

Cucumber Testing Tutorial

Cucumber is a widely used tool for Behaviour Driven Development because it provides an easily understandable testing script for system acceptance and automation testing.

What is Cucumber?

Cucumber is a [Behavior Driven Development](https://www.javatpoint.com/cucumber-behavior-driven-development) tool used to develop test cases for the behavior of software's functionality. It plays a supporting role in automated testing.

In other words,

"Cucumber is a software tool used by the testers to develop test cases for the testing of behavior of the software."

Cucumber tool plays a vital role in the development of acceptance test cases for automation testing. It is mainly used to write acceptance tests for web applications as per the behavior of their functionalities.

It follows a **BDD** (Behavior Driven Development) framework to observe the behavior of the software's functionalities.

In the Cucumber testing, the test cases are written in a simple English text, which anybody can understand without any technical knowledge. This simple English text is called the [Gherkin language](https://www.javatpoint.com/cucumber-testing-gherkin-language).

It allows business analysts, developers, testers, etc. to automate functional verification and validation in an easily readable and understandable format (e.g., plain English).

We can use Cucumber along with Watir, Selenium, and Capybara, etc. It supports many other languages like **PHP**, **Net**, **Python**, **Perl**, etc.

What is BDD?

BDD (Behavioral Driven Development) is a software development approach that was developed from **Test Driven Development (TDD)**.

BDD includes test case development in the form of simple English statements inside a [feature file](https://www.javatpoint.com/feature-file-in-cucumber-testing), which is human-generated. Test case statements are based on the system's behavior and more user-focused.

BDD is written in simple English language statements rather than a typical programming language, which improves the communication between technical and non-technical teams and stakeholders.

Which language is used in cucumber?

**Cucumber** tool was originally written in the "**Ruby**" programming language. It was exclusively used only for testing of Ruby as a complement to the **RSpec** BDD framework.

But now, Cucumber supports a variety of different programming languages including Java, JavaScript, PHP, Net, Python, Perl, etc. with various implementations. In Java, it supports **native JUnit**.

## Basic Terms of Cucumber

* Feature File
* Features
* Tags
* [Scenario](https://www.javatpoint.com/test-scenario)
* Gherkin Language
* Step Definition

AD

## How does Cucumber Testing Works?

Cucumber test cases are written parallel with the code development of software. These test cases are called step in a Gherkin Language.

* Firstly, Cucumber tool reads the step written in a Gherkin or plain English text inside the feature file.
* Now, it searches for the exact match of each step in the step definition file. When it finds its match, then executes the test case and provides the result as pass or fail.
* The code of developed software must correspond with the BDD defined test scripts. If it does not, then code refactoring will be required. The code gets freeze only after successful execution of defined test scripts.

AD

## Software tools supported by Cucumber

The piece of code to be executed for testing may belong to different software tools like **Selenium**, **Ruby on Rails**, etc. But cucumber supports almost all popular software platforms, and this is the reason behind Cucumber's popularity over other frameworks such as **JDave**, **Easyb**, **JBehave**, etc. Some Cucumber supported tools are given below:

* Ruby on Rails
* Selenium
* PicoContainer
* [Spring Framework](https://www.javatpoint.com/spring-tutorial)
* Watir

## Advantages of Cucumber Tool

* The main focus of the Cucumber Testing is on the end-user experience, as the success of the software ultimately depends on the end-user experience.
* The test case writing is very easy and understandable.
* It provides an end-to-end testing framework, unlike other tools.
* It supports almost all popular different languages like Java.net, JavaScript Ruby, PHP, etc.
* It works as a bridge between business and technical language, and this bridge is sustainable because of test cases written in a plain English text.
* The testing environment set up and execution both are very quick and easy.
* It is a well efficient tool for testing.

# Difference between Cucumber and Selenium

Before knowing the differences between cucumber and Selenium first, let's take a quick introduction of these.

## Cucumber

It is a **Behavior Driven Development** tool used to develop test cases for the behavior of software's functionality. It plays a supporting role in automated testing.

In other words, cucumber is a software tool used by the testers to develop test cases for the testing of behavior of the software.

Cucumber tool makes the test scripts very easy, because the test cases are written in a simple English text, which anybody can understand without any technical knowledge. This simple English text is called the Gherkin language.

It allows business analysts, developers, testers, etc. to automate functional verification and validation in an easily readable and understandable format (e.g., plain English).

Cucumber tool can be used along with various testing tools such as Watir, Selenium, and Capybara, etc.

**Cucumber** tool was originally written in the "**Ruby**" programming language. It was exclusively used only for testing of Ruby as a complement to the **RSpec** BDD framework.

But now, Cucumber supports a variety of different programming languages including Java, JavaScript, PHP, Net, Python, Perl, etc. with various implementations. In Java, it supports **native JUnit**.

## Selenium

Selenium is one of the most widely used open-source Web UI (User Interface) automation testing suite. Selenium supports automation across different browsers, platforms, and programming languages.

Selenium can be easily deployed on platforms such as Windows, Linux, Solaris, and Macintosh. Moreover, it supports OS (Operating System) for mobile applications like iOS, windows, mobile, and android.

Selenium supports a variety of programming languages through the use of drivers specific to each language. Languages supported by Selenium include **C#**, **Java**, **Perl**, **PHP**, **Python** and **Ruby**. Currently, Selenium Web driver is most popular with Java and C#. Selenium test scripts can be coded in any of the supported bprogramming languages and can be run directly in most modern web browsers. Browsers supported by Selenium include Internet Explorer, Mozilla Firefox, Google Chrome, and Safari.

**Now, let's have a look at fundamental differences between Cucumber and Selenium:**

|  |  |
| --- | --- |
| **Cucumber** | **Selenium** |
| It is a **Behavior Driven Development** tool used to develop test cases for the behavior of software's functionality. | It is an automated testing tool. |
| Cucumber is a free or open-source BDD (Behavior Driven Development) tool. | Selenium is also a free or open-source testing tool. |
| Cucumber is a BDD supported tool. | Selenium is a both Functional and Performance (Selenium Grid) testing tool. |
| Cucumber framework supports many languages, such as Java, Scala, Groovy, etc. beyond Ruby. | Selenium also supports many languages, such as Java, .Net, etc. |
| Cucumber includes both testers and developers to write automation steps. | Like Cucumber, Selenium also includes both testers and developers to write automation steps. |
| Cucumber is used to test only web applications. | Like Cucumber, Selenium also used to test only web applications. |
| Cucumber testing is less reliable as compared to Selenium and QTP. | The process of Selenium makes testing more reliable and dependable. |
| Cucumber works very fast in Plugin. | Selenium works slower in Plugin than Cucumber. |

# Behavior Driven Development

BDD (Behavioral Driven Development) is a **software development** approach that was developed from **Test Driven Development (TDD)**.

BDD includes test case development on the basis of the behavior of software functionalities. All test cases are written in the form of simple English statements inside a [feature file](https://www.javatpoint.com/feature-file-in-cucumber-testing), which is human-generated. Acceptance test case statements are entirely focused on user actions.

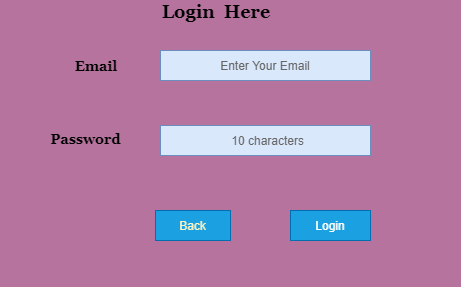
BDD is written in simple English language statements, not in a typical programming language. BDD improves communication between technical and non-technical teams and stakeholders.

Let's understand through an example, how we can develop [test cases](https://www.javatpoint.com/test-case) on the basis of the behavior of a particular function.

In the following example, we are going to take the **login function** of a web application.

## Example:

In order to ensure the working of Login Functionality, we are developing acceptance test cases on the basis of BDD.



**Feature:** Login Function

To enter in the System

User must be able to

Access software when login is successful

**Scenario:** Login

**Given** User has its Email

**And** Password

**When** User enters the correct Email and Password

**Then** It should be logged in

**Scenario:** Unsuccessful Login

**When** User enters either wrong Email or Password

**Then** It should be reverse back on the login page with an error message

## Need to Choose BDD

TDD works satisfactorily unless the business owners are familiar with the use of the [unit testing](https://www.javatpoint.com/unit-testing). Also, their technical skills should strong enough, which is not always possible.

In these circumstances, BDD is advantageous because test cases are written in a common English language, which is easily understandable by all stakeholders.

The familiar, easily understandable language is the most significant advantage of using BDD because it plays a vital role in cooperation between technical and non-technical teams to perform a task with better efficiency.

## Characteristics of BDD

**Strong collaboration**

BDD provides a strong collaboration between involved parties. It is just because of easy test cases which are written in the English language. In cucumber testing, stockholders play a vital role in constructive discussions as only they know the expectations from the software.

**High Visibility**

Everyone gets strong visibility in the progress of the project due to the easy English language.

**The software design follows the business value**

BDD gives great importance to business value and needs. By setting priorities with the client, depending on the value provided by them, developers are able to give a better result because they have a strong understanding of how the customer thinks.

**The Ubiquitous Language**

As mentioned earlier, test cases are written in the ubiquitous language, which is understandable by all the members of the team, whether they are from a technical field or not. This helps to reduce misconceptions and misunderstanding between the members related to concepts. Ubiquitous language makes easy joining of new members into the working.

**Software development meets the user need.**

BDD focuses on the business's needs so that users can be satisfied, and of course, satisfied users imply a growing business. With BDD, tester focuses on the behavior which has more impact than the implementation.

**More confidence from the developers' side**

The teams using BDD are generally more confident because they do not break the code, and when it comes to their work, a better forecast is done.

**Lower Costs**

By improving the quality of the code, BDD basically reduces the cost of maintenance and minimizes the risks of the project.

# Feature File in Cucumber Testing

The feature file is the essential segment of cucumber tool, which is used to write acceptance steps for automation testing. Acceptance steps generally follow the application specification.

A feature file is usually a common file which stores [feature](https://www.javatpoint.com/feature-in-cucumber-testing), [scenarios](https://www.javatpoint.com/scenario-in-cucumber-testing), and feature description to be tested.

The feature file is an entry point, to write the cucumber tests and used as a live document at the time of testing.

The extension of the feature file is "**.feature**". Each functionality of the software must have a separate feature file.

## Example:

In order to ensure the working of Login Functionality, we are implementing the cucumber test by creating a feature file. It will verify whether the Login Functionality is working properly or not.

**Feature:** Login

**Scenario:** Login Functionality

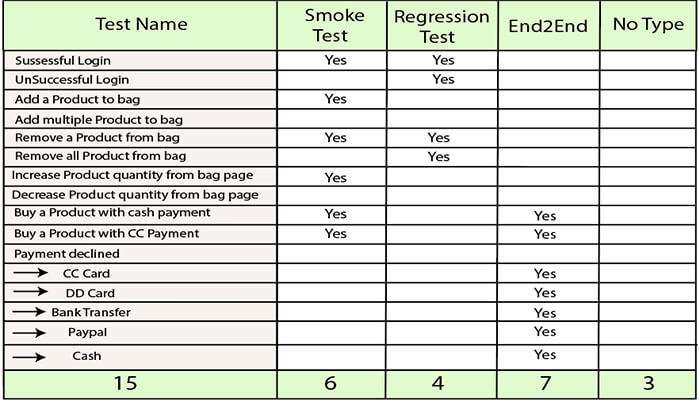
**Given** user navigates to the website **javatpoint.com**

**And** there user logs in through **Login Window** by using Username as "USER" and Password as "PASSWORD"

**Then** login must be successful.

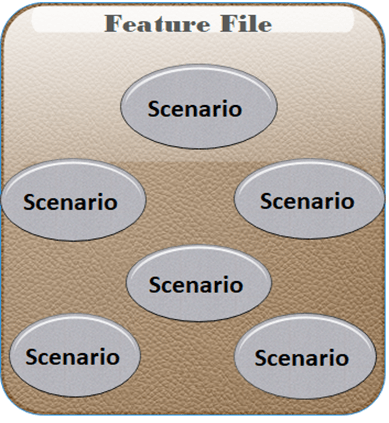
After performing the automation testing, a table is created as a result of automation testing. This table is used in tags.

### **The resulting table looks like the following table:**



## Feature file with Multiple Scenario

Feature file can contain multiple scenarios or scenario outlines. We can write all possible Scenarios of a particular feature in a feature file.



By using the keyword "Scenario" or "Scenario Outline", One Scenario can be separated from another.

However, a single feature file can contain any number of scenarios but focuses only on one feature such as registration, login etc at a time. Therefore, it is better to keep the scenarios related to a particular feature in a single feature file.

