**What is Cucumber?**

Cucumber is an open-source testing tool that supports the [Behavior Driven Development (BDD)](https://www.lambdatest.com/blog/behaviour-driven-development-by-selenium-testing-with-gherkin/" \t "_blank) method. It can be integrated with a variety of testing frameworks and tools, including [Selenium](https://www.lambdatest.com/selenium) and [Appium](https://www.lambdatest.com/appium" \t "_blank), and it supports numerous programming languages. Since Cucumber offers a standard language for describing requirements and test scenarios, it facilitates collaboration between developers, testers, and business analysts in Agile software development approaches.

**What are the advantages of using Cucumber?**

Cucumber offers several advantages for software testing:

* Cucumber tests are written in plain-text English, so anyone with a minimum level of technical knowledge can create scenarios.
* It enables us to include business stakeholders who need help to read code easily.
* Allows quick and simple setup and operation
* A high degree of code reuse in the tests
* It is compatible with Selenium and various testing frameworks, including JUnit and TestNG.

**What are the different types of testing that can be performed using Cucumber?**

Cucumber is a testing tool that supports various types of testing, such as acceptance testing, functional testing, and regression testing.

* [Acceptance testing](https://www.lambdatest.com/learning-hub/acceptance-testing) includes testing software's performance and compliance with requirements. Cucumber simplifies verifying that the software complies with the required specifications by allowing teams to define acceptance tests in a human-readable language.
* [Functional testing](https://www.lambdatest.com/learning-hub/functional-testing) includes evaluating certain software features or functions to ensure they function as intended. Teams can build and carry out available tests using Cucumber in a language that technical and non-technical stakeholders can easily comprehend.
* [Regression testing](https://www.lambdatest.com/blog/regression-testing-what-is-and-how-to-do-it/) requires retesting the software following changes to ensure that the previously functional features are still operational.

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

A feature file in Cucumber is a plain text file that acts as the starting point for defining the software's behavior in a human-readable language. The feature file often includes several scenarios explaining how the software should behave in different situations. Each scenario in the feature file is divided into a set of steps that define the tasks to be completed and the outcomes that should be expected.

**What is a scenario in Cucumber?**

A scenario is a fundamental Gherkin structure. Each scenario begins with the keyword "Scenario:" (or a localized variation of it) at the beginning and ends with a title.

**How do you share state between steps in Cucumber?**

Cucumber uses instance variables to allow state to be shared between phases. Step definitions allow for the definition of and access to instance variables that can be used in numerous steps.

**How can you pass data between Cucumber steps?**

Scenario context, a shared data structure that can be accessed by all Cucumber steps in a scenario, allows data to be exchanged across Cucumber steps. Alternatively, data can be transferred between steps using tables or samples of scenario outlines.

**What are the different types of steps in Cucumber?**

Cucumber supports three types of steps: Given, When, and Then.

**What is the purpose of the Given step in Cucumber?**

The Given step in Cucumber is used to specify the initial state or the prerequisites necessary for the test scenario to execute. In addition to helping to build up the necessary test data and create the required test objects, it describes the environment in which the test is executed.

**What is the purpose of the When step in Cucumber?**

The When step in Cucumber is used to specify the action or event that must occur as part of the test scenario. It outlines the procedures needed to simulate a user's interaction with the system being tested.

**What is the purpose of the Then step in Cucumber?**

The Then step in Cucumber defines the desired result or test scenario result. It outlines the necessary validation or verification processes to ascertain whether the system under test is acting by expectations.

**What is the difference between a Background and a Scenario Outline in Cucumber?**

A Background is a set of steps that every scenario in a feature file shares. The same scenario is run several times with varying input values using a scenario outline. A Scenario Outline can be used several times to run the same scenario with different input data sets. However the Background section can only be used once per feature file.

**What is a tag in Cucumber?**

To categorize and arrange the scenarios and features in a feature file, Cucumber uses tags, which are labels. A tag is a keyword that is preceded by the "@" symbol and the tag name. For instance, the tag "@smokeTest" might be used to identify features or scenarios that are part of a smoke test suite.

**What is the purpose of a Cucumber tag?**

A Cucumber tag is used to assign metadata to a scenario or feature so that it may be filtered and put through particular tests. In addition to grouping related scenarios, and marking them as skipped or pending, tags can also be used to prevent some scenarios from being executed.

**What is the purpose of using regular expressions in Cucumber?**

In Cucumber, regular expressions are used to match step definitions to the relevant stages in feature files. To create placeholders that can be changed at runtime to certain values, regular expressions are required.

**What is a hook in Cucumber?**

A hook in Cucumber is a code block that can be added before or after a scenario or feature. With the support of hooks, you can carry out setup and teardown actions before and following each scenario or feature. According to their placement, hooks are defined in a separate file and are automatically run either before or after each scenario or feature.

