**Cucumber interview questions syntax 20-09-2024**

**What is Cucumber?**

The Cucumber is a testing framework which is used for testing other software application.it supports BBD which means behavior of application is written in Gherkin language. Gherkin is simple plain English language, so that non-technical stakeholder can understand the application behavior.

**What is BDD?**

Behaviour of the test cases is written in gherkin language. Instead of showing the actual code showing the functionality of application to the stakeholder. Behaviour is going to drive the framework.

**Why did you decide to start using Cucumber?**

The best reason for choosing the cucumber is that we can explain the requirements

in simple english .

**What is the difference between Cucumber and TestNG?**

**Target Audience**

* **Cucumber**:
  + Designed for **business stakeholders, testers, and developers** who collaborate on writing test scenarios in plain English.
* **TestNG**:
  + Primarily used by **developers and testers** for running and organizing test cases at the code level.

In TestNG, you define test classes, which are Java classes that contain your test methods. You annotate these test methods using annotations like @Test, @BeforeClass, @AfterMethod, etc

**Cucumber** is a testing tool that supports **Behavior-Driven Development (BDD)**. It allows teams to write acceptance tests in a gherkin language format that both technical and non-technical stakeholders can understand.

Acceptance testing is a type of software testing where the system is tested to ensure that it meets the business requirements and is ready for delivery to the customer or end users. It is usually performed at the final stage of the software development **lifecycle before the software is deployed.**

**Summary:**

| **Feature** | **Cucumber** | **TestNG** |
| --- | --- | --- |
| **Main Focus** | Behavior-Driven Development (BDD) | Unit/Integration/Functional Testing |
| **Test Writing** | Gherkin (Given, When, Then) | Java with annotations (@Test, etc.) |
| **Target Audience** | Business, Testers, Developers | Developers and Testers |
| **Integration** | Works with Selenium, JUnit, TestNG | Works with Selenium, Maven, Jenkins |
| **Reporting** | Human-readable reports | Detailed HTML test reports |
| **Test Types** | Acceptance and End-to-End testing | Unit, Integration, Regression, Functional testing |

**Give an example of a behavior driven test in plain text?**

**Feature:** Defines what feature you will be testing in the tests below

**Scenario** : we are describing the test case here

**Given:** Tells the precondition of the test

**When** Defines actions to be taken

**And:** Defines additional/ continues conditions of the test

**Then:** States the post condition. You can say that it is the expected result of the test.

**What are the two files required to run a cucumber test?**

**1 A feature file.**

**2 A step definition file.**

**5. What is a feature file?**

A **feature file** is an essential component of **Cucumber**, where test scenarios are written in a structured format using the **Gherkin** language.

**What is the extension?**

**File Extension:**

* The extension of a feature file is **.feature**.

**What is there in a feature file?**

**Contents of a Feature File:**

A feature file typically includes:

* **Feature**: Describes the functionality or feature being tested.
* **Scenario**: Defines individual test cases or behavior to be tested.
* **Steps**: These are the individual actions within a scenario, written using **Gherkin keywords** like Given, When, Then, And, But.

**What are the keywords used in feature files?**

Feature: Title of the feature (description of the feature being tested)

Scenario: Title of the scenario (a specific test case)

Given [initial context or state]

When [action/event]

Then [expected outcome]

**What is the language used to write scenarios in a feature file?**

The language used to write scenarios in a **feature file** is called **Gherkin**.

**What is the main purpose?**

**Main Purpose of Gherkin**:

* The main purpose is to provide a **common language** that allows collaboration between developers, testers, and business stakeholders.
* It helps to define **acceptance criteria** in the form of scenarios that describe how the software should behave.
* Gherkin scenarios serve as **documentation** for the systems expected behavior and are also **automated tests**.

**Is it mandatory to use Given, When, Then keywords while writing scenario? What**

**is the difference?**

No it is **not mandatory to use Gherkin keywords** . Gherkin keywords makes our

scenario be in a more readable format. \* can also be used to write steps

in the feature file. Using Gherkin keywords provide better readability as each keyword

has a specific meaning: **Given** is for precondition, **When** is for action, **Then** is for result

**Feature:** Account Balance

**Scenario:** Verify Positive Balance

**\*** I have $100 in my account

\* I withdraw $50

\* I should have $50 balance

**Name any two build management tools that can be integrated with Cucumber?**

. gradle

. maven

**Explain Cucumber Tags?**

1. **tags**------------------is cucumber option which is used to run multiple scenario of your own choices.