Scenarios can be executed parallel, or you can execute them together in a group. Let's take an example for more clarity:

### **Example:**

**Feature File 1:**

**Feature:** Registration

**Background:**

**Given** user on the homepage

**And** user follows "**Sign in**"

**@regression**

**Scenario:** Create a New User

**When** user fills "registration email textbox" with "chitrali.sharma27@gmail.com"

**And** user clicks "create an account button"

**And** user enters the following details

| First Name | Chitrali|

| Last Name | Sharma|

| Password | Inquiry@1234 |

| Date | 17| | Month | 02| | Year | 1992 |

**And** user clicks "register button"

**Scenario:** User does not follow form validations

**When** user enters wrong characters

**Then** error message displayed with invalid password

**And** user returns back on registration page

**Feature File 2:**

**Feature:** Login

**Background:**

**Given** user on the login page

**And** user follows "**Log in**"

**@regression @smoke**

**Scenario:** Verification of Login Function

**Given** user on the Login Page

**And** user enters "email address" with "**chitrali.sharma27@gmail.com**"

**And** user enters "password" with "**Inquiry@1234**"

**And** user click "**log in**" button

**Then** user should see "**My Account**"

**Scenario: Unsuccessful login**

**Given** user on the Login Page

**And** user enters "email address" with "**chitrali.sharma27@gmail.com**"

**And** user enters "password" with "**qsder@1234**"

**And** user clicks "**login**" button

**Then** error message displayed with wrong password

**And** user returns back on login page

## Comments in Feature File

If we do not need to execute a particular scenario at a time, then we can comment that scenario.

In Eclipse, to comment a multi-line or use block comment first select all the line to be commented and then press **Ctrl + /**. Similarly, to remove comments, we need to press **Ctrl + \**. Other IDEs may contain other shortcuts to do this.

While commenting any scenario, do not forget to comment the complete scenario. Otherwise, remaining lines of scenario which are not commented will be considered as a part of the previous scenario.

# How to create Feature File

There are several approaches to create a feature file in different IDEs, here we are creating it in Eclipse IDE.

We can create a feature file with the "**.feature**" extension.

Following are the steps to create a feature file by using **eclipse IDE**:

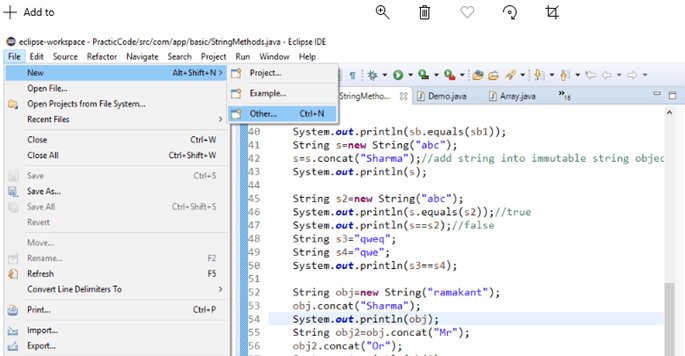
1. In order to create a feature file in eclipse, go to the **File** option at the left side of the window then select **New**.

When you click the **New**, you will get the following three options:

* **Project**
* **Example**
* **Other**

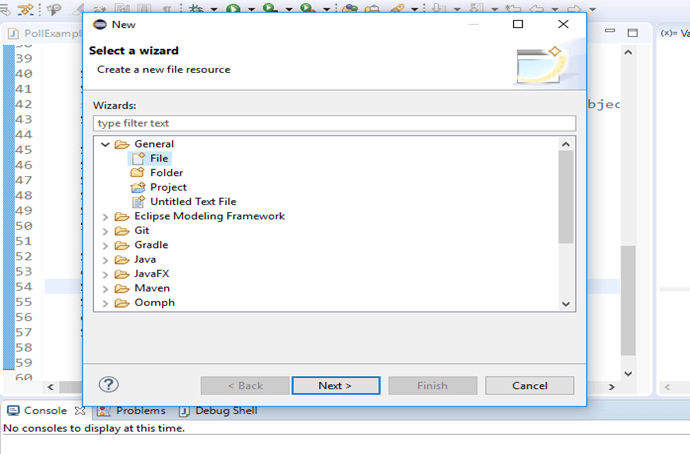
Select **Other** option from these three options.

### **Consider the following image:**



2. After selecting **Other** option, you will get several options, from these options, select **General < File** option, and then click **Next**.

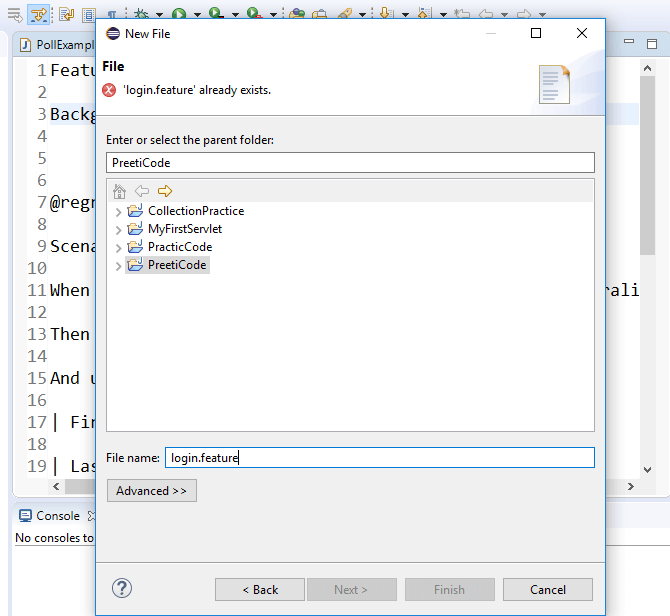
### **Consider the following image:**



3. After clicking the **Next**, select the project inside which you want to create a feature file.

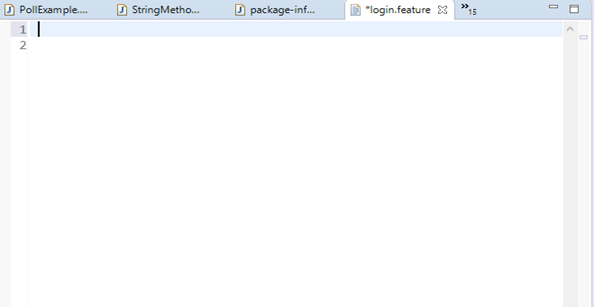
After selecting the project, you can create a feature file by giving a name and "**.feature**" extension. You can provide any name to the feature file on your own choice. After providing a name, click on the **Finish** button.

### **Consider the following image:**



4. Now, the created feature file will appear inside the project.

### **Consider the following image:**



The generated feature file will be similar to the above image. In it, we can write features, scenarios, and feature description of the web application to be tested.

# What is tag in Cucumber testing?

1. [What's the need for tags in cucumber testing?](https://www.javatpoint.com/cucumber-tags#need-for-tags)
2. [How to create a set of multiple tags in cucumber testing?](https://www.javatpoint.com/cucumber-tags#create-a-set-of-multiple-tags)
3. [How to ignore tags in cucumber testing?](https://www.javatpoint.com/cucumber-tags#ignore-tags)

In Cucumber, tags are used to associate a test like [smoke](https://www.javatpoint.com/smoke-testing), [regression](https://www.javatpoint.com/regression-testing) etc. with a particular scenario.

Tag fulfils the following purposes:

* If we have many [scenarios](https://www.javatpoint.com/scenario-in-cucumber-testing) in the [feature file](https://www.javatpoint.com/feature-file-in-cucumber-testing), to keep them in one group, we use tags in Cucumber, through which we will be able to prepare reports for specific scenarios under the same tag.
* By default, Cucumber executes all the scenarios inside the **feature file**, but if we need to execute or skip any specific scenario under a specific test, so we can declare scenarios within a tag.

We can declare a tag in a feature file by the following syntax:

@TestName

**Scenario:** Mention the Scenario

Where,

**@:** It is a symbol used to declare a tag.

**TestName:** It is the name of a specific test.

**Scenario:** It is a scenario.

Now, if we need to execute a scenario under multiple tests, in this case, we can create a set of multiple tests by using a tag.

### **Syntax:**

**@TestName@TestName**

**Scenario:** Mention the scenario

Let's understand tag through an example:

### **Example:**

Suppose, a feature file of an application contains 100 test scenarios, and when we test this application through Cucumber testing each time 100 test scenarios will get executed unnecessarily. And due to that, system performance is getting low.

To overcome this problem, we can use a tag.

Let's take an instance of a feature file with few scenarios.

**@SmokeTest**

**Scenario:** Search contact

**Given:** Desired contact will be displayed

**@RegressionTest**

**Scenario:** Search a deal

**Given:** Desired deal will be displayed

**@SmokeTest**

**Scenario:** Search an email

**Given:** Desired email will be displayed

There are two benefits by using the tag in the above feature file:

* First, the @SmokeTest or @RegressionTest tag contains only those scenarios that are applicable to the smoke or regression testing.
* Second, scenarios can be included or excluded as per the requirement at the time of execution.

AD

Now suppose, we need to test only those scenarios which are declared under the smoke test, then we can mention **@SmokeTest** tag inside the testing code in the following way:

tags={**"@SmokeTest"**}

After mentioning the tag inside the testing code, only the scenarios which are declared under the smoke test will be tested and remaining will be skipped.

## What's the need for tags in cucumber testing?

It looks easy when we just have a few numbers of scenarios in a feature file. However, in real-time projects, it does not happen.

In real-time projects, there may be a large number of feature files, which may have a **different purpose** such as Smoke test/Regression test, **different status** such as Ready for execution/Work in progress, **different prospective** such as Developer/QA/BA, etc.

In order to manage the execution of such large feature files, we use the tag with scenarios inside the feature file.

The benefit of the tag is that we can only test the specific scenario of a feature file, which we need to execute, and not those scenarios which are unnecessary.

## How to create a set of multiple tags in cucumber testing?

We can also use multiple tags when we need to perform more than one testing of a single scenario.

### **Example:**

**@SmokeTest tag @RegressionTest tag**

**Scenario:** Search contact

**Given:** Desired contact will be displayed

**@RegressionTest tag**

**Scenario:** Search a deal

**Given:** Desired deal will be displayed

**@SmokeTest tag**

**Scenario:** Search an email

**Given:** Desired email will be displayed

As per the above example, we can test the first scenario for both smoke testing and regression testing.

The testing through multiple tags can be done by using two operators:

* **OR operator**
* **AND operator**

### **OR operator**

The OR operator can be used in the case, when we need to test an application like this, if the application has failed in the first test, then the next test should be checked. If the next test is also failed, then another next test should be checked and so on.

In other words, each test case among from the set of multiple tags must be tested whether it is failed or pass.

To use the OR operator in the test executing program, use the following syntax:

Tags= {"@FirstTest, @SecondTest"}

tags= ("@regression or @smoke"),

**Example:**

tags= {"@SmokeTest, @RegressionTest"}

tags= ("@regression or @smoke"),

In the above example, OR operator executes all the tagged tests i.e., **@SmokeTest**, **@RegressionTest**.

### **AND Operator**

The AND operator can be used in the case, when we need to test an application like this, if the application has passed in the first test, then jump to check the next test, but if it gets failed, then testing should be terminated.

To use the AND operator in the test executing program, use the following syntax:

tags= {"@FirstTest", "@SecondTest"}

tags= ("@regression and @smoke"),

**Example:**

tags= {"@SmokeTest", "@RegressionTest"}

tags= ("@regression and @smoke"),

In the above example, AND operator executes the first test, if the result of this test is passed then it will jump to check the next test. But if the result is failed then testing will be terminated.

## How to ignore tags in Cucumber testing?

In the case, when we need to skip a test, then we can use Special symbol "**~**" within the tag. This Special Character also works to skip both Scenarios and Features. And this can also work in conjunction with OR or AND.

### **Example:**

Suppose there is a group of **@SmokeTest** and **@RegressionTest** tests. Now, we need to skip the regression test, to do that, consider the following code:

tags={"@SmokeTest", "~@RegressionTest"}

# Gherkin Language

As we have learned in the cucumber testing, [feature files](https://www.javatpoint.com/feature-file-in-cucumber-testing) are created with the executable test scripts. The language, in which these executable test scripts are written, is known as Gherkin language.

Basically, Gherkin is a plain English text language used to interpret and execute the test scripts.

## When Cucumber supports the English text, why do we need a separate Gherkin language?

It has been discussed several times that Cucumber supports plain English text, then why we need a separate Gherkin language. The answer to this lies in the concept of the [BDD](https://www.javatpoint.com/cucumber-behavior-driven-development) (Behavior Driven Development).

As we know, BDD involves various software behavior development possibilities while preparing test scripts. These can be development-related possibilities or business-related possibilities. In order to accomplish these possibilities, we need members from different communities like testers, developers, product owners, and project managers while developing test scripts.

Since these members do not belong to the same community; therefore, it is hardly possible to use the common language by each of them. Due to that, the concept of the test script is at high risk. In order to reduce this risk, Gherkin was developed.

Gherkin offers a common set of keywords in the plain English text, which can be used by members from different communities and can get the same output from the test scripts.

Gherkin offers the following specific keywords to write the common test scripts in the feature file:

* **Feature**
* **Scenario**
* **Given**
* **When**
* **Then**
* **But**
* **And**
* **Background**



### **Feature**

Each feature file of Cucumber testing starts with a [feature](https://www.javatpoint.com/feature-in-cucumber-testing) keyword. It is a standalone unit or functionality to be tested. For example, login feature, payment transfer feature, registration feature, etc.

**Example:**

**Feature:** Login

**Scenario:** Login verification

**Given** user navigates to the website **javatpoint.com**

**And** user logs in through **Login Window** by using Username as "USER" and Password as "PASSWORD"

**Then** login must be successful.

### **Scenario**

Each feature contains the required number of tests to test the feature. Each test is named as a Scenario.