**What is a data table in Cucumber?**

A data table in Cucumber is a way to deliver a set of data to a Step in a Scenario in a tabular manner. In Cucumber, a data table is defined by a series of pipes (|) to split the columns and hyphens (-) that divide the header row from the data rows. For instance, the following data table has two rows of data and the following two columns: "Username" and "Password":

| Username | Password |

| user1 | pass1 |

| user2 | pass2 |

**What is a scenario outline in Cucumber?**

A Cucumber feature called a Scenario Outline allows the creation of many test cases based on a single scenario. It is used when a scenario needs to be run with various sets of data. The term "Scenario Outline" is used to define the Scenario Outline. Angle brackets (">") are used to indicate placeholders, which are replaced with actual values during test execution. Below is a small example:

Scenario Outline: Login Functionality

Given I am on the login page

When I enter "<username>" and "<password>"

And I click the login button

Then I should be logged **in**

Examples:

| username | password |

| user1 | pass1 |

| user2 | pass2 |

In the above example, the login and password are placeholders in the Scenario Outline that are changed to actual values from the Examples table during test execution. Several sets of test data for the same scenario are defined in the Examples table.

**What is the purpose of the Background keyword in Cucumber?**

Cucumber's Background keyword is used to define a list of common actions that occur before each scenario in a feature file. Moving recurring activities into the background section helps to reduce code duplication and streamline the feature file.

**How do you skip a scenario in Cucumber?**

Cucumber allows you to skip scenarios by adding the tag “@ignore” or “@skip” to the scenario. Here's an illustration of how Cucumber skips a scenario:

@ignore

Scenario: This scenario will be skipped

Given I **do** not want to execute this scenario

When I tag it **with** the @ignore tag

Then Cucumber will skip it

**How do you generate HTML reports in Cucumber?**

To create HTML reports, Cucumber comes with a built-in plugin called HTML Formatter. To create reports with more specific information, you can also use third-party plugins like ExtentReports or Allure Reports.

**What is a cucumber report?**

A report in Cucumber is a document that shows the outcomes of the tests that have been run. After a test run is over, cucumber reports are generated. They offer essential details about the test execution, including the state of each test scenario and the overall state of the test suite.

**What is a step definition in Cucumber?**

A step definition in Cucumber is a section of code that specifies the behavior for a specific step in a scenario. Each step in a scenario is matched to a step description by Cucumber before it is executed. Depending on the programming language used to construct the test automation framework, step definitions may be written in Java, Ruby, or JavaScript.

**What is a scenario outline in Cucumber?**

A Scenario Outline in Cucumber is a means to execute the same scenario using several input data sets systematically. It enables testers to create a single scenario and include placeholder input values that can later be changed at runtime by other data sets. The input data are defined using an ‘’Examples table’’, and the scenario outline is defined using the phrase ‘’Scenario Outline’’.

**What is a cucumber feature?**

A cucumber feature is a high-level description of how an application or system behaves that is stated in plain English and adheres to a particular syntax that Cucumber has established. The goal of employing Cucumber features is to give developers, testers, and business analysts a shared knowledge of the specifications and functionality of a system. Cucumber enables various stakeholders to work together efficiently and ensures that the system is created and tested following the expected behavior by using a common language and syntax.

**What is a cucumber plugin?**

Cucumber offers a plugin mechanism to expand its capabilities beyond the basic features. In essence, a plugin is a piece of code that gives Cucumber additional functionality. Plugins can be used for various purposes, including report generation, output format customization, and integration with other testing tools.

By simply adding the plugin name to the Cucumber command-line arguments or by including them in the configuration file, plugins can be introduced to the project. Formatters, reporters, and custom plugins are just a few of Cucumber's many plugin kinds.

**What is a cucumber scenario outline example?**

A cucumber scenario outline example is a template for creating various scenarios from a collection of input values. The scenario outline is a framework for building a collection of connected scenarios with similar actions but distinct input values. The different input values that will be used to produce the methods are represented by placeholders referred to as "sample tables." Consider the following example of a login function that accepts various usernames and passwords:

Scenario Outline: User logs **in** **with** valid credentials

Given the user is on the login page

When they enter "<username>" and "<password>"

