**
31. **Explain your test strategy for a microservices-based application.**
32. **→ Talk about API + contract testing, mocks/stubs, service virtualization.**
33. **How do you handle flaky tests in your Cl pipeline?**
34. **Techniques like retries with logic, root cause analysis, and stabilization.**
35. **How do you test a feature that involves 3 services owned by different teams?**
36. **→ Discuss integration points, mocks, test data setup, and ownership boundaries.**
37. **What's your framework design pattern and why?**
38. **→ Talk about POM, data-driven testing, custom utilities, and code reuse.**
39. **How do you test for accessibility (a11y) in your Ul automation?**
40. **→ Lighthouse, axe-core, and how you integrated it into your pipeline.**
41. **What's your approach to testing asynchronous operations or eventual consistency?**
42. **→ Polling, waits, custom assertions, timeouts with context.**
43. **Selenium WebDriver:**
44. **What are different types of locators in Selenium?**
45. **When do you use Xpath over CSS locators?**
46. **How to get a specific value from a dropdown and reuse it in verifications?**
47. **When do we use JavaScript Executors?**
48. **Is it possible to validate Captcha using Selenium? If yes how?**
49. **What should be the ideal way to store data using Selenium WebDriver only?**
50. **Is it possible to use Xpath like parent/child/node/.. ? If yes, provide an example where to use it?**
51. **What happens if you receive browser notifications during test automation execution?**
52. **Why does Stale Element Exception occur, and how to handle it?**
53. **What is Invalid Certificate Exception?**

**APIs:**

1. **What are the components of an HTTP request?**
2. **What is the difference between API and unit testing?**
3. **What is an HTTP response?**
4. **How can we add validation points in Postman?**
5. **what do you understand by server-side validation?**
6. **What is 3-tier architecture?**
7. **What is the difference between web services and APIs?**
8. **What is REST, SOAP, and GraphQL in APIs?**
9. **What do you test in standalone APIs?**
10. **What do you test in 3rd-party integrated APIs?**

**Postman:**

1. **When to use collections, environments, and global variables?**
2. **How to execute a collection end-to-end?**
3. **How to validate that an API response has the correct status code?**
4. **What happens when an API response returns Form Data instead of JSON, and how to validate it?**
5. **How to set up Basic Auth in Postman?**
6. **Where do you store environment credentials?**
7. **How to save a demo response for an API request?**
8. **How will you validate an API request if VPN is required for it to work?**
9. **How do you filter results in an API request using Postman?**
10. **How to set up custom headers in Postman?**

**Git:**

1. **What are the different stages in committing the code to GitHub?**
2. **Is it possible to revert changes in a remote repository? If yes, how?**
3. **When do you commit your code? After committing, how do you validate that everyone has the updated code?**
4. **How to merge stashed changes in a local repository?**
5. **Why do we need a`.gitignore` file? How do you add files to it?**

**TestNG:**

1. **What is the execution format of tests in TestNG?**
2. **Can priority be negative for methods? If yes, what is the execution flow as per priority?**
3. **What is the difference between `dependsOnMethods` and `dependsOnGroups'?**
4. **What are the different ways to exclude tests in TestNG?**
5. **What does 'threadPoolSize' mean in TestNG, and how does it work?**

**Selenium WebDriver Basic Questions**

1. **What is Selenium?**
2. **Difference between Selenium 3 and Selenium 4?**
3. **How do you handle dropdowns in Selenium?**
4. **How to handle multiple windows in Selenium?**
5. **How do you handle alerts and pop-ups?**
6. **How do you handle frames and iframes?**
7. **What is the difference between 'driver.quit()` and `driver.close()`?**

**Selenium WebDriver Advanced Questions**

1. **Implicit vs Explicit Wait - What's the difference?**
2. **What is Fluent Wait, and when do you use it?**
3. **How do you handle 'StaleElement ReferenceException'?**
4. **What is JavaScriptExecutor? How do you use it?**
5. **How do you scroll a webpage using Selenium?**
6. **How do you take a screenshot in Selenium?**
7. **How do you validate broken links in Selenium?**
8. **How do you capture network logs in Selenium?**

**Selenium Framework Questions for Best Practices**

1. **What is the Page Object Model (POM)?**
2. **What is Page Factory? How is it different from POM?**
3. **What are Selenium Grid and its advantages?**
4. **How do you handle dynamic elements in Selenium?**
5. **What is the role of 'Desired capabilities in Selenium?**
6. **What are the different types of Assertions used in Selenium?**
7. **What are the limitations of Selenium WebDriver?**
8. **How do you integrate Selenium with TestNG**
9. **What are TestNG Listeners, and how do you implement them?**

**Selenium Execution & Debugging questions**

1. **How do you run Selenium tests in headless mode?**
2. **How do you handle authentication pop-ups in Selenium?**
3. **How to execute parallel tests in Selenium?**
4. **What are the different ways to maximize a browser window in Selenium? TestNG?**
5. **What is BDD?**
6. **What is Cucumber?**
7. **What is feature file?**
8. **What are step definitions?**
9. **What is scenario Outline?**
10. **What is the background keyword?**
11. **How do you use tags in Cucumber?**
12. **How do you handle data tables?**
13. **What are Cucumber hooks?**