**How to run only specific scenarios? How to run multiple**

**scenarios with tags? How to exclude tagged scenarios?**

Tags are used to:

● Group scenarios

● Ignore scenarios from execution

● Logically group (OR & AND)

For example we are having feature file with three test cases here i want to execute only

smoke test case:

**Feature:** ECommerce Application Login

**@SmokeTest**

**Scenario:** Successful Login

**Given** This is a blank test

**@RegressionTest**

**Scenario:** UnSuccessful Login

**Given** This is a blank test

To execute **specific tags** we have to mention them in CucumberOption in our runner

class.

**@Cucumber.Options(tags="@smoke ")**

To execute **multiple tags** :

Execute all tests tagged as @smoke **OR** @regression

**@CucumberOptions(tags = "@smoke or @regression")**

Execute all tests tagged as @smoke **AND** @regression

**@CucumberOptions(tags = "@smoke and @regression")**

**to skip specific tags**

Execute all tests of the feature tagged as @smoke

**@CucumberOptions(tags = "not @smoke")**

This is AND condition, which means all the scenario tagged as @FunctionalTest but not

@SmokeTest

**11. What is the use of the keyword "background" in a feature file?**

Background keyword is used to write all the steps which are common in all scenarios present in a feature file. . You can say that this is precondition before run of every scenario.

These steps are executed before each scenario in the feature file.

Common steps mean common between 2 or more than 2 scenarios not in a single scenario.

**12. Explain Cucumber Hooks? What’s Before/After hook?**

**What are hooks???**

The steps which are common in all the feature files like opening browser and close browser written in a class called hooks. There are two hooks in cucumber @before and @after. We provide hooks for precondition and post condition to write common steps of all feature files.

We provide these hooks under the step package where our java code is present.

-------------------------------------------------------------------------------------------------------------------------

**There are two hooks in cucumber**.

**@Before** hooks will be run before the first step of each scenario.

**@After** hooks will be run after the last step of each scenario, even when there are failing, undefined, pending or skipped steps.

**What is @CucumberOptions in test runner? List the properties of**

**@CucumberOptions**

@CucumberOptions ()

@CucumberOptions is an annotation in Cucumber that is used to provide various configurations for running your **Cucumber tests**. It is typically applied to the **TestRunner** class, where it specifies details like where the feature files are located, where the step definitions can be found, the type of reports to generate, and other options to customize the test execution.

**There are 6 cucumber options generally used**.

1. **feature** ------is cucumber option where you specify the path of feature files present in your project.
2. **glue**---------- is a cucumber option which is used to make link between feature file and step definition
3. **dry-run** ---------is a cucumber option which is used to generate unimplemented steps, and saves your time by just scanning the code.
4. **monochrome** ---------is a cucumber option which is used to remove unnecessary code from console
5. **plugins-----------------------** is cucumber option which is used to generate html and json report for your test cases.

**14. What is the difference between scenario and scenario outline?**

**What is difference in scenario and scenario outline???**

**Scenario is a keyword which mean how many possibilities are there to test the user story or functionality.**

**Scenario Outline:**

* A **Scenario Outline** is a keyword used with Examples table to pass multiple sets of data to test single scenario.
* A **Scenario Outline** is typically paired with an Examples table that provides the data.

**Example:**

Example table is applicable for whole scenario.

There is no need of loop to iterate across multiple sets of data , examples table automatically iterate the scenario until the condition is met.

Every time open the browser and close the browser.

**How can we achieve data driven testing in Cucumber? What is DataTable in**

**Cucumber? DataTable vs Scenario Outline?**

To achieve data driven nesting in Cucumber we can use:

. Scenario Outline and Examples keyword

. DataTable

Cucumber DataTables lets us store data in a feature file.

There are 2 types of DataTable: with the header and without the header.

**Cucumber DataTable without header**

**#Author: asel@syntaxtechs.com**

**Feature: Dashboard**

Scenario: Dashboard menu view for admin

When user is logged with valid admin credentials

Then user see dashboard menu is displayed

| Admin | PIM | Leave | Time | Recruitment | Performance | Dashboard | Directory |

**2. Cucumber DataTable with header**

**@regression**

**Scenario: Adding multiple employees**

**When user enters employee details and click on save then employee is added**

**| FirstName | MiddleName | LastName |**

**| John | J | Doe |**

**| Jane | J | Smith |**

**What is Difference between data table and examples table??**

* Data table is declared with scenario keyword and examples table is declared with scenario outline keyword
* Data table not need examples keyword
* Data table is applicable only for one step
* Examples table is applicable for entire scenario
* Data table open and close browser just once.
* Examples table open and close browser for each iteration
* Example tables waste lot of time to unnecessarily open and close browser after each iteration.