And they click the login button

Then they should be redirected to the dashboard

Examples:

| username | password |

| user1 | pass1 |

| user2 | pass2 |

| user3 | pass3 |

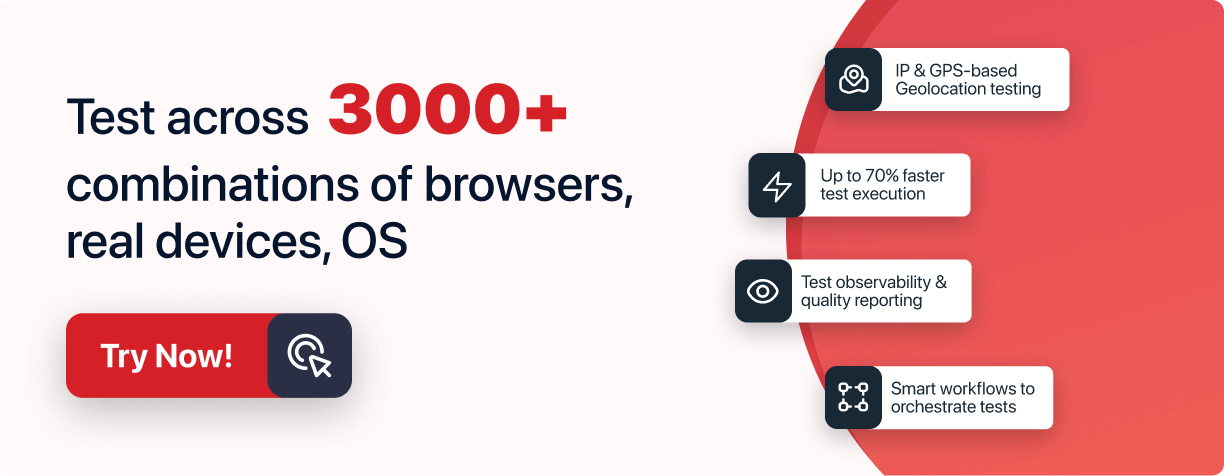
In this illustration, the scenario outline offers a framework for building a variety of login scenarios using various identities and passwords. The input values for each situation are listed in the example table. Each row in the sample table receives its scenario when Cucumber executes this scenario description.

**What is a cucumber step?**

A cucumber step is a specific action or behavior specified in the feature file and put into practice in the step definition file. It stands for a particular action that must be taken throughout the test scenario and often contains a description of the action and any relevant parameters. The implementation of the step in the step definition file would come after a step such as "Given the user navigates to the login page." Since cucumber steps are intended to be reused across several contexts, testing may be done more quickly and effectively.

**What is a cucumber test?**

A powerful method for validating the accuracy and quality of software programs is the cucumber test. A cucumber test is a test case that uses the Behavior Driven Development (BDD) methodology and the Cucumber testing framework to build and run automated tests. Cucumber's test cases are created using the Gherkin syntax and natural language, making it simple for stakeholders, testers, and developers to collaborate. The test cases are arranged into feature files, which provide scenarios and instructions that specify how the system under test should behave.

[](https://accounts.lambdatest.com/register)

**What is a cucumber world in Cucumber?**

A World is a global object in Cucumber that can contain a state which is shared among several stages and scenarios in a feature file. It enables data sharing and communication between phases, which is extremely helpful in complex systems. Every step definition function in a scenario receives the World object as an argument after it is generated and before the first step is executed. This makes it possible to transfer information between steps by enabling the step definitions to access and alter the state kept in the World object. The World object is specific to a feature file and does not persist across different feature files, which is a crucial distinction to make.

**What is Gherkin in Cucumber?**

Cucumber scenarios are written in Gherkin, a language that is readable by people. It is a simple, domain-specific language that enables technical team members and stakeholders to work together and define the application's behavior in a concise and easy to understand manner. The application's desired behavior is described using Gherkin in terms of features, situations, and actions that are simple to comprehend and can be carried out using Cucumber. It uses plain English text syntax with phrases like Given, When, Then, And, and But to express how the application behaves. Gherkin files have a.feature extension and may be created in any text editor.

**What is the difference between a feature and a scenario in Cucumber?**

|  |  |
| --- | --- |
| **Feature in Cucumber** | **Scenario in Cucumber** |
| A feature in Cucumber is a high-level description of a software need or feature. It often has one or more situations and is defined in a feature file. | A scenario is a particular test case or illustration that confirms a specific behavior or result of the feature. |
| The "Feature" keyword is used at the start of a feature in the Gherkin language. | The "Scenario" keyword is used at the start of scenarios written in the Gherkin language. |
| Specific test cases can be described by one or more scenarios for a feature. | One or more steps in a scenario can specify the behaviour that will be tested. |

**What is the difference between a scenario and a step in Cucumber?**

|  |  |
| --- | --- |
| **Scenario in Cucumber** | **Step in Cucumber** |
| A scenario is a single test case that describes one particular aspect of an application's functionality or behavior. | A step is a distinctive assertion or action that is carried out as a part of a scenario. |
| Consists of one or more steps. | Consists of only one step. |
| Can have a variety of stages with various contexts. | Can have various scenarios with various steps. |
| Used to verify and validate the feature's acceptance criteria. | Used to specify particular test procedures to verify the system's operation. |
| Aids in understanding the system's overall behavior. | Aids in understanding a particular behavior of the system. |