For example, feature login functionality can contain two scenarios, first for a **successful login** and second for **unsuccessful login**.

**Scenario:** Successful Login with Valid entries

**Given** user navigates to the website **javatpoint.com**

**And** user logs in through **Login Window** by using Username as "USER" and Password as "PASSWORD"

**Then** login must be successful.

**Scenario:** Unsuccessful Login with Invalid entries

**Given** user navigates to the website **javatpoint.com**

**And** user logs in through **Login Window** by using Username as "USER" and Password as "1234erty"

**But** user entered wrong username and password

**Then** login must be unsuccessful.

### **Given**

This keyword refers to the pre-condition of the test. For example, to access any web application, the first requirement or precondition is to navigate its home page, because, from the home page, we can navigate to other links such as **signup**, **login**, etc.

**Example:**

**Feature:** Login

**Scenario:** Login verification

**Given** user navigates to the website **javatpoint.com**

**And** user logs in through **Login Window** by using Username as "USER" and Password as "PASSWORD"

**Then** login must be successful.

### **When**

It usually refers to the actions of a user that is to be executed.

**Example:**

**Scenario:** Successful Login with Valid entries

**Given** user navigates to the website **javatpoint.com**

**When** User Navigate to Login Page

**And** user logs in through **Login Window** by using Username as "USER" and Password as "PASSWORD"

**Then** login must be successful.

**When** user fills "registration email textbox" with "nikita.sharma27@gmail.com".

### **Then**

This keyword refers to the outcome of the previous step or upcoming action.

**Example:**

**Scenario:** Successful Login with Valid entries

**Given** user navigates to the website **javatpoint.com**

**When** User Navigate to Login Page

**And** user logs in through **Login Window** by using Username as "USER" and Password as "PASSWORD"

**Then** login must be successful.

### **But**

This keyword is used to add negative conditions.

**Example:**

**Scenario:** Unsuccessful Login with Invalid entries

**Given** user navigates to the website **javatpoint.com**

**And** user logs in through **Login Window** by using Username as "USER" and Password as "1234erty"

**But** user entered wrong password

**Then** login must be unsuccessful.

### **And**

This keyword is used to add more conditions into your steps.

**Example:**

**Given** User is on Home Page

**And** Login Link is displayed

**When** User Navigates to Login form

**And** User enters email and Password

**Then** Login Successfully will be displayed

**And** Logout Link should be displayed

### **Background**

This keyword is used to define the steps that are common to all tests in the feature file. For example, Navigation to Home Page, Click on the Login, Enter User Name and Password, Click on Submit button are the common steps in almost all web applications.

# What is a feature in cucumber testing?

A feature is a functionality or standalone unit of a software application. In other words, the feature is a parameter which is used to test the requirements of the customer from the software product.

Let's understand it through a very common example of a social networking site.

Few basic features of the social networking site can be determined as -

* Create and delete the user from the social networking site.
* User login functionality to access the social networking site.
* Sharing videos or photos on the social networking site.
* Sending a friend request.
* Logout or sign out.

At the time of testing, it is the best practice that we should determine the features first, before deriving the test scripts to be tested.

Hence from the above discussion, it is clear that, when we talk about **cucumber**, each independent functionality of the product or web application can be called as a **feature**. A feature typically has a list of scenarios to be tested for that feature, and the feature and its description are stored in the [feature file](https://www.javatpoint.com/feature-file-in-cucumber-testing).

There can be numerous features of a software product. Hence for the better management of features, we should create a separate feature file for each feature.

|  |  |  |
| --- | --- | --- |
| **Sr.No** | **Feature** | **Feature Filename** |
| 1 | Login Feature | Login.feature |
| 2 | Share Post Feature | sharePost.feature |
| 3 | Create an Account Feature | AccountCreation.feature |
| 4 | Delete an Account Feature | AccountDelete.feature |

The keyword **"Feature**" represents a feature under the test in Gherkin language.

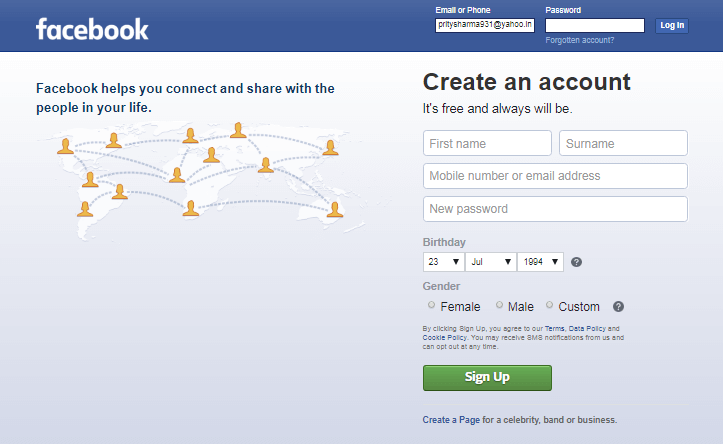
#### Note:**It is recommended that write a small description of features beneath the keyword feature in the feature file.**

### **Example:**

Suppose, the feature login functionality of a social networking site is under the test. Hence, we need to test it as per the following aspects:

* If both credentials, i.e., username and the password are correct, then the user should log in into the social networking site.
* If the username is incorrect, but the password is correct, then the user should be shown an error message.
* If the user name is correct, but the password is incorrect, then the user should be shown an error message.
* After the successful login, the user should be navigated to My Account or Profile page.

AD



Now, we are going to create a feature file for the login feature of the social networking site:

**Feature:** Login functionality

**Scenario:** Successful Login with Valid entries

**Given** user navigates to the website **facebook.com**

**And** user logs in through **Login link** by using username as "**pritysharma321@yahoo.com**" and password as "**prity123sharma**"

**Then** login must be successful.

**Scenario:**  Unsuccessful Login with Invalid entries

**Given** user navigates to the website **facebook.com**

**When** username is incorrect, but the password is correct

user logs in through **Login link** by using Username as "**Parma321@yahoo.com**" and Password as "**prity123sharma**"

**When** username is correct, but the password is incorrect

user logs in through **Login link** by using username as "**pritysharma321@yahoo.com**" and Password as "12345678"

**Then** login must be unsuccessful.

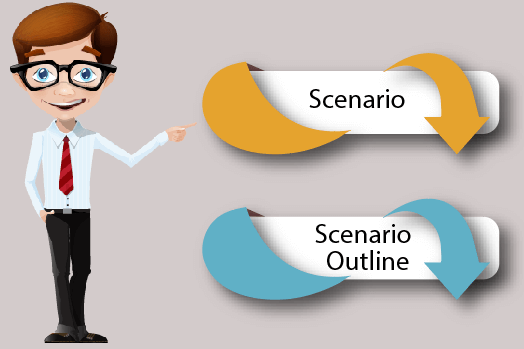
According to the above example, we can create feature files as per the particular feature. A feature file is always based on the behavior of an application under specific circumstances.

# What is Scenario in Cucumber Testing?

The **scenario** is one of the core structures of the Gherkin language. Scenario includes all the possible circumstances of the [feature](https://www.javatpoint.com/feature-in-cucumber-testing) and test scripts for these circumstances.

The keyword "**Scenario**" represents a scenario in [Gherkin language](https://www.javatpoint.com/cucumber-testing-gherkin-language).

One feature can have multiple scenarios, and each scenario consists of one or more steps.



Let's understand the scenario through an example:

### **Example:**

In order to ensure Feedback functionality of a web application, we are creating a cucumber test to verify whether it is working or not.

**Scenario** - Verification of Feedback Functionality

**Given** the user navigates to **JavaTpoint.com**

**When** the user clicks on **Feedback**, then the Feedback page opens

**And** the user submits feedback message

**Then** feedback should be received on the admin page

**And** admin can reply to the user

Each scenario follows the **Given**, **When**, **And**, and **Then format**. This format comes under a particular language, called "**Gherkin language**".

* **Given:** It describes the task of a particular function of an application. It is basically a pre-known state or pre-condition for testing.
* **When:** It is used to describe the action. For example, when the user tries to login, it is considered as an action.
* **Then:** The expected result should be placed here. For example, the verification of a successful login is a result.
* **And:** We can use it to combine two or more actions of the same type. For example, username and password belong to the same type, i.e., login function.

## Scenario Outline

In Gherkin language, scenario outline is the keyword which is used to run the same scenario multiple times.

It is also defined as "Scenario outlines are used when the same test is performed multiple times with a different combination of values."

The keyword scenario outline can also be used by the name **Scenario Template**. In other words, the keyword Scenario Template is a synonym of scenario outline.

Scenario outline is exactly similar to the scenario structure, but the only difference is the **provision of multiple inputs**. In order to use scenario outlines, we do not need any smart idea, we just need to copy the same steps and re-execute the code.

Let's understand the scenario outline through an example:

### **Example:**

Suppose we need to test whether the login functionality for multiple users is working properly or not. This case requires the execution of the login functionality scenario multiple times. Hence, the scenario outline can be used for the testing of login functionality scenario with multiple usernames and passwords.

While mentioning variables such as "Username" and "Password" in gherkin statements, mention them with "**<>**". Consider the following test script:

**Scenario:** Successful Login with Valid entries

**Given** user navigates to the website **javatpoint.com**

**When** User Navigate to Login Page

**And** user logs in through **Login Window** by using **<Username>** as "username1"

**And <Password>** as "password1"

**Then** login must be successful.

Let's take the above test script, in which we need to test login functionality with several different sets of username and password.

Feature: Login Functionality

Scenario Outline: Login functionality

Username | Password

username1 | password1

username2 | password2

username3 | password3

username4 | password4

In the above example, we have provided multiple input values for the variables "Username" and "Password". While executing the actual test, Cucumber will replace the variables with the provided input values.

Once username1 has been executed with password1, the test will execute for the second iteration with another input value. Cucumber will follow this path until the completion of all provided values.

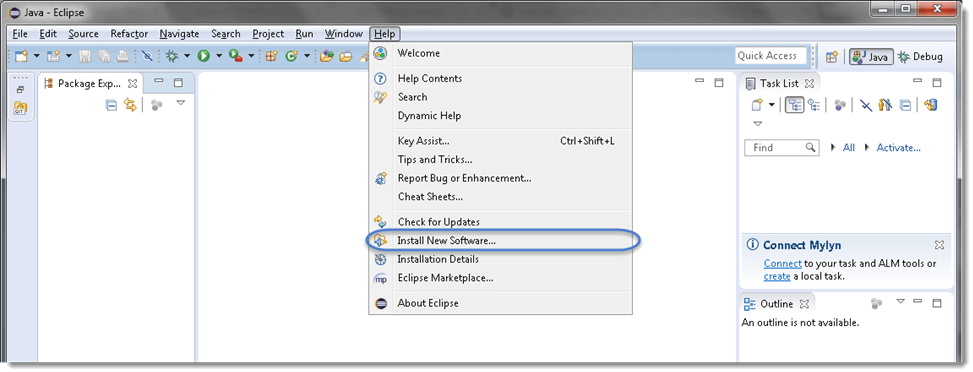
#### Note:**A**Scenario Outline**must contain a**Scenario**section. Scenario steps are interpreted as a template and never executed directly.**

# Install Cucumber Eclipse Plugin

We can use cucumber in eclipse only after plugin. In order to plugin cucumber with eclipse, use the following steps:

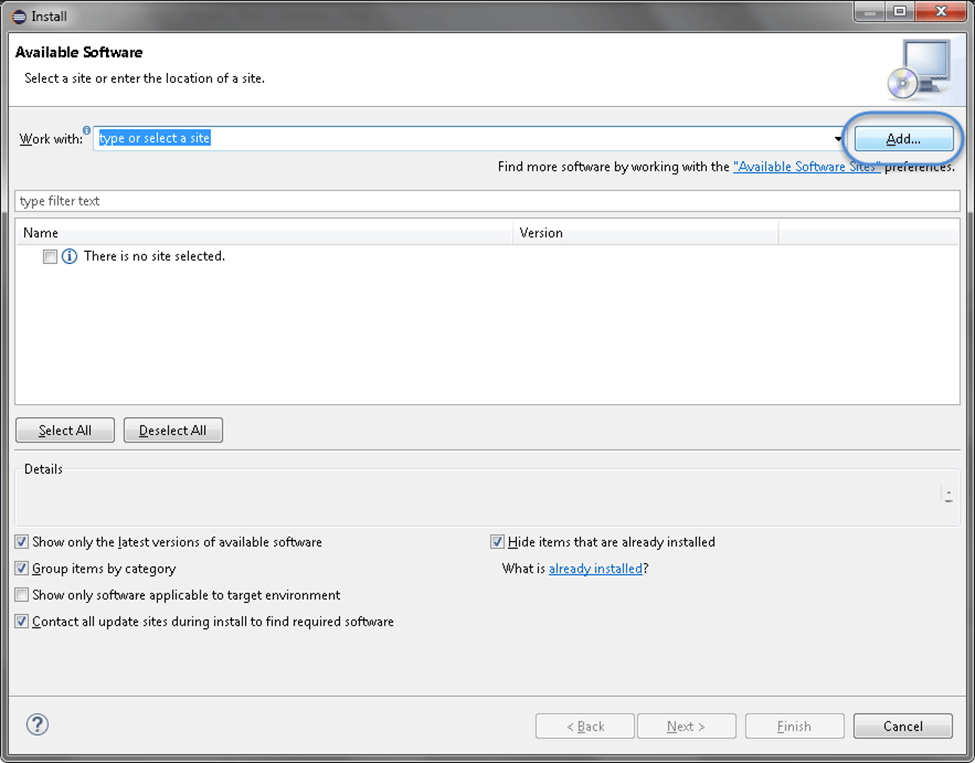
**Step 1**

In a first step, make sure the availability of good Internet connection. Now, launch the **Eclipse IDE** then go to Help menu, and click "**Install New Software**".