**16. When in Cucumber some test scenarios fail and now you want to run failed ones,**

**how would you do it?**

**1. Modify existing runner class**

Add **rerun** :target/rerun.txt to your plugin

**@RunWith(Cucumber.class)**

**@CucumberOptions(**

**features = "src/test/resources/features"**

**, glue = "com/hrms/steps"**

**, plugin = { "pretty",**

**"html:target/cucumber-default-reports",**

**"json:target/cucumber.json",**

**"rerun:target/rerun.txt"}**

**, tags= {"@Temp"}**

**, monochrome = true**

**, dryRun = false)**

**public class TestRunner {**

**}**

This will run all tests and then list all failed scenarios in rerun.txt file

The file rerun.txt file is located in the target folder.

Rerun.txt contains path to the feature file and line number for scenario that failed

**Create another runner class**

**@RunWith(Cucumber.class)**

**@CucumberOptions(**

**features="@target/rerun.txt"**

**, glue = "com/hrms/steps"**

**, monochrome = true**

**)**

**public class FailedRunner {**

**}**

**17. Let's say I have a Google app and in a Cucumber file I have 20 scenarios and there**

**are 2 steps that are all common for all scenarios. How can I achieve that without**

**repeating the code?**

**1. Use the Background Keyword:**

**The Background keyword allows you to define steps that are common across all scenarios in a feature. These steps will run before each scenario. This is useful if the same setup or preconditions are needed for all scenarios.**

**18. In cucumber I have scenario with 10 steps and let’s say step #5 I want to execute 5 times. How can I do it?**

To execute **step #5** multiple times (e.g., 5 times) within a single scenario in **Cucumber**, you can achieve this by implementing a loop in the **step definition**. The **Gherkin** language itself does not support loops or repetitions directly, so you’ll need to handle this in your step definition using the programming language you're working with (e.g., Java).

-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**These are extra questions taken from chat gpt**

Interview Questions:

1. How does your framework handle parallel execution?
2. Explain the purpose of the Page Object Model in your framework.
3. How do you manage different environments (e.g., dev, staging, prod) in your framework?
4. How does your framework handle test data management?
5. Explain how you've implemented reporting in your framework.
6. How does your framework handle waits and synchronization issues?
7. How do you handle cross-browser testing in your framework?
8. Explain the role of the Hooks class in your framework.
9. How do you handle dependencies in your project?
10. How does your framework support CI/CD integration?
11. **How does your framework handle parallel execution?**
    1. My framework handles parallel execution using the @DataProvider annotation in TestNG or by configuring the Maven surefire plugin for JUnit. I set up the threadCount or use Cucumber options like parallel=true to run tests simultaneously, improving test efficiency.
12. **Explain the purpose of the Page Object Model in your framework.**
    1. POM helps in maintaining cleaner code by separating test logic from the page structure. Each web page is represented by a class, and all elements and actions on that page are encapsulated, enhancing reusability and maintainability.
13. **How do you manage different environments (e.g., dev, staging, prod) in your framework?**
    1. I manage different environments by using configuration files (e.g., config.properties or YAML) for storing environment-specific URLs and credentials. Depending on the environment (dev, staging, prod), the appropriate values are loaded dynamically during test execution.
14. **How does your framework handle test data management?**
    1. I handle test data through external files such as Excel, CSV, or JSON. Using tools like Apache POI for Excel or Jackson for JSON, I extract data as needed. Environment-specific data can also be managed through properties files.
15. **Explain how you've implemented reporting in your framework.**
    1. For reporting, I use tools like Extent Reports or Allure, integrated within the framework. These tools generate detailed, visual reports that include test execution status, logs, and screenshots for failed steps.
16. **How does your framework handle waits and synchronization issues?**
    1. The framework manages waits using Selenium’s WebDriverWait for explicit waits or FluentWait for customized polling intervals. Implicit waits are also set globally to avoid synchronization issues between the browser and the script.
17. **How do you handle cross-browser testing in your framework?**
    1. I handle cross-browser testing by using Selenium WebDriver and the @Parameter annotation in TestNG or Maven profiles to specify different browsers like Chrome, Firefox, and Edge. I also use tools like BrowserStack or Selenium Grid for cloud-based testing.
18. **Explain the role of the Hooks class in your framework.**
    1. In Cucumber, the Hooks class contains setup and teardown steps using @Before and @After annotations. It ensures preconditions (e.g., browser setup) are met before each test, and cleanup activities (e.g., closing the browser) are done afterward.
19. **How do you handle dependencies in your project?**
    1. I manage dependencies through Maven or Gradle by defining required libraries in the pom.xml or build.gradle file. This ensures all necessary dependencies like Selenium, TestNG, Cucumber, etc., are automatically handled.
20. **How does your framework support CI/CD integration?**
21. My framework supports CI/CD using Jenkins, where I configure jobs to run tests automatically on code commits. Test execution results are shared via reports, and build failures trigger alerts, ensuring early detection of issues.