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

|  |  |
| --- | --- |
| **JUnit** | **Cucumber** |
| JUnit is a unit testing framework that is generally used to test isolated code modules or units. It offers a collection of annotations and assertions for writing and running tests for Java applications. | Cucumber is a framework for behavior-driven testing that focuses on evaluating a system's overall behavior by specifying scenarios and actions using Gherkin syntax in natural language. |
| JUnit uses annotations to specify test methods (@Test, @Before, @After). | Cucumber defines scenarios using the "given-when-then" syntax (Gherkin language) |
| The reading and understanding of tests can be difficult, particularly for non-technical stakeholders. | Scenarios are stated in simple terms so that non-technical stakeholders can understand them. |
| Can integrate with testing tools like Mockito and Selenium. | Can integrate with testing tools like Selenium and RestAssured |
| JUnit delivers basic test reporting | Cucumber provides a detailed test reporting in JSON, HTML and other formats |

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

|  |  |
| --- | --- |
| **Cucumber** | **RSpec** |
| Cucumber is used for acceptability testing and end-to-end testing | RSpec is mainly used for unit testing. |
| Cucumber's natural language syntax facilitates easier cooperation between technical and non-technical stakeholders. | RSpec's syntax may be more difficult for non-technical stakeholders to understand |
| Cucumber focuses on testing the behavior of an entire system or application | RSpec focuses on testing individual code units. |

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

|  |  |
| --- | --- |
| **Cucumber** | **TestNG** |
| Cucumber is used for acceptability testing | TestNG is mostly used for unit testing. |
| Cucumber supports a variety of programming languages | TestNG is focused on Java-based testing. |
| Cucumber provides a more natural language approach using Gherkin syntax for defining tests | TestNG is more developer-centric and relies on annotations. |
| Cucumber offers a behavior-driven development (BDD) strategy that stresses cooperation between developers, testers, and business stakeholders. | TestNG does not have a behavior-driven development (BDD) focus |

**What is the difference between a hook and a step definition in Cucumber?**

|  |  |
| --- | --- |
| **Hook in Cucumber** | **Step Definition in Cucumber** |
| A hook in Cucumber is a section of code that runs before or after a scenario or feature. | A step definition is a piece of code that specifies a step's behavior in a feature file. |
| Suitable for setup and takedown operations like starting a browser or cutting off a database connection. | Can be used to carry out any automation job, including communicating with UI elements and executing API calls. |
| Using context objects or global variables, data can be shared between scenarios or steps. | Can leverage Cucumber's built-in data sharing features, such as scenario outline tables or background steps, but cannot share data between steps or scenarios. |
| Can be used to conditionally change or skip scenarios depending on specific circumstances. | Can be used to construct conditional logic within a step's implementation, but cannot be used to skip or change scenarios. |

**What is the difference between a scenario and a feature file in Cucumber?**

The crucial Cucumber components Scenario and Feature both have a specific function. The key variations between them are as follows:

|  |  |
| --- | --- |
| **Feature File** | **Scenario** |
| A text file called a feature file contains a detailed description of a feature that needs to be tested. | A single test case in the Gherkin language is known as a scenario. |
| It outlines the software's functioning, which needs to be tested. | It outlines the specific steps to be taken to test a specific functionality. |
| It can involve a variety of scenarios, each of which tests a distinct function. | It usually comprises of a number of steps that are carried out in a particular order. |
| The scenarios are placed within a context, and they are arranged into logical groups. | To organise and group scenarios, it can be connected to one or more tags. |

**What is the difference between a tag and a data table in Cucumber?**

|  |  |
| --- | --- |
| **Tag** | **Data Table** |
| Tag begins with the "@" symbol | Data Tablr starts with " |
| Tags organize and filter scenarios | Data Table passes numerous parameters to a scenario |
| A scenario may contain several tags. | A scenario can contain only one data table. |
| The step definitions don't use tags. | With step definitions, data tables are used as arguments. |

**What is the difference between a test scenario and a test case in Cucumber?**

|  |  |
| --- | --- |
| **Test Scenario** | **Test Case** |
| A test scenario is a specific instance of a feature. | A test case is a set of conditions or variables that a tester will use to verify whether or not a system is functioning properly. |
| Test Scenario is defined using Gherkin syntax and consists of a collection of stages that specify the activities and expected outcomes for specific system behavior. | Test Case is often defined by a QA engineer and contains comprehensive instructions on how to test a particular system feature or behavior. |

**What is the purpose of using Cucumber?**

Cucumber is a tool that supports Behavior Driven Development (BDD). This agile software development methodology encourages cooperation between developers, testers, and business stakeholders to guarantee that the software satisfies the requirements and caters to the needs of the end users. The main goal of utilizing Cucumber is to create, run, and automate test cases that clearly and simply describe how the product behaves for everyone engaged in the development process. The development team, the testing team, and the business stakeholders can all communicate and understand one another more easily because of Cucumber's ability to create feature files that act as a common language.