**Step 2**

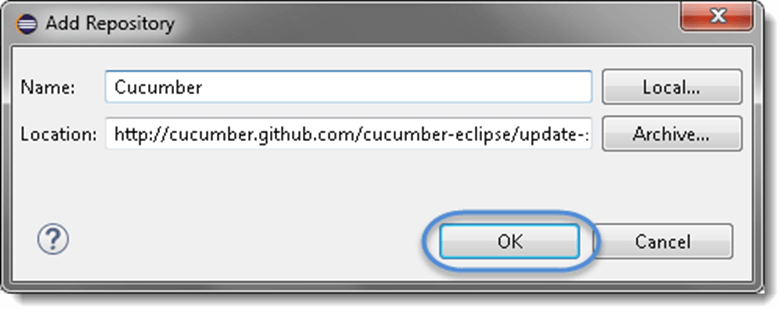
After clicking "**Install New Software**", a window will be prompted, on this window, click the "**Add**" button.



**Step 3**

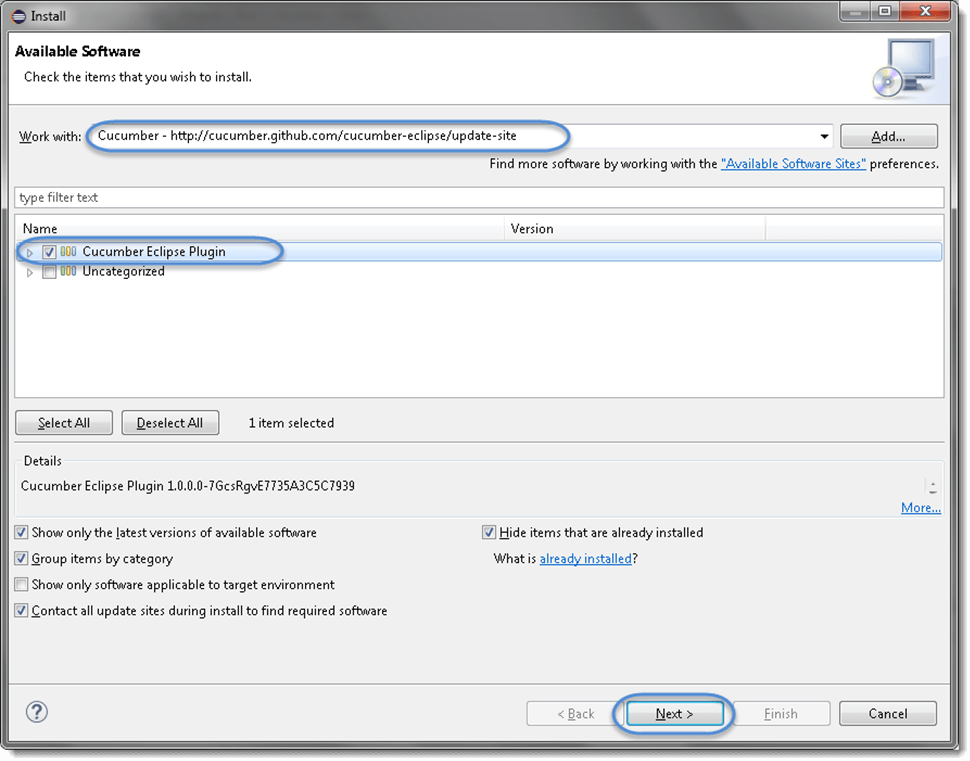
After clicking the "**Add**" button, give the **Name** in the in the text box as per your choice. We provided "**Cucumber**".

Now, in the Location text box type "[**http://cucumber.github.com/cucumber-eclipse/update-site**](https://cucumber.github.com/cucumber-eclipse/update-site)" as locationand then click **OK**.



**Step 4**

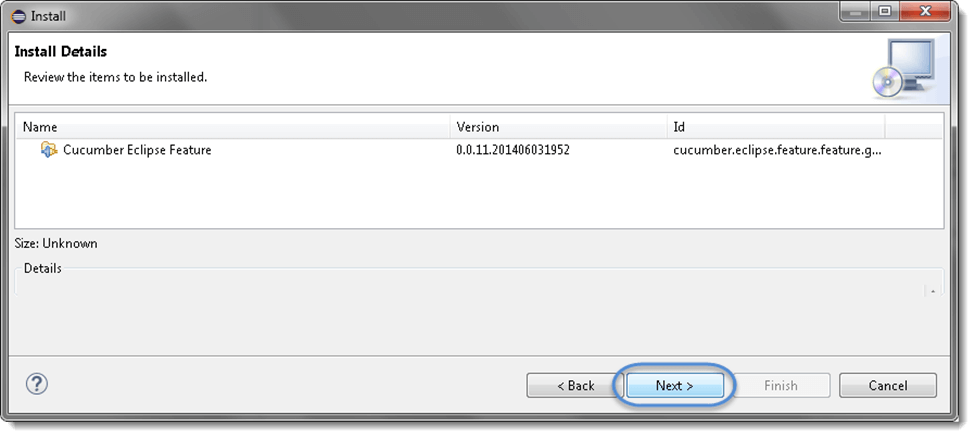
Now, you will come back on the previous window, but this time you will see "**Cucumber Eclipse Plugin**" in the software list. Just click "**Check Box**" and then the "**Next**" button.



#### Note:**If the proxy server is running behind, then you will get an error saying**'HTTP Proxy Authentication Required'**. In this circumstance, you need to contact a system administrator to set up your proxy server settings.**

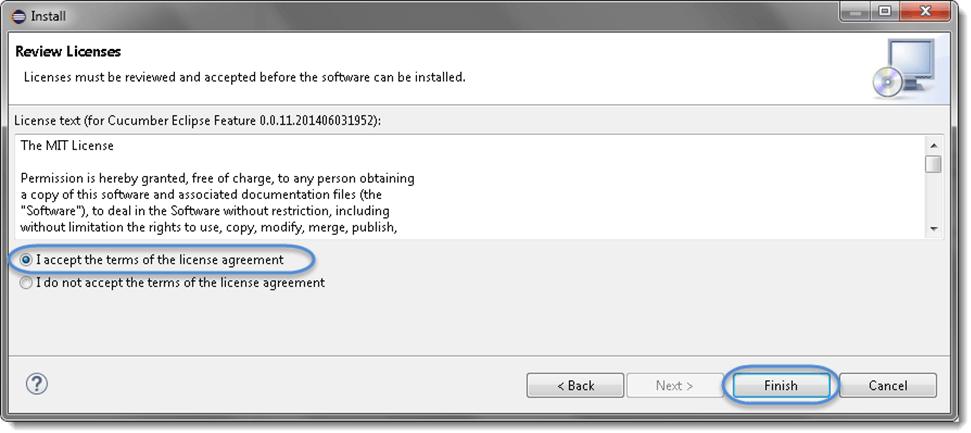
**Step 5**

Now, click on the "**Next**" button.



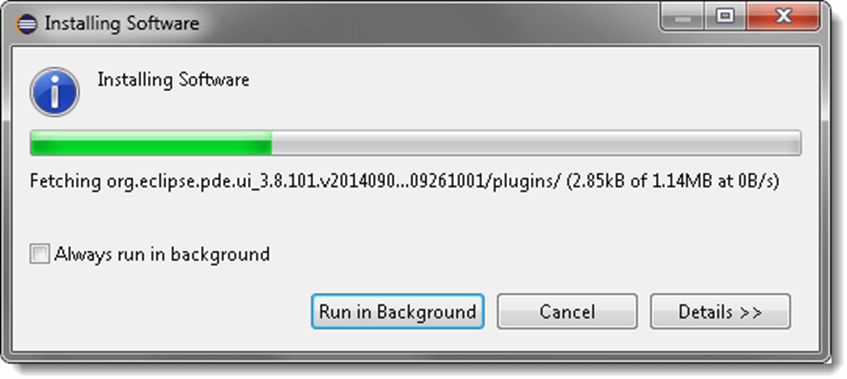
**Step 6**

Click the check box "**I accept the terms of the license agreement**" on the license window then click **Finish**.



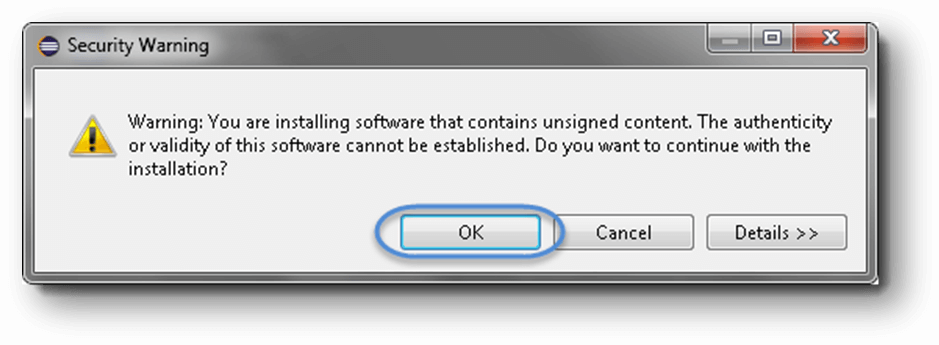
**Step 7**

Now, the installation will be started. It can take some time to be completed.



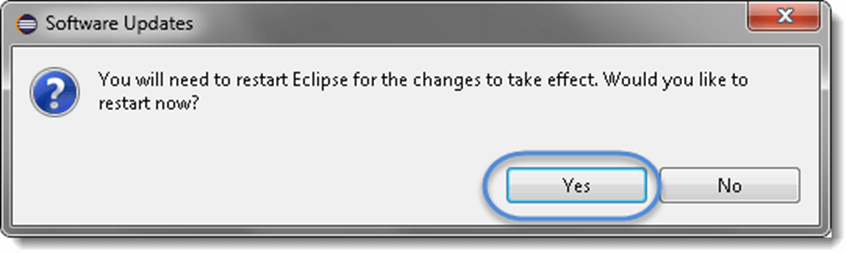
**Step 8**

If you encounter a Security warning, just click **OK**.



**Step 9**

The installation has been completed, now just click "**Yes**" button.



What is the Data Table in Cucumber?

Data tables are used when we need to test numerous input parameters of a web application. For example, the registration form of the new user involves several parameters to test, so for this, we can use the data table.

Let's understand the data table with an instance of a registration form of javatpoint.com, which is a web application. For this registration form, we are going to create a [typical feature file](https://www.javatpoint.com/feature-file-in-cucumber-testing), and later we will create a **feature file with data table** to recognize the difference between them.

The registration form contains the following parameters:

* **User Name**
* **Email**
* **Password**
* **Confirm Password**
* **Birth-date**
* **Gender**
* **Phone Number**

**Feature file to test the feature "User Registration."**

**Feature:** New user registration.

**Scenario:** Verification of successful registration when the inputs are correct.

**Given** user on the user registration page

**When** user enters a valid user name

**And** valid e-mail address

**And** valid password

**And** valid confirmation password

**And** valid Birth-date

**And** valid Gender

**And** valid phone number

**Then**user registration should be successful

In the above feature file, we can see that it looks a bit complex at first glance due to using "**And**" multiple times. So, to reduce such type of complexity, we can use "**Data Table**."

Data table involves a set of input parameters, and these parameters are provided to a single tag such as **GIVEN, WHEN,** or **THEN**.

Let's create the above feature file with the data table, and see how it will look:

**Given** the user on the user registration page.

**When** user enter invalid data on the page

| Fields| Values|

| First Name | User Name |

| Last Name | User Last Name |

| Email Address | someone@gmail.com |

| Re-enter Email Address | someone@gmail.com |

| Password |PASSWORD|

| Birth-date | 02|

**Then** the user registration should be successful.

## Data Tables in Cucumber

In this example, we will pass the test data using the data table and handle it using ***Raw()*** method.

Scenario: Successful Login with Valid Credentials

Given User is on Home Page

When User Navigate to LogIn Page

And User enters Credentials to LogIn

| testuser\_1 | Test@153 |

Then Message displayed Login Successfully

***The implementation of the above step will be like this:***

The implementation of the above step will belike this:

@When("^User enters Credentials to LogIn$")

public void user\_enters\_testuser\_\_and\_Test(DataTable usercredentials) throws Throwable {

//Write the code to handle Data Table

List<List<String>> data = usercredentials.raw();

//This is to get the first data of the set (First Row + First Column)

driver.findElement(By.id("log")).sendKeys(data.get(0).get(0));

//This is to get the first data of the set (First Row + Second Column)

driver.findElement(By.id("pwd")).sendKeys(data.get(0).get(1));

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

}

In the above feature file, we can see that all parameters of the registration form have been organized in a simple and recognizable manner.

How to create Selenium Maven Project in Eclipse?