----------------------------------------------------------------------------------------------------------------------------------

**How does your framework implement the Page Object Model?**

* My framework implements the Page Object Model (POM) by creating separate classes for each page of the application. These classes contain WebElements and methods to interact with the elements. This separation improves maintainability and reduces duplication of code.

**Explain the role of the PageInitializer class in your framework.**

* The PageInitializer class is responsible for initializing all the page objects used in the tests. It uses the PageFactory.initElements method to instantiate page objects, making it easier to manage page interactions in test execution.

**How do you handle different browsers in your framework?**

* Different browsers are handled through WebDriver initialization in my framework. I use browser-specific drivers for Chrome, Firefox, and Edge, and I parameterize the browser choice via configuration files or by passing it as a Maven/Gradle parameter.

**What is the purpose of the commonMethods class?**

* The commonMethods class contains reusable utility methods that can be used across different test classes, such as methods for handling alerts, taking screenshots, or managing dropdowns. It promotes code reuse and reduces redundancy.

**How are you managing test data in your framework?**

* Test data is managed using external sources like Excel files, CSV, or JSON. I read data using libraries such as Apache POI (for Excel) or Jackson (for JSON), making it easier to maintain and update test data without modifying the code.

**Explain how you've implemented reporting in your framework.**

* Reporting is implemented using Extent Reports or Allure. These tools integrate with the framework to generate detailed HTML reports, including test pass/fail status, logs, and screenshots for failed steps.

**How does your framework handle waits and synchronization?**

* My framework uses explicit waits (WebDriverWait) to handle synchronization issues, ensuring that elements are available before performing actions. I also use implicit waits globally, and for specific needs, FluentWait helps with customized polling intervals.

**What is the role of the Hooks class in your Cucumber framework?**

* The Hooks class in my Cucumber framework contains @Before and @After methods that execute before and after each test scenario. It handles setup tasks like initializing the browser and teardown tasks like closing the browser or capturing logs.

**How do you parameterize tests in your framework?**

* Tests are parameterized using TestNG’s @DataProvider or Cucumber’s Examples keyword in scenarios. This allows me to run the same test with different sets of data, enhancing test coverage and efficiency.

**Explain how you've implemented screenshot capture for failed tests.**

* Screenshot capture is implemented in the @After method of the Hooks class, where I use Selenium’s TakesScreenshot interface. If a test fails, the screenshot is captured and embedded into the test report for better debugging.

**How does your framework handle different environments (e.g., dev, staging, prod)?**

* Different environments are handled using configuration files (e.g., config.properties). Environment-specific values like URLs or credentials are stored and loaded dynamically based on the environment specified during test execution.

**What strategy do you use for locating elements in your framework?**

* I primarily use CSS Selectors and XPath for locating elements. CSS is faster, but XPath is more versatile for complex element identification. I choose the strategy depending on the web page structure and the requirements of the test.

**How have you implemented parallel test execution in your framework?**

* Parallel execution is implemented using TestNG’s parallel attribute in the XML file or Maven’s surefire plugin for JUnit. Cucumber tests are also run in parallel by configuring the plugin options with parallel=true.

**Explain the role of the constants class in your framework.**

* The constants class stores all the static constants used across the framework, such as URLs, file paths, or browser types. This centralizes the configuration and avoids hardcoding values, making the framework more flexible and easier to maintain.

**How do you manage dependencies in your project?**

* Dependencies are managed through Maven or Gradle. By listing all necessary libraries in the pom.xml or build.gradle file, Maven/Gradle automatically downloads and manages versions of libraries like Selenium, Cucumber, and TestNG, ensuring smooth project setup.

--------------------------------------------------------------------------

What is sure fire plugin???

**Parallel Test Execution**: It allows running tests in parallel to speed up the test execution time.

**Reports**: After running the tests, Surefire generates reports summarizing the results (pass, fail, skipped tests) in formats like text, XML, and HTML.