Eventually, the purpose of deploying Cucumber is to increase teamwork, communication, and development process effectiveness while guaranteeing large quality and dependability of the software output.

**How do you debug a Cucumber test?**

A Java debugger can be used to set breakpoints in the step definition code and walk through the code to find any problems when debugging a Cucumber test. To print out intermediate values and messages while the test is being run, another strategy is to include log statements in the step definition code. Also, when executing tests from the command line, Cucumber offers the option to run them in debug mode by adding the “- - debug” switch.

**How do you debug failing Cucumber tests?**

You can use logging and debugging tools given by your programming language or IDE to debug failing Cucumber tests. Also, the Cucumber "—debug flag" enables you to execute tests in debug mode, allowing you to step through each test step and determine precisely where a failure occurred.

**What is the difference between a soft assertion and a hard assertion in Cucumber?**

|  |  |
| --- | --- |
| **Soft Assertion** | **Hard Assertion** |
| continues running tests even after a failure | stops testing after a failure |
| Logs the error message before moving on to the next assertion. | logs the error message and stops the test. |
| can be helpful in situations when several claims need to be verified before the test is stopped. | useful in situations where the failure of a single assertion signals the failure of a test |

**How do you use Cucumber hooks?**

A Cucumber Hook is executed at specific times during the Cucumber test cycle. Hooks are used to perform post-actions after a scenario or step has been completed and to set up preconditions before they are met. Many hook annotations are provided by Cucumber, which can be used to specify the intended behavior. A few are listed below.

1. @Before: This hook is executed before each scenario or scenario step. It can be used to do any required setup tasks or to set up the test environment.
2. @After: This hook is used to determine whether a scenario or a step in a scenario is to be executed. It can be used to teardown the test environment or perform cleanup operations.
3. @BeforeStep - It is used before each step in a scenario. It can be used to carry out particular tasks before each stage is carried out.
4. @AfterStep: In a scenario, this hook is used after each step. It can be used to carry out particular tasks upon the completion of each stage.

**How do you use Cucumber with Selenium?**

It is necessary to establish the step definitions that interact with the Selenium WebDriver to carry out operations on the web page when using Cucumber with Selenium. With a single feature file, you can automate several scenarios by passing data to the step definitions using Cucumber DataTable.

**Advanced Cucumber Interview Questions**

Candidates with a solid knowledge of Cucumber's fundamental ideas, syntax, and use should feel comfortable answering advanced level Cucumber interview questions. The below questions aim to assess the candidate's overall understanding of Cucumber and their ability to use Cucumber to resolve challenging testing scenarios.

**How do you pass parameters to a step definition in Cucumber?**

Using regular expressions, you can capture the parameter values from a step in the feature file and then give them as arguments to the appropriate method in the step definition in Cucumber. Here's an illustration:

In the feature file:

Given the user has entered "John" **as** their name

In the step definition:

@Given("^the user has entered "([^"]\*)" **as** their name$")

public void enterName(String name) {

// Use the captured name parameter in the step definition code

}

In this illustration, the string name is captured by the regular expression "(["]\*)" and passed as an argument to the enterName method.

**How do you run a specific scenario in Cucumber?**

The @scenario name tag in your feature file can be used to identify the scenario you want to run in Cucumber. The tag can then be specified in your command line by using the -t or —tags option and the tag name. You can use the following command to run a scenario, for instance, if it has the tag @login:

cucumber -t @login}

Only the scenarios with the @login tag will be executed. A comma can be used to divide tags if you have more than one. For instance, you can use the following command to execute scenarios that have both the @login and @smoke test tags:

cucumber -t @login,@smoke\_test

Just the scenarios with the tags @login or @smoke test will be executed by this.

**How do you skip a scenario in Cucumber?**

In Cucumber, you can skip a scenario by adding the @ignore tag before the method that needs to be skipped. The method marked with the @ignore tag will be ignored when you run the Cucumber tests. Here's an illustration:

@ignore

Scenario: This scenario will be skipped

Given some precondition

When some action is taken

Then some expected result is obtained

The scenario in the aforementioned example has the tag "@ignore," meaning that Cucumber tests will skip it.

**How do you integrate Cucumber with Jenkins?**

To integrate Cucumber with Jenkins, you can follow these steps:

1. Install the Jenkins Cucumber plugin.
2. For your Cucumber project, create a Jenkins job.
3. Set up the Jenkins task to use a build script or the command line to run the Cucumber tests.
4. Set up the Cucumber plugin in the Jenkins job to generate Cucumber reports.
5. You can either manually start the Jenkins task or schedule it to execute at a particular time.

pipeline {

agent any

stages {

**stage**('Build') {

steps {

sh 'mvn clean install'

}

}

**stage**('Test') {

steps {

sh 'mvn test'

}

post {

always {

cucumber htmlReports: **true**, jsonReports: **true**

}

}

}

}

}

This Jenkinsfile creates and tests a Maven-based Cucumber project and uses the Cucumber plugin to produce HTML and JSON reports. This file can be modified to suit the needs of your project.