Before creating Selenium Maven testing Project for cucumber testing in Eclipse, first we must have the following dependencies in our computer system:

* **Java**
* **Eclipse**
* **Cucumber Eclipse Plugin**
* **Web driver Java Client**
* [**Selenium**](https://www.javatpoint.com/selenium-tutorial)

Here we are creating a Maven project to test the registration feature through the data table:

**Step 1**

In order to create a Maven project first, Go to **File → New → Others → Maven → Maven Project → Next**.

After clicking the **Next**, a window will prompt. In this window, we need to provide **group Id** (group Id identifies the project uniquely across all projects). We have provided group Id as **"com.javatpoint."** You can give any name on your own choice.

Also, provide **artifact Id** (artifact Id represents the name of the project). We have provided artifact Id as **"CucumberTesting."** You can give any name on your own choice.

Click on **Finish**.

**Step 2**

Open pom.xml by using the following steps:

* Go to package explorer on the left side of the Eclipse window.
* Expand the project which is created for data table testing in cucumber then select its pom.xml file.
* Now open pom.xml and add the following dependencies.

AD

**Add dependency inside pom.xml for Selenium:** This will indicate to Maven, which Selenium jar files will be downloaded from the central repository to the local repository.

* In pom.xml file, create a dependencies tag **(<dependencies></dependencies>)**, inside the project tag.
* Now, inside the dependencies tag, create a dependency tag **(<dependency></dependency>)**, and provide the following information within it.

1. <dependencies>
2. <dependency>
3. <groupId> org.seleniumhq.selenium </groupId>
4. <artifactId> selenium-java </artifactId>
5. <version> 2.47.1 </version>
6. </dependency>
7. </dependencies>

**Step 3**

**Add dependency inside pom.xml for Cucumber-Java:** It will indicate to Maven; which Cucumber files will be downloaded from the central repository to the local repository.

* Now, inside the dependencies tag **(<dependencies></dependencies>)**, create a dependency tag **(<dependency></dependency>)**, and provide the following information within it.

1. <dependencies>
2. <dependency>
3. <groupId> info.cukes </groupId>
4. <artifactId> cucumber-java </artifactId>
5. <version> 1.0.2 </version>
6. <scope> test </scope>
7. </dependency>
8. </dependencies>

**Step 4**

**Add dependency for Cucumber-Junit:** It will indicate to Maven, which Cucumber JUnit files will be downloaded from the central repository to the local repository.

* Now, inside the dependencies tag **(<dependencies></dependencies>)**, create a dependency tag **(<dependency></dependency>)**, and provide the following information within it.

1. <dependencies>
2. <dependency>
3. <groupId> info.cukes </groupId>
4. <artifactId> cucumber-junit </artifactId>
5. <version> 1.0.2 </version>
6. <scope> test </scope>
7. </dependency>
8. </dependencies>

**Step 5**

**Add dependency for Junit:** It will indicate to Maven, which JUnit files will be downloaded from the central repository to the local repository.

* Now, inside the dependencies tag **(<dependencies></dependencies>)**, create a dependency tag **(<dependency></dependency>)**, and provide the following information within it.

1. <dependencies>
2. <dependency>
3. <groupId> junit </groupId>
4. <artifactId> junit </artifactId>
5. <version> 4.10 </version>
6. <scope> test </scope> </dependency>
7. </dependencies>

After completing all dependencies, verify binaries.

* Once pom.xml is completed successfully, then save it.
* Go to your Project → Clean - It can take a few minutes.

Now, create a package named **dataTable** under **src/test/java** folder of your project.

**Step 6**

Create a Feature file:

* Inside the package dataTable, create a feature file, named **dataTable.**feature.
* Inside the feature file, write the following text.

**Feature** - Data table

Verify that the new user registration is successful after passing correct inputs.

**Scenario:**

**Given** the user on the user registration page.

**When** user enter invalid data on the page

| Fields | Values |

| First Name | Preeti |

| Last Name | Sharma |

| Email Address | someone@gmail.com |

| Re-enter Email Address | someone@gmail.com |

| Password |PASSWORD |

| Birthdate | 02 |

**Then** the user registration should be successful.

* Save this file.

**Step 7**

Creation of the step definition file:

* Create the step definition file inside the package dataTable with extension ".java" and named as 'dataTable.java.'
* Inside the step definition file, write the following code.

1. **package** dataTable;
2. **import** java.util.List;
3. **import** org.openqa.selenium.By;
4. **import** org.openqa.selenium.WebDriver;
5. **import** org.openqa.selenium.WebElement;
6. **import** org.openqa.selenium.firefox.FirefoxDriver;
7. **import** org.openqa.selenium.support.ui.Select;
8. **import** cucumber.annotation.en.Given;
9. **import** cucumber.annotation.en.Then;
10. **import** cucumber.annotation.en.When;
11. **import** cucumber.table.DataTable;
12. **public** **class** StepDefinition {
13. WebDriver driver = **null**;
14. @Given("^I am on user registration page$")
15. **public** **void** goToFacebook() {
16. //Intiate web browser instance. driver = new FirefoxDriver();
17. driver.navigate().to("https://www.javaTpoint.com/");
18. }
20. @When("^I enter valid data on the page$")
21. **public** **void** enterData(DataTable table){
22. //Initialize data table
23. List<list> data = table.raw();
24. System.out.println(data.get(1).get(1));
26. //Enter data
27. driver.findElement(By.name("firstname")).sendKeys(data.get(1).get(1));
28. driver.findElement(By.name("lastname")).sendKeys(data.get(2).get(1));
29. driver.findElement(By.name("registered\_email\_\_")).sendKeys(data.get(3).get(1));
30. driver.findElement(By.name("registered\_email\_confirmation\_\_")).
31. sendKeys(data.get(4).get(1));
32. driver.findElement(By.name("registered\_passwd\_\_")).sendKeys(data.get(5).get(1));
34. Select dropdownB = **new** Select(driver.findElement(By.name("birth\_day")));
35. dropdownB.selectByValue("12");
37. Select dropdownM = **new** Select(driver.findElement(By.name("birth\_month")));
38. dropdownM.selectByValue("7");
40. Select dropdownY = **new** Select(driver.findElement(By.name("birth\_year")));
41. dropdownY.selectByValue("1992");
43. driver.findElement(By.className("\_59mt")).click();
44. // Click submit button driver.findElement(By.name("websubmit")).click();
45. }
47. @Then("^User registration should be successful$")
48. **public** **void** User\_registration\_should\_be\_successful() {
49. **if**(driver.getCurrentUrl().equalsIgnoreCase("https://www.javaTpoint.com/")){
50. System.out.println("Test Pass");
51. } **else** {
52. System.out.println("Test Failed");
53. }
54. driver.close();
55. }
56. }

**Step 8**

After creating the step definition file now, we need to create a runner class file.

* Create a runner class inside the package dataTable with extension "**.java**" and named as RunTest.java.
* Inside runner class **RunTest.java**, write the following code.

1. **package** dataTable;
2. **import** org.junit.runner.RunWith;
3. **import** cucumber.junit.Cucumber;
4. @RunWith(Cucumber.**class**)
5. @Cucumber.Options(format = {"pretty", "html:target/cucumber"})
6. **public** **class** RunTest { }

Save this file, and run the test by using the following options:

* Select the runner class i.e., RunTest.java file inside your package.
* Right-click on it, and select the option, **Run as → JUnit**.

If your execution is successful, you will observe the following things:

* **JavaTpoint** website gets loaded.
* We will see the home page or the page provided by the respective website.
* Data can be entered on the registration page.
* Submit button will be clicked.

# What is Hook in Cucumber?

In Cucumber, the **hook is the block of code** which can be defined with each scenario in step definition file by using the annotation **@Before** and **@After**. These @Before and @After annotations create a block in which we can write the code.



Cucumber **hook** facilitates us to handle the code workflow better and also helps us to reduce code redundancy.

**Syntax:**

1. @Before setup ()
2. {
3. logic
5. } @
7. Scenario
8. Given
9. When
10. And
11. Then
13. @After cleanup (){
14. logic
15. }

As per the code logic, hook's job is to start and close the web driver session after a specific function/method. Hence, in actual, it is not relevant to any function/method or scenario.

#### **Note: Hooks can be defined only in the step definition file.**

## The Need of Hook

At the time of testing, we may encounter circumstances where we need to perform some conventional prerequisite steps before the testing of the test scenario.

Consider the following prerequisite to understand the kind of prerequisites which may encounter at the time of testing:

* To Start a web driver
* Set up of Data Base connections
* Set up of test data
* Set up of browser cookies
* Navigation to a certain page

Similarly, there are always some prerequisite steps which may encounter after testing:

* To stop the web driver
* To Close DB connections
* To Clear the test data
* To Clear browser cookies
* To Log out from the application
* Printing reports or logs
* Taking the screenshots of error

AD

In order to handle these types of conventional prerequisite steps, using cucumber hook is the best option.

## Hook Annotations

Unlike TestNG Annotations, the cucumber supports only two hooks:

* **@Before**
* **@After**

**@Before**

As the name suggests, we can use the **@Before** hook with the function/method after which we need to start web driver.

**@After**

As the name suggests, we can use the **@After** hook with the function/method after which we need to close the web driver.

Let's understand this notion better with an example of a step definition file.

### **Example:**

Here is an instance of a step definition file of a **Maven testing project**. This project is created for the testing of web application javaTpoint.

In order to use the hook, we created the step definition file named **hookTest.java** under the package **javatpointTest**.

1. **package** javatpointTest;
3. **import** org.openqa.selenium.By;
4. **import** org.openqa.selenium.WebDriver;
5. **import** org.openqa.selenium.firefox.FirefoxDriver;
7. **import** cucumber.annotation.en.Given;
8. **import** cucumber.annotation.en.Then;
9. **import** cucumber.annotation.en.When;
11. **public** **class** hookDemo {
12. WebDriver driver = **null**;
14. @Before **public** **void** setUp(){
15. driver = **new** FirefoxDriver();
16. }
18. @Given("^User navigates to javatpoint$")
19. **public** **void** goToFacebook() {
20. driver.navigate().to("https://www.javatpoint.com/");
21. }
23. @When("^ user enter Username as \"([^\"]\*)\" and Password as \"([^\"]\*)\"$" ")
24. **public** **void** User\_enter\_Username\_and\_Password(String arg1, String arg2) {
25. driver.findElement(By.id("emailAddress")).sendKeys(arg1);
26. driver.findElement(By.id("password")).sendKeys(arg2);
27. driver.findElement(By.id("u\_0\_v")).click();
28. }
30. @Then("^login should be unsuccessful$")
31. **public** **void** validateRelogin() {
32. **if**(driver.getCurrentUrl().equalsIgnoreCase(
33. "https://www.javatpoint.com/login.php?login\_attempt=1&lwv=110")){
34. System.out.println("Test Pass");
35. } **else** {
36. System.out.println("Test Failed");
37. }
38. driver.close();
39. }
41. @After **public** **void** cleanUp(){
42. driver.close();
43. }
44. }

When we execute this code, the following will be the sequence of execution:

* At the beginning, **@Before** annotation will set up the web driver and other required prerequisites to execute the test.
* After setting up web driver and other prerequisites, the **Given** statement will be executed.
* After the execution of the **Given** statement, the **When** statement will be executed.
* After the execution of the **When** statement, the **Then** statement will be executed.
* Now at the last, **@After** hook will **close** the web driver and do the cleanup process.

Since we know that, to execute step definition file, we should have a complete **Maven testing project** so first create it in eclipse.

## Tagged Hooks

The hook can also we used with tag. We can use **@before** and **@after** hooks with a specific test.

**Example:**

1. @Before ('@RegressionTest)
3. @After ('@RegressionTest)

We can also use the same concept of the hook with logical and/or operator.

**Example:**

1. @Before ('@RegressionTest, @SmokeTest)
3. @ After ('@RegressionTest, @SmokeTest)

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

[Software Testing Help](https://www.softwaretestinghelp.com/)

<https://www.softwaretestinghelp.com/cucumber-interview-questions/>

**Introduction to Cucumber with most frequently asked Cucumber Interview questions:**

Cucumber is a tool that is based on Behavior Driven Development (BDD) framework.

BDD is a methodology to understand the functionality of an application in the simple plain text representation.

### Most Frequently Asked Cucumber Interview Questions

**Q #1) Explain Cucumber shortly.**

**Answer:** Cucumber is a tool that is based on Behavior Driven Development (BDD) methodology.

The main aim of the Behavior Driven Development framework is to make various project roles such as Business Analysts, [Quality Assurance](https://en.wikipedia.org/wiki/Quality_assurance), Developers, etc., understand the application without diving deep into the technical aspects.

**Q #2) What language is used by Cucumber?**

**Answer:**[Gherkin](https://github.com/cucumber/cucumber/wiki/Gherkin) is the language that is used by the Cucumber tool. It is a simple English representation of the application behavior. Gherkin language uses several keywords to describe the behavior of applications such as Feature, Scenario, Scenario Outline, Given, When, Then, etc.

**Q #3) What is meant by a feature file?**

**Answer:** A feature file must provide a high-level description of an Application Under Test (AUT). The first line of the feature file must start with the keyword ‘Feature’ followed by the description of the application under test.

A feature file may include multiple scenarios within the same file. A feature file has the extension .feature.