**How do you handle exceptions in Cucumber?**

Cucumber's try-catch block in the step specification can be used to handle exceptions. The Cucumber scenario will fail if an exception arises in a phase, and the console output will show what went wrong. An illustration of how to handle an exception in a step definition is provided here:

@When("I perform some action")

public **void** **performAction**() {

**try** {

// perform the action that may throw an exception

} **catch** (Exception e) {

// handle the exception

}

}

As an alternative, you can specify a unique exception handler using the @CucumberOptions annotation. For instance, you would include the subsequent annotation to your runner class to use the MyExceptionHandler custom exception handler class:

@RunWith(Cucumber.class)

@CucumberOptions(

// other options

plugin = {"com.example.MyExceptionHandler"}

)

public **class** **MyCucumberTest** {

// test code

}

In this case, all exceptions thrown during the Cucumber test would be handled by the MyExceptionHandler class.

**How do you write a scenario outline in Cucumber?**

To write a scenario outline in Cucumber, you can follow these steps:

1. The Feature keyword should come first, followed by a concise description of the feature you want to test.
2. To begin the outline, use the keyword Scenario Outline.
3. Using placeholders for the variable data, write the steps of the scenario outline. The names should be descriptive, and the placeholders should be enclosed in < >.
4. Create a table called Examples containing rows of sample data you may use in your testing. With precise information, each row should fill in the blanks in the Scenario Outline phases.
5. Add a.feature extension to the feature file while saving it.

Feature: Search functionality

As a user

I want to be able to search **for** products

So that I can find the products I need

Scenario Outline: Search **for** a product

Given I am on the home page

When I search **for** "<product>"

Then I should see search results **for** "<product>"

Examples:

| product |

| iPhone |

| iPad |

| MacBook |

By using the Examples table to fill in the blank for the product name, the Scenario Outline in this example evaluates the search functionality for various products. Each example row uses the same Given, When, and Then stages, streamlining the test and making it simpler to read.

**How do you define a background in Cucumber?**

A background in Cucumber is a set of functions that each scenario in a feature file uses. It is employed to specify the prerequisites that must be satisfied before the scenarios can be carried out.

In Cucumber, you may define a background by using the “Background” keyword followed by the steps you want to describe. Here's an illustration:

Feature: User registration

Background:

Given the user is on the registration page

Scenario: Valid registration

When the user enters valid information

Then the user should be registered successfully

Scenario: Invalid registration

When the user enters invalid information

Then the user should see an error message

The Background keyword is used to indicate a single step in the example above, namely that the user is now on the registration page. Before both scenarios in the feature file, this step will be carried out.

**How do you handle dynamic input in Cucumber?**

Cucumber can handle dynamic input by using a scenario outline with an Examples table. Here is a brief sample of code:

Scenario Outline: Login **with** dynamic credentials

Given User is on login page

When User enters "<username>" and "<password>"

And User clicks on login button

Then User should be logged **in** successfully

Examples:

| username | password |

| user1 | pass123 |

| user2 | pass456 |

In this example, the steps of the scenario were defined using the scenario outline, and dynamic input was then added using the Examples table. The table contains the actual values for the placeholders "username" and "password." For each entry in the table, Cucumber will automatically construct a scenario, replacing the placeholders in the scenario outline with values from the table.

**How do you execute a scenario using a different browser in Cucumber?**

Use the @driver tag in your scenario, followed by the name of the chosen browser, to run a scenario in Cucumber using a different browser. Here's an illustration:

@driver=firefox

Scenario: User logs **in** **with** valid credentials using Firefox

Given the user is on the login page

When the user enters valid credentials

And the user clicks the login button

Then the user should be redirected to the home page

The @driver=firefox tag in the example above indicates that the scenario should be carried out using the Firefox web browser. Any alternative browser name, such as "chrome," "safari," or "edge," can be used in place of "firefox."

You must set up your test environment to support various browsers before launching your test suite with the required browser-specific command-line options or configuration settings to run the scenario using the selected browser.

**How do you handle multiple data sets in Cucumber?**

You can use the Scenario Outline and Examples tables in your feature file to handle numerous data sets in Cucumber.Here's an illustration:

Scenario Outline: Search using different keywords

Given I am on the search page

When I search **for** "<keyword>"

Then I should see results **for** "<keyword>"

Examples:

| keyword |

| cucumber |

| selenium |

| java |

In this example, we define a search scenario for several keyword combinations. In the scenario phases, we use the placeholder "keyword" and give it various values from the Examples database. For each entry in the table, Cucumber will automatically create a scenario, replacing the placeholders with the actual values.

**How do you write a comment in a feature file in Cucumber?**

Cucumber allows you to add comments to feature files by using the "#" sign and then the comment text. Here's an illustration:

Feature: Login

As a registered user

I want to login to my account

So that I can access my dashboard

# This is a comment

Scenario: Valid login

Given I am on the login page

When I enter valid credentials

And click the login button

Then I should be redirected to the dashboard page

The comment is denoted in this example by the hash mark ("#") and is typed on a separate line before the term "Scenario."

**How do you pass arguments to Cucumber from the command line?**

You can use the "—tags" option and the name of the tag you want to run to send parameters to Cucumber from the command line. For instance, if you want to execute only the scenarios you have tagged with @smoke, you can pass the —tags option with the following details:

cucumber --tags @smoke

To execute scenarios with any of the specified tags, you can also pass multiple tags, separated by commas. For instance:

cucumber --tags @smoke,@regression

**How do you parameterize a step definition in Cucumber?**

In Cucumber, you may use regular expressions and capture groups to extract values from a step definition and parameterize it. The step definition method can then accept the captured values as parameters. Consider the following as an example scenario:

Scenario: Search **for** a product

Given I am on the search page

When I search **for** "cucumber"

Then I should see search results **for** "cucumber"

You can parameterize the command "Then I should get search results for "cucumber"" by extracting the product name using a regular expression and a capture group:

@Then("I should see search results for "(.\*)"")

public **void** **verifySearchResults**(String productName) {

// Verify the search results for the given product name

}

In this example, any string of characters can be matched by the regular expression "(.\*)," and the captured value is supplied as a parameter to the verifySearchResults method. In this instance, the parameter productName will be set to "cucumber".

**How do you define a tag in a feature file in Cucumber?**

Cucumber uses tags to categorize scenarios, features, or specific steps. These tags can then be used to execute a particular collection of procedures or features. Add the tag symbol (@) and tag name to the line above the quality, scenario, or particular step to define a tag in a feature file. To specify a tag for a feature, for example:

@myTag

Feature: My Feature

As a user

I want to **do** something

So that I can achieve a goal

The tag name in the above instance is "@myTag".

**How do you ignore a scenario using tags in Cucumber?**

Using the @ignore tag in the scenario in Cucumber, you can ignore a procedure using tags. The scenarios marked with the @ignore tag will not be executed when running the Cucumber test. Here's an illustration of how tags in Cucumber can be used to ignore a scenario:

Feature: My feature

@ignore

Scenario: My scenario

Given I have some precondition

When I perform some action

Then I should see some result

Scenario: My other scenario

Given I have some other precondition

When I perform some other action

Then I should see some other result

The first scenario in this example is given the @ignore tag. The second scenario will be the only one that is conducted when you run the Cucumber test.

**How do you run Cucumber tests in parallel?**

Using a test runner like JUnit or TestNG and configuring it to execute tests concurrently will allow us to run Cucumber tests in parallel. Here is an illustration of coding using TestNG:

**import** cucumber.api.CucumberOptions;

**import** cucumber.api.testng.AbstractTestNGCucumberTests;

@CucumberOptions(features = {"src/test/resources/features"},

glue = {"com.example.steps"})

public **class** **TestRunner** **extends** **AbstractTestNGCucumberTests** {

// nothing to do here, TestNG will run the Cucumber tests in parallel

}

You can define the location of the feature files and step definitions in the @CucumberOptions annotation. Then, develop a TestNG test runner by extending AbstractTestNGCucumberTests. In accordance with the settings in the testng.xml configuration file, TestNG will automatically run the Cucumber tests in parallel.

**How do you implement a data-driven testing approach in Cucumber?**

Cucumber's Scenario Outline and Examples tables can be used to build data-driven testing. You can create a template scenario with placeholders in the Scenario Outline that can be updated with test data from the Examples table. Here is an illustration of data-driven testing in action with Cucumber:

Scenario Outline: Login functionality **with** multiple users

Given I am on the login page

When I enter "<username>" and "<password>"

And I click on login button

Then I should be logged **in**

Examples:

| username | password |

| user1 | pass1 |

| user2 | pass2 |

| user3 | pass3 |

The Scenario Outline template in this example specifies the stages for the login feature, and the Samples table contains various sets of test data. For each entry in the Examples table, Cucumber will create a unique case during execution, substituting the placeholders with the actual values.