**Q #4) What are the various keywords that are used in Cucumber for writing a scenario?**

**Answer:** **Mentioned below are the keywords that are used for writing a scenario:**

* Given
* When
* Then
* And

**Q #5) What is the purpose of a Scenario Outline in Cucumber?**

**Answer:**Scenario outline is a way of parameterization of scenarios. This is ideally used when the same scenario needs to be executed for multiple sets of data, however, the test steps remain the same. Scenario Outline must be followed by the keyword ‘Examples’, which specify the set of values for each parameter.

**Q #6) What programming language is used by Cucumber?**

**Answer:** Cucumber tool provides support for multiple programming languages such as Java, .Net, Ruby etc. It can also be integrated with multiple tools such as Selenium, Capybara, etc.

**Q #7) What is the purpose of the Step Definition file in Cucumber?**

**Answer:** A step definition file in Cucumber is used to segregate the feature files from the underlying code. Each step of the feature file can be mapped to a corresponding method on the Step Definition file.

While feature files are written in an easily understandable language like, Gherkin, Step Definition files are written in programming languages such as Java, .Net, Ruby, etc.

**Q #8) What are the major advantages of the Cucumber framework?**

**Answer:** Given below are the advantages of the Cucumber Gherkin framework that make Cucumber an ideal choice for rapidly evolving [Agile methodology](https://www.softwaretestinghelp.com/agile-scrum-methodology-for-development-and-testing/) in today’s corporate world.

* Cucumber is an open-source tool.
* Plain Text representation makes it easier for non-technical users to understand the scenarios.
* It bridges the communication gap between various project stakeholders such as Business Analysts, Developers, and Quality Assurance personnel.
* Automation test cases developed using the Cucumber tool are easier to maintain and understand as well.
* Easy to integrate with other tools such as [Selenium](https://www.softwaretestinghelp.com/selenium-tutorial-1/) and Capybara.

**Q #9) Provide an example of a feature file using the Cucumber framework.**

**Answer:** Following is an **example** of a feature file for the scenario ‘Login into the application’:

**Feature:** Login to the application under test.  
**Scenario:** Login to the application.

* Open the Chrome browser and launch the application.
* When the user enters the username onto the UserName field.
* And User enters the password into the Password field.
* When the user clicks on the Login button.
* Then validate if the user login is successful.

**Q #10) Provide an example of a Scenario Outline using the Cucumber framework.**

**Answer:** The following is an **example** of a Scenario Outline keyword for the scenario ‘Upload a file’. The number of parameter values to be included in the feature file is based on the tester’s choice.

**Scenario Outline:** Upload a file

Given that the user is on upload file screen.  
When a user clicks on the Browse button.  
And user enters <filename> onto the upload textbox.  
And user clicks on the enter button.  
Then verify that the file upload is successful.

**Example:**

|filename|  
|file1|  
|file2|

**Q #11) What is the purpose of the Behaviour Driven Development (BDD) methodology in the real world?**

**Answer:** BDD is a methodology to understand the functionality of an application in the simple plain text representation.

The main aim of the Behavior Driven Development framework is to make various project roles such as Business Analysts, Quality Assurance, Developers, Support Teams understand the application without diving deep into the technical aspects.

**Q #12) What is the limit for the maximum number of scenarios that can be included in the feature file?**

**Answer:** A feature file can contain a maximum of 10 scenarios, but the number can vary from project to project and from one organization to another. But it is generally advisable to limit the number of scenarios included in the feature file.

**Q #13) What is the use of Background keyword in Cucumber?**

**Answer:** Background keyword is used to group multiple given statements into a single group. This is generally used when the same set of given statements are repeated in each scenario of the feature file.

**Q #14) What symbol is used for parameterization in Cucumber?**

**Answer:** Pipe symbol (|) is used to specify one or more parameter values in a feature file.

**Q #15) What is the purpose of Examples keyword in Cucumber?**

**Ans:** Examples keyword is used to specify values for each parameter used in the scenario. Scenario Outline keyword must always be followed by the keyword Examples.

**Q #16) What is the file extension for a feature file?**

**Answer:** File Extension for a feature file is .feature. A feature file is ideally written in a notepad file and is saved with the extension feature.

**Q #17) Provide an example of a step definition file in Cucumber.**

**Answer:** Step definition corresponding to the step “Open Chrome browser and launch the application” may look like the code mentioned below:

@Given("^Open Chrome browser and launch the application$")

public void openBrowser()

{

driver = new ChromeDriver();

driver.manage().window().maximize();

driver.get("www.facebook.com");

}

**Q #18) What is the purpose of the Cucumber Options tag?**

**Answer:** Cucumber Options tag is used to provide a link between the feature files and step definition files. Each step of the feature file is mapped to a corresponding method on the step definition file.

**Below is the syntax of Cucumber Options tag:**

@CucumberOptions(features="Features",glue={"StepDefinition"})

**Q #19) How can Cucumber be integrated with Selenium WebDriver?**

**Answer:** [Cucumber can be integrated with the Selenium Webdriver](https://www.softwaretestinghelp.com/selenium-webdriver-cucumber-selenium-tutorial-31/) by downloading the necessary JAR files.

**Given below are the list of JAR files that are to be downloaded for using Cucumber with Selenium web driver:**

* cucumber-core-1.2.2.jar
* cucumber-java-1.2.2.jar
* cucumber-junit-1.2.2.jar
* cucumber-jvm-deps-1.0.3.jar
* cucumber-reporting-0.1.0.jar
* gherkin-2.12.2.jar

**Q #20) When is Cucumber used in real-time?**

**Answer:** Cucumber tool is generally used in real-time to write acceptance tests for an application. It is generally used by non-technical people such as Business Analysts, Functional Testers, etc.

**Q #21) Provide an example of Background keyword in Cucumber.**

**Answer:**

**Background:** Given the user is on the application login page.

**Q #22) What is the use of Behavior Driven Development in Agile methodology?**

**Answer:** The advantages of Behavior Driven Development are best realized when non-technical users such as Business Analysts use BDD to draft requirements and provide the same to the developers for implementation.

In Agile methodology, user stories can be written in the format of feature file and the same can be taken up for implementation by the developers.

**Q #23) Explain the purpose of keywords that are used for writing a scenario in Cucumber.**

**Answer:**

* **“Given”** keyword is used to specify a precondition for the scenario.
* **“When”** keyword is used to specify an operation to be performed.
* **“Then”** keyword is used to specify the expected result of a performed action.
* **“And”** keyword is used to join one or more statements together into a single statement.

**Q #24) What is the name of the plugin that is used to integrate Eclipse with Cucumber?**

**Answer:**Cucumber Natural Plugin is the plugin that is used to integrate Eclipse with Cucumber.

**Q #25) What is the meaning of the TestRunner class in Cucumber?**

**Answer:** TestRunner class is used to provide the link between the feature file and the step definition file. The next question provides a sample representation of how the TestRunner class will look like. A TestRunner class is generally an empty class with no class definition.

**Q #26) Provide an example of the TestRunner class in Cucumber.**

**Answer:**

Package com.sample.TestRunner

importorg.junit.runner.RunWith;

importcucumber.api.CucumberOptions;

importcucumber.api.junit.Cucumber;

@RunWith(Cucumber.class)

@CucumberOptions(features="Features",glue={"StepDefinition"})

public class Runner

{

}

**Q #27) What is the starting point of execution for feature files?**

**Answer:** When integrated with Selenium, the starting point of execution must be from the TestRunner class.

**Q #28) Should any code be written within the TestRunner class?**

**Answer:**No code should be written under the TestRunner class. It should include the tags @RunWith and @CucumberOptions.

**Q #29) What is the use of features property under the Cucumber Options tag?**

**Answer:**Features property is used to let the Cucumber framework identify the location of the feature files.

**Q #30) What is the use of glue property under the Cucumber Options tag?**

**Answer:** Glue property is used to let the Cucumber framework identify the location of step definition files.

**Q #31) What is the maximum number of steps that are to be written within a scenario?**

**Answer:** 3-4 steps.

**Recommended Reading:**[**Automation testing with Cucumber and Selenium**](https://www.softwaretestinghelp.com/cucumber-bdd-tool-selenium-tutorial-30/)

### Conclusion

* BDD is a methodology to understand the functionality of an application in the simple plain text representation.
* Cucumber is a tool that uses Behaviour Driven Development to write acceptance tests of an application. It is used to bridge the communication gap between various project stakeholders.
* The main use of Cucumber lies in its simplicity to understand and usage of feature files by non-technical users.

**Guru99**

# Top 20 Cucumber Interview Questions & Answers (2022 Update)

**1) What is Cucumber? What are the advantages of Cucumber?**

To run functional tests written in a plain text Cucumber tool is used. It is written in a Ruby programming language.

**Advantages of Cucumber are:**

* You can involve business stakeholders who cannot code
* End-user experience is a priority
* High code reuse

**2) What are the two files required to execute a Cucumber test scenario?**

Two files required to execute a Cucumber test scenario are

* Features
* Step Definition

**3) explain the use of Background keyword in Cucumber?**

Background keyword is used to group multiple given statements into a single group. The keyword mostly used when the same set of given statements are repeated in each scenario of the feature file.

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

* **Feature:**Visit**XYZ** page in abc.com
* **Scenario:** Visit abc.com
* **Given:** I am on abc.com
* **When:**I click on XYZ page
* **Then:**I should see ABC page

**5) What is Scenario Outline in feature file?**

**Scenario Outline is**the same scenario can be executed for multiple sets of data using the scenario outline. The data is provided by a tabular structure separated by (I I).

**6) Explain the term step definition in Cucumber**

A step definition is the actual code implementation of the feature mentioned in the feature file.

**7) Give an example for step definition using “Given” function?**

For example to make visitor visit the site “Yahoo” the command we use for given

Given (/^ I am on www.yahoo.com$/) do

Browser.goto “http://www.yahoo.com.”

end – This will visit [www.yahoo.com](http://www.yahoo.com)

**8) What are the differences between Jbehave and Cucumber?**

Although Cucumber and Jbehave are meant for the same purpose, acceptance tests are completely different frameworks

* Jbehave is, and Cucumber is Ruby-based
* Jbehave are based on stories while Cucumber is based on features

**9) Explain test harness**

A test harness for Cucumber and rspec allows for separating responsibility between setting up the context and interacting with the browser and cleaning up the step definition files

**10) When to use Rspec and when to use Cucumber?**

* Rspec is used for Unit Testing
* Cucumber is used for Behavior-driven development. Cucumber can be used for System and Integration Tests

**11) What is the language used for expressing scenario in feature file?**

Gherkin language is used to express scenario in feature files and ruby files containing unobtrusive automation testing for the steps in scenarios

**12) What are regular expressions?**

A regular expression is a pattern describing a certain amount of text. The most basic regular expression consists of a single literal character.

**13) What is BDD?**

BDD or Behavior-driven development is a process of developing software based on TDD (Test Driven Development) which focusses on the behavioral specification of software testing units.

**14) What software do you need to run a Cucumber Web Test cases?**

* Ruby and its Development Kit
* Cucumber
* IDE like ActiveState
* Watir ( To simulate browser)
* Ansicon and rspec (if required)

**15) What does a cucumber features/ support file contain?**

Features/ support file contains supporting ruby code. Files in support load before those in step\_definitions, which can be useful for environment configuration.

**16) What is a Feature file?**

Features file contain a high-level description of the ~~Test Scenario~~ application under test in simple language. It is known as Gherkin which is a plain English text language. Feature File consists of the following components like:

* Feature: It describes the current test script which has to be executed.
* Scenario: It is steps and expected outcome for a specific test case.
* Scenario outline: Scenario can be executed for multiple sets of data using scenario outline.
* Given: It specifies the context of the text to be executed.
* When: specifies the test action which has to perform.
* Then: Expected outcome of the test can be represented by “Then”

**17) What is Selenium?**

Selenium is an automation tool which is a widely used tool for Functional Testing of the web-based application. Selenium supports different language like ruby, java, python C#, etc.

**18) Why use Cucumber with Selenium?**

Cucumber and Selenium are two popular technologies. Many organizations use Selenium for functional testing. These organizations which are using Selenium want to integrate Cucumber with Selenium as Cucumber helps you to read and to understand the application flow.

**19) Advantages of Cucumber**

Here, are some prominent advantages of using Cucumber.

* It is helpful to involve business stakeholders who can’t easily read the code
* Cucumber Testing enhances the end-user experience
* Style of writing tests allow for easier reuse of code in the tests
* Allows quick and easy setup and execution

**20) What is a Step Definition?**

Step definition maps the Test Case Steps in the feature files to code. It executes the steps on Application Under Test and checks the outcomes against expected results. In order to execute step definition it must match the given component in a feature.

[**Free PDF Download: Cucumber Interview Questions & Answers**](https://drive.google.com/uc?export=download&id=1Dc3NZLwZslcautbqphSoliEFOLQdBH81)

### You Might Like:

* [**Cucumber Framework: What is Cucumber Testing Tool?**](https://www.guru99.com/introduction-to-cucumber.html)
* [**How to Download & Install CUCUMBER in Windows**](https://www.guru99.com/cucumber-installation.html)
* [**What is Cucumber Feature File & Step Definition? (Example)**](https://www.guru99.com/cucumber-basics.html)
* [**Cucumber Testing Tutorials for Beginners**](https://www.guru99.com/cucumber-tutorials.html)
* [**Gherkin Language: Format, Syntax & Gherkin Test in Cucumber**](https://www.guru99.com/gherkin-test-cucumber.html)

**Software Testing material**

<https://www.softwaretestingmaterial.com/cucumber-interview-questions/>

## ****Most Frequently Asked Cucumber Interview Questions & Answers****

### **1) What are the files needed in a Cucumber framework?**

Answer: Cucumber is a tool used to run automated acceptance tests created in a BDD format. One of its most outstanding features of the tool is the ability to carry out plain-text functional descriptions (written in the language called Gherkin) as automated tests.

**Advantages of Cucumber are**

* Its focus on end-user experience
* Cucumber tests are written in plain-text English so people with no or less technical skills can also write scenarios
* It allows us to involve business stakeholders who can’t easily read a code
* Allows quick and easy setup and execution
* High reusability of code in the tests
* It can be integrated with Selenium and other testing frameworks like JUnit & TestNG

### **2) What are the files needed in a Cucumber framework?**

Answer: The files needed in a Cucumber framework are listed below:

**Feature file –** The feature file has an extension of .feature. It has single or multiple test scenarios described in plain text. The tests are written with keywords like Then, When, Background, Scenario Outline, Feature, And, But, and so on. Thus it is a file that maintains the features and their descriptions.

**Step Definition file –** This file has an extension of .java. It basically provides the mapping of the test scenario steps described in the feature file to the automation code. Thus when Cucumber runs a step described in the feature file, it searches the step definition file and executes the relevant functions that are mapped to that step.

**TestRunner file –** This file has an extension of .java. It links the step definition file and the feature file. It gives the user the option to execute one or more than one feature file. It has the path of the step definition file and the feature file.

### **3) What are the different keywords used in feature file?**

Answer: The different keywords used in the feature file are

* Feature
* Background
* Scenario
* Scenario Outline
* Given
* When
* Then
* And
* But

### **4) Explain Scenario Outline in the Cucumber framework.**

Answer: A Scenario Outline is used to run a particular scenario with more than one data set in multiple combinations. A feature file can have fewer lines if we take the help of a Scenario Outline. The data to be passed at runtime need not be hardcoded in the feature file if we use Scenario Outline.

With Scenario Outline the runtime values are passed as parameters enclosed in <> in the feature file. The data set is expressed in a tabular format separated with pipe delimiter starting with the Examples keyword.

Feature file with Scenario Outline implementation.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | Feature: Admin Login Verification  Scenario Outline: Multiple Users Login Test  Given User navigates to Admin Login Page  When Page Title is Welcome to Admin  Then User inputs "<Name>" and "<EmployeeId>"  Examples:           | Name         | EmployeeId |           | Adam | 3678 |           | Sam          | 2589       | |

### **5) Explain keyword Background in the Cucumber framework.**

Answer: The keyword Background is used to describe a test step or a group of test steps that are common to every test inside the feature file. A test having a Background executes before all the test scenarios in the feature file.

Thus a keyword Background is used to run a scenario as a precondition for other scenarios in the feature file.

Feature file with Background implementation

|  |  |
| --- | --- |
|  | Feature: Admin Login Verification  Background: User Login Test  Given User navigates to Admin Login Page  When Page Title is Welcome to Admin  Scenario: New user addition by admin user  Given User navigates to New Member Page  Then Add New Member button should be visible  Scenario:  User deletion by admin user  Given User navigates to Member Delete Page  Then Delete Member button should be visible |

In the above example, we are testing the Admin Login Verification feature. We have two scenarios – New user addition by admin user and User deletion by an admin user. The precondition to both these scenarios is defined with Background scenario – User Login Test and it shall execute two times before each of the scenarios.

### **6) Explain keyword Examples in the Cucumber framework.**

Answer: We can achieve a data-driven approach in Cucumber with the help of the Examples keyword. The Scenario Outline in a feature file should be accompanied by the Examples part which consists of the multiple data set to be passed at the runtime.

The Examples section in the feature file should have headings that match with the variables in the Scenario Outline followed by (|) symbol. Each row below the heading represents a set of data. So if there are two rows after heading, the test scenario shall run two times with the two different sets of data.

Feature file with Examples implementation

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | Feature: New User Registration  Scenario Outline: Registration Verification Test  Given User navigates to Registration Page  Then User inputs "<Firstname>" and "<Lastname>" and "<Email>"  Examples:          | Firstname | Lastname | Email        |          | Rob            | Smith        | abc@gmail.com|          | Deu            | Alfred       | xyz@gmail.com| |

### **7) Explain BDD.**

Answer: BDD means Behavior Driven Development. In Behavior Driven Development, the tests are designed in plain English and are created from a more end-user perspective and describe the application characteristics.

Behavior Driven Development framework increases collaboration and coordination among members in an agile team of developers, testers, business analysts, product owners, customers, and other stakeholders. It does not require any technical knowledge and is written in plain text and requires no technical clarity.

Behavior Driven Development framework like Cucumber helps to bridge any understanding gap between business stakeholders and the developers. Some of the benefits of using Behavior Driven Development are listed below:

* Since all the team members and stakeholders contribute to the project development, it ensures maximum testing coverage.
* Since the tests are designed in plain text, there is proper understanding and knowledge over the business requirements.
* Test tools like Cucumber and SpecFlow in Behavior Driven Development help to automate scenarios.
* Behavior Driven Development helps to achieve reusability of code to a large extent.
* Data parameterization can be easily implemented in Behavior Driven Development.
* Behavior Driven Development allows easy integration with continuous integration tools like Jenkins.

### **8) Describe a feature file with an example.**

Answer: A feature file is the starting of tests in Cucumber. All the test scenarios are elaborated here in plain simple text. It may contain single or multiple test scenarios at a time. It is recommended to create a separate feature file for each standalone feature.

If we are using an **Eclipse IDE**, a feature file can be created by selecting our project, then right-click and select the **New** option. Next click on **File**. Then we have to provide the name of the file with **.feature** as an extension.

A feature file normally contains some keywords described in Gherkin language. Let us discuss some of them:

* **Feature –** Defines a particular feature that we will test.
* **Description –** This is not mandatory and it describes what the feature deals in.
* **Scenario –** Describes a particular scenario.
* **Given –** Describes the prerequisites of a test scenario.
* **Then –** Describes the expected outcome of the test.
* **And –** Describes some of the further conditions to be tested.

Feature file implementation.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | Feature: Payment Verification       Description: Verify if the user can complete a payment       Scenario: Verify the credit card payment type       Given User navigates to Credit Card Payment Page        Then Proceed With Payment button should be enabled         And Credit card payment should be done. |

### **9) Describe a step definition file with an example.**

Answer: First of all we have created a feature file with the test scenarios. But the Cucumber is still unaware of which code to be run for a particular scenario. This is determined by the step definition file which gives one to one mapping of each scenario in the feature file to the function to be executed.

Thus a step definition file is a file with **.java extension** and contains all the methods in a class having annotations that map them to the scenarios in the feature file. All the keywords used in the Feature file is added to the step definition file in the form of annotations and then importing **cucumber.api.java.en** package.

Now let us create a step definition file for the **Feature** Payment Verification in the feature file discussed above.

Step Definition file implementation.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | import cucumber.api.java.en.Given;  import cucumber.api.java.en.Then;  import cucumber.api.java.en.And;  public class PaymentVerification{    @Given ("^User navigates to Credit Card Payment Page$")  public void navigate\_to\_credit\_card\_payment (String url) {        System.out.println ("Url is : " + url);      }    @Then ("^Proceed With Payment button should be enabled$")  public void verify\_payment\_button\_status () {        System.out.println ("Button is enabled ");        }    @And ("^Credit card payment should be done$")  public void verify\_payment\_status () {        System.out.println ("Credit card payment is successful");        }  } |

### **10) How to achieve parameterization in the Cucumber framework?**

Answer: We can achieve parameterization in Cucumber. This helps to pass multiple data sets at runtime in multiple combinations. We can perform data parameterization in the following ways:

* With the help of keyword Examples.
* Without the help of keyword Examples.

The Scenario Outline in a feature file should be accompanied by the Examples part. This consists of the multiple data sets to be passed at runtime.

The Examples section in the feature file should have headings that match with the variables in the Scenario Outline followed by (|) symbol. Each row below the heading represents a set of data. So if there are two rows after heading, the test scenario shall run two times with the two different sets of data.

Let us consider the feature file having the Feature New User Registration described previously. Next let us now create a step definition file corresponding to that feature file.

Step Definition file implementation with Examples.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | import cucumber.api.java.en.Given;  import cucumber.api.java.en.Then;  public class ParameterizationWithExample {  @Given ("^User navigates to Registration Page$")  public void navigate\_to\_registration () {        System.out.println ("Navigation to registration is done");      }    @Then ("^User inputs \"(.\*)\" and \"(.\*)\" and \"(.\*)\"$")  public void user\_input (String fname, String lname, String email){        System.out.println ("First name is: " + fname);        System.out.println ("Last name is: " + lname);     System.out.println ("Email is: " + email); }        } |

In the above example, we have passed three values Firstname, Lastname and Email at the run time without requiring to hardcode test data inside the step definition file. With the help of **User inputs \”(.\*)\” and \”(.\*)\” and \”(.\*)\** statement, Cucumber understands that there are three runtime parameters. Also, three parameters are passed as arguments to the **user\_input** method.

Now let us see how to do data parameterization without Examples.

Feature file implementation.

**Feature:** Launch Software Testing Material application  
**Scenario:** Software Testing Material launching URL  
**Given** Navigate to “https://www.softwaretestingmaterial.com/”

In the above example, we have passed the URL directly within the Given statement in the feature file.

Step Definition code Implementation.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | import cucumber.api.java.en.Given;  import cucumber.api.java.en.Then;  public class ParameterizationWithoutExample {  @Given ("^Navigate to \"([^\"]\*)\"$")  public void navigate (String url) {        System.out.println ("URL is: " + url); }      } |

In the above example, we have passed one value **https://www.softwaretestingmaterial.com/** that executes at run time without requiring to hardcode test data inside the step definition file. With the help of **Navigate to \”([^\”]\*)\** statement, Cucumber understands that there is one runtime parameter. Also, this parameter is passed as an argument to the navigate method.

### **11) How is the Options tag used in the Cucumber framework?**

Answer: The Cucumber Options tag is a part of the TestRunner file and comes in the form of an annotation called **@CucumberOptions**. It contains parameters like **feature** and **glue**. The feature parameter has the path of the feature file and the glue parameter has the path of the step definition file.

Code Implementation of TestRunner file with Option tag.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | import org.junit.runner.RunWith;  import cucumber.api.CucumberOptions;  import cucumber.api.junit.Cucumber;  @RunWith (Cucumber.class)  @CucumberOptions (  features = "src/test/java/features ",  glue = {"stepDefinitions"}  )  public class TestRunner {  } |

We have to import **org.junit.runner.RunWith** for the @RunWith annotation and **cucumber.api.CucumberOptions** for the @CucumberOptions annotation.

### **12) What are some of the keywords used to build a scenario in Cucumber?**

Answer: various keywords that are used in Cucumber for writing a scenario

* Given
* When
* Then
* And

### **13) Explain the purpose of keywords that are used for writing a scenario in Cucumber.**

Answer:

**“Given”** keyword is used to specify a precondition for the scenario.  
**“When”** keyword is used to specify an operation to be performed.  
**“Then”** keyword is used to specify the expected result of a performed action.  
**“And”** keyword is used to join one or more statements together into a single statement.

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

* Gradle
* Maven

### **15) Describe a TestRunner file with an example.**

Answer: A TestRunner file contains a JUnit class. It contains the @RunWith and @CucumberOptions annotations. We need to import **org.junit.runner.RunWith** and **cucumber.api.CucumberOptions** for the annotations @RunWith and @CucumberOptions respectively.

The @CucumberOptions has parameters like features, glue, tags, plugin, and so on. The **features** parameter contains the location of the feature file. The **tags** parameter is used to include or exclude a tagged scenario in a feature file in the execution. If we provide the (~) before the tag name, we are simply excluding that scenario from the run.

The parameter **plugin** is used for Cucumber reports in various formats like HTML, JSON, and so on. The **pretty, html: test-output** values in the plugin is used to generate an HTML report inside the test-output folder within the framework. The parameter **strict** is used to verify whether any step is not included in the step definition file and the **dryRun** parameter is used to validate if there is a proper mapping between the step definition file and the feature file.

Inside the JUnit class, we can add annotations like @BeforeClass and @AfterClass, to run a precondition or a postcondition for a test. We have to **import org.junit.BeforeClass** and **org.junit.AfterClass** for the @BeforeClass and @AfterClass annotations respectively.

Code Implementation of TestRunner file.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | import org.junit.runner.RunWith;  import cucumber.api.CucumberOptions;  import cucumber.api.junit.Cucumber;  import org.junit.BeforeClass;  import org.junit.AfterClass;  @RunWith (Cucumber.class)  @CucumberOptions (  features = "src/test/java/features ",  glue = {"stepDefinitions"},  plugin = {"pretty", "html:test-output"},  strict = true,  dryRun = false,  tags = {"~@Smoke" , "@Payment"}  )  public class TestRunner {    @BeforeClass  public void before () {    System.out.println ("Executes before all tests"); }    @AfterClass  public void after () {    System.out.println ("Executes after all tests");     }  } |

### **16) Can Cucumber be integrated with Selenium?**

Answer: Yes we can integrate Cucumber with Selenium. First of all we need to download the Selenium-server-standalone jar from the below link and add this to our project:

https://www.selenium.dev/downloads/

Next, we should navigate to the Maven repository with the below link:

https://mvnrepository.com/search?q=Cucumber

Then download the following Cucumber Jars listed below and add them to the Project:

* JUnit
* Gherkin
* Cucumber-JUnit
* Cucumber-Java
* Hemcrest-core
* Cucumber – reporting
* Cucumber – html
* Cucumber – core
* Cobertura – code coverage
* Cucumber – jvm –deps

Next we have to install the Cucumber Eclipse plugin for which we have to navigate to Install New Software from the Help menu in Eclipse and then add the link: http://cucumber.github.io/cucumber-eclipse/update-site/ for installation.

Then we have to create a feature file with the test scenarios in a step by step approach in Gherkin language, a step definition file for actual Selenium code implementation of the scenarios, and the test runner file to run the Cucumber tests.

### **17) Which language is used in Cucumber?**

Answer: The language that Cucumber understands is called Gherkin. It is a simple English representation of the application behavior. It is used to define test cases. It is designed to be non-technical and human-readable, and collectively describes use cases relating to a software system. It is a Business Readable, Domain Specific Language (DSL).

### **18) Define programming language that is used by Cucumber?**

Answer: Cucumber tool provides support for numerous programming languages such as Java, .Net, Ruby, etc. It can also be included with several tools such as Capybara, Selenium, etc.

### **19) What are hooks in the Cucumber framework?**

Answer: A block of code tagged with hooks in Cucumber can run prior or post a scenario with the help of @Before and @After annotations. Thus hooks help to control the flow of the program and optimize lines of code.

There may be scenarios that require certain preconditions to be executed like launching the application, establishing database connection, configuring the test data, and so on. Also, there are certain postconditions to be executed like terminating database connection, closing the browser, refreshing test data, application log out, and so on.

All these conditions are handled in Cucumber with the help of the hooks. The @Before hook executes before the actual scenario and the @After hook executes after the actual scenario even if the test corresponding to the actual scenario fails.

We have to **import cucumber.api.java.en.Before** for the @Before annotation and **import cucumber.api.java.en.After** for the @After annotation.

Code Implementation with hooks.

Feature file.

* **Feature:** Invoice Generation
* **Scenario:** Verify the invoice generates in pdf format
* **Given** User navigates to Invoice Page
* **Then** User should be able to open the invoice in pdf format
* **Scenario:** Verify the invoice fields
* **Given** User opens the invoice
* **Then** User should be able to see the payment amount in invoice

In the above feature file, we have included two scenarios.

Step definition file.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | import cucumber.api.java.en.Given;  import cucumber.api.java.en.Then;  import cucumber.api.java.en.Before;  import cucumber.api.java.en.After;  public class InvoiceValidation {    @Given ("^User navigates to Invoice Page$")  public void navigate\_to\_invoice () {        System.out.println ("Navigation to Invoice Page");      }  @Then ("^User should be able to open the invoice in pdf format$")  public void open\_invoice\_pdf () {        System.out.println ("Invoice opened in pdf format ");        }  @Given ("^User opens the invoice$")  public void open\_invoice () {        System.out.println ("Invoice is opened");      }  @Then ("^User should be able to see the payment amount in invoice$")  public void verify\_payment\_invoice () {        System.out.println ("Invoice contains payment amount");  }  @Before      public void precondition () {          System.out.println ("The method runs before all scenarios");      }  @After      public void postcondition () {          System.out.println ("The method runs after all scenarios ");      }  } |

In the above example, since there are two scenarios, the test method precondition with @Before annotation and the test method postcondition with @After annotation shall run twice.

### **20) How to comment on the Cucumber framework?**

Answer: We can add comments in the Cucumber code. The comments are incorporated in code mostly for documentation and not for including program logic. It makes the code easier to apprehend and to debug the errors. Also, comments can be put at any point in the code.

Inside the feature file, the comments are added after the (#) symbol and the step definition contains comments after the (//) symbol.

Feature file with comments.

* **Feature:** Customer verification
* # Scenario with customer
* **Scenario:** Verify customer details
* **Given** User navigates to Customer Information Page

Step definition file with comments

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | import cucumber.api.java.en.Given;  public class CustomerInformation {    @Given ("^User navigates to Customer Information Page$")  //method to map @Given scenario in feature file  public void navigate\_to\_customerInformation () {        System.out.println ("Customer Information Page is navigated");      }  } |

### **21) How to use regular expressions in the Cucumber framework?**

Answer: We can use regular expressions in the Cucumber framework inside the step definition file. The regular expression can be used to combine two or more similar steps in the feature file within a test method in the step definition file. Also, the regular expression is used for data parameterization in Cucumber as the Feature **New User Registration** in the feature file described previously.

Feature file implementation.

* **Feature:** Student Study Schedule
* **Description:** Verify student study schedule
* **Scenario:** Verify hours spent on science subjects
* **Given** Student studies Chemistry on Tuesday
* **Given** Student studies Mathematics on Tuesday
* **Given** Student has thirty minutes recess

Step definition file with regular expression.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | import cucumber.api.java.en.Given;  public class StudentSchedule {    @Given ("^Student studies ([^\"]\*) on Tuesday$")  public void student\_Tuesday\_schedule (String subject) {         if (subject.equals ("Chemistry")){                System.out.println ("Student studies Chemistry on Tuesday");          } else {                 System.out.println ("Student studies Mathematics on Tuesday");           }      }  @Given ("^Student has thirty minutes recess$")  public void student\_recess () {        System.out.println ("Student recess time is thirty minutes");      }  } |

In the above example, we have computed the subject’s **Chemistry** and **Mathematics** inside one method **student\_Tuesday\_schedule** without requiring to hardcode test data inside the step definition file. With the help of **Student studies ([^\”]\*)** statement we have combined two **Given (Student studies Chemistry on Tuesday and Student studies Mathematics on Tuesday)** statements inside one method.

### **22) How to skip tests in the Cucumber framework?**

Answer: We can skip tests in the Cucumber framework with the help of the tags concept which comes under CucumberOptions in the TestRunner file. We can tag a scenario in the feature file with help **@<<tagname>>** keyword.

A scenario inside the feature file can have one or multiple tags. With the help of the tagging, we can segregate test scenarios. We can mention these tagged scenarios inside the **tags** parameter under @CucumberOptions in the TestRunner file. To skip tests in Cucumber we have to pass the **~<<tagname>>** expression within the tags parameter.

Feature file implementation with tags.

* **Feature:** Insurance Details
* **@Information**
* **Scenario:** To verify Insurance information
* **Given** User is on the Insurance Information Page
* **@Premium**
* **Scenario:** To verify Insurance premium information
* **Given** User is on the Insurance Premium Page

Code Implementation of TestRunner file to skip test.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | import org.junit.runner.RunWith;  import cucumber.api.CucumberOptions;  import cucumber.api.junit.Cucumber;  @RunWith (Cucumber.class)  @CucumberOptions (  features = "src/test/java/features ",  glue = {"stepDefinitions"},  tags = {"~@Premium"}  )  public class TestRunner {  } |

In the above example, since there are two scenarios with tags **@Information** and **@Premium**, the test scenario with @Information shall run and the test scenario @Premium shall skip since the parameter **tags = {“~@Premium”}** has been added in the TestRunner file under @CucumberOptions.

### **23) How to set priority to tests in the Cucumber framework?**

Answer: We can set priority to tests in Cucumber to determine the order of execution. We use Cucumber hooks to control the flow of execution. But this can be modified with the help of the **order**.

Let us take a step definition file having two test methods with @Before annotations. In order to control the sequence of their execution, we can use **@Before (order = int)** statement. This ensures that the test methods are executed in an incremental manner. This means the test method having order = 1 shall execute before the method having order = 2.

Thus we can set priority to a test method such that the method having lower order shall be executed first followed by the test method having higher order.

Step definition file implementation.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | import cucumber.api.java.en.Given;  import cucumber.api.java.en.Before;  public class PriorityTest {    @Given ("^User navigates to Reorder page$")  public void navigate\_reorder () {        System.out.println ("Reorder page is navigated");      }  @Before (order = 2)      public void precondition1 () {          System.out.println ("The precondition1 is to be executed");      }  @Before (order = 1)      public void precondition2 () {          System.out.println ("The precondition2 is to be executed");      }  } |

In the above example, the test method **precondition2** [having order = 1] shall be executed first, followed by the test method **precondition1** [having order = 2]. Finally the test method **navigate\_reorder** will be executed.

### **24) How to run a precondition test for a scenario in the Cucumber framework?**

Answer: We can run a precondition test for a scenario in Cucumber. A scenario inside the feature file can have one or multiple tags. With the help of the tagging we can segregate test scenarios.

A block of code tagged with hooks in Cucumber can run prior to a scenario with the help of @Before annotation. Thus tagged hooks help to control the flow of the program. There may be a scenario that requires a certain precondition to be executed.

This kind of situation is handled in Cucumber with the help of the tagged hooks. The **@Before (“Scenario Tagname”)** expression is used to execute a certain precondition for a scenario. We have to **import cucumber.api.java.en.Before** for the @Before annotation.

Feature file implementation with hooked tags on scenarios.

* **Feature:** Buy Insurance
* **@Dental**
* **Scenario:** To verify buying of Dental Insurance
* **Given** User is on the Dental Insurance Page
* **@Car**
* **Scenario:** To verify buying of Car Insurance
* **Given** User is on the Car Insurance Page
* **@Accidental**
* **Scenario:** To verify buying of Accidental Insurance
* **Given** User is on the Accidental Insurance Page

In the above feature file, we have included three scenarios.

Step definition file implementation.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | import cucumber.api.java.en.Given;  import cucumber.api.java.en.Before;  public class Insurance {  @Given ("^User is on the Dental Insurance Page$")  public void navigate\_dental\_Insurance () {        System.out.println ("Navigation to Dental Insurance Page");      }  @Given ("^User is on the Car Insurance Page$")  public void navigate\_car\_Insurance () {        System.out.println ("Navigation to Car Insurance Page");      }  @Given ("^User is on the Accidental Insurance Page$")  public void navigate\_accidental\_Insurance () {        System.out.println ("Navigation to Accidental Insurance Page");      }  @Before ("@Car")      public void preconditionCar () {          System.out.println ("Method runs before car insurance scenario");      }  @Before ("@Dental")      public void preconditionDental () {       System.out.println ("Method runs before dental insurance scenario");      }  @Before ("@Accidental")      public void preconditionAccidental () {        System.out.println ("Method runs before accidental insurance"); }      } |

In the above example, the test method **preconditionCar** [having @Before (“@Car”)] shall be executed before the **navigate\_car\_Insurance** method since it is tagged with @Car in the feature file. We can say that the **preconditionCar** method is just like a precondition to the **navigate\_car\_Insurance** method.

The test method **preconditionDental** [having @Before (“@Dental”)] shall be executed before the **navigate\_dental\_Insurance** method since it is tagged with @Dental in the feature file. We can say that the **preconditionDental** method is just like a precondition to the **navigate\_dental\_Insurance** method.

The test method **preconditionAccidental** [having @Before (“@Accidental”)] shall be executed before the **navigate\_accidental\_Insurance** method since it is tagged with @Accidental in the feature file. We can say that the **preconditionAccidental** method is just like a precondition to the **navigate\_accidental\_Insurance** method.

### **25) How to run a selected test from a group of tests in Cucumber framework?**

Answer: We can run a selected test from a group of tests in the Cucumber framework with the help of tags concept. This comes under the @CucumberOptions in the TestRunner file. We can tag a scenario in the feature file with help **@<<tagname>>** keyword.

A scenario inside the feature file can have one or multiple tags. With the help of the tagging, we can segregate test scenarios. We can mention these tagged scenarios inside the **tags** parameter under @CucumberOptions in the TestRunner file. To run a selected test in Cucumber we have to pass the **<<tagname>>** value and to exclude from running we have to pass the **<<~tagname>>** value within the tags parameter.

Let us consider the **Feature:** Buy Insurance in the feature file described previously.

Code Implementation of TestRunner file to run selected test.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | import org.junit.runner.RunWith;  import cucumber.api.CucumberOptions;  import cucumber.api.junit.Cucumber;  @RunWith (Cucumber.class)  @CucumberOptions (  features = "src/test/java/features ",  glue = {"stepDefinitions"},  tags = {"@Dental", "@Car", "~Accidental" }  )  public class TestRunner {  } |

In the above example, there are three scenarios, but only the test scenarios with @Dental, @Car tags shall run and the test scenario with @Accidental tag shall not be included in the execution.

### **26) Why is the property glue used in the Cucumber Options tag?**

Answer: The property glue is used with the Cucumber Options tag which is a part of the TestRunner file. It takes the location or path of the step definition file.

Code Implementation of TestRunner file with glue.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | import org.junit.runner.RunWith;  import cucumber.api.CucumberOptions;  import cucumber.api.junit.Cucumber;  @RunWith (Cucumber.class)  @CucumberOptions (  features = "src/test/java/features ",  glue = {"stepDefinitions"}  )  public class TestRunner {  