**How do you handle timeouts in Cucumber?**

You can utilize the Ruby Timeout library's Timeout class to manage timeouts in Cucumber. An example of code that sets a step's timeout to 10 seconds is shown below:

require 'timeout'

Timeout.timeout(10) **do**

# code to be executed **with** a timeout **of** 10 seconds

end

The Timeout.timeout method in the code above takes a block of code that will be run during the designated timeout duration. A Timeout::Error will be produced if the code execution takes longer than the allotted period. This error can be restored, and the code can handle it correctly.

**How do you handle pop-up windows in Cucumber?**

Cucumber can handle pop-up windows by switching the driver focus to the pop-up window using Selenium's switch to method. Here is a brief piece of code that shows how to manage a pop-up window:

# Click on a button that opens a pop-up window

button = find\_element(:id, "popup-button")

button.click

# Switch to the pop-up window

popup\_window = driver.window\_handles.last

driver.switch\_to.window(popup\_window)

# Perform actions on the pop-up window

# ...

# Switch back to the main window

main\_window = driver.window\_handles.first

driver.switch\_to.window(main\_window)

In this illustration, first locate and select the button that opens the pop-up window. The pop-up window's handle is then obtained using driver.window handles.last, and the driver's attention is switched to the window using driver.switch to.window. After performing the actions on the pop-up window, use driver to return the driver's focus to the primary window. driver.switch to.window and window handles.first.

**How do you handle dynamic web elements in Cucumber?**

You can use Cucumber's waitFor method to manage dynamic web components from the Selenium WebDriver library. Before moving on to the next step, this method waits for a specified condition to be true. An illustration of the use of waitFor in a step definition is given here:

@When("^I click on the button with id "([^"]\*)"$")

public void clickButtonWithId(String buttonId) {

WebElement button = driver.findElement(By.id(buttonId));

new WebDriverWait(driver, 10).until(ExpectedConditions.elementToBeClickable(button));

button.click();

}

In this case, the expected condition is elementToBeClickable. Before clicking on a button, it waits for it to be clickable. Because it makes sure the element is available and clickable before interacting with it, this helps in handling dynamic web elements.

**How do you handle multiple windows in Cucumber?**

You can leverage Selenium's window handles method in Cucumber to manage many windows. Here's an illustration:

@When("^I click on a link to open a new window$")

public **void** i\_click\_on\_a\_link\_to\_open\_a\_new\_window() throws InterruptedException {

String mainWindowHandle = driver.getWindowHandle();

// Click on the link that opens a new window

WebElement newWindowLink = driver.findElement(By.id("newWindowLink"));

newWindowLink.click();

// Wait for the new window to open and switch to it

WebDriverWait wait = **new** WebDriverWait(driver, 10);

wait.until(ExpectedConditions.numberOfWindowsToBe(2));

Set<String> windowHandles = driver.getWindowHandles();

**for** (String handle : windowHandles) {

**if** (!handle.equals(mainWindowHandle)) {

driver.switchTo().window(handle);

**break**;

}

}

}

@Then("^I can perform actions in the new window$")

public **void** **i\_can\_perform\_actions\_in\_the\_new\_window**() {

// Do something in the new window

driver.findElement(By.id("newWindowInput")).sendKeys("Hello World");

}

In this illustration, after obtaining the handle of the primary window, we click a link to launch a new window. Next, after a short wait, move to the new window with the driver. window().switchTo().handle() method. The new window is now ready for action when needed.

**How do you handle frames in Cucumber?**

The switchTo() method given by Selenium WebDriver can be used to switch to the frame and carry out functions within it when handling frames in Cucumber.Here is an illustration of the code:

//Switch to frame by name or ID

driver.switchTo().frame("frameNameOrID");

//Perform actions on elements within the frame

driver.findElement(By.id("elementID")).click();

//Switch back to default content

driver.switchTo().defaultContent();

In the above illustration, you first change to a frame by utilizing its name or ID. Then, take a specific action on a frame element. Finally, you return to the default content that is outside of the frame.

**How do you handle exceptions in Cucumber?**

Exceptions in Cucumber can be handled by including a try-catch block in the step specification. If an exception arises while a step is being executed, it can be caught in the catch block and handled appropriately. An illustration of how to handle exceptions in a step definition is given here:

@When("I do something that may throw an exception")

public **void** **doSomething**() {

**try** {

// code that may throw an exception

} **catch** (Exception e) {

// code to handle the exception

}

}

The step definition in this example tries to execute specific codes that might produce an exception. If an exception arises, it is caught in the catch block so that the correct response can be executed.

**How do you organize your Cucumber feature files?**

Organizing your feature files according to the functionality or module they cover is a good idea. For each module, you can make a directory and place the feature files that go with it. Finding and managing the files are made simpler as a result.

- features/

- authentication/

- login.feature

- registration.feature

- user\_management/

- create\_user.feature

- delete\_user.feature

- orders/

- create\_order.feature

- view\_order.feature

The feature files in this example are organized into functional groups and stored in subdirectories. The feature files are kept small and focused on a particular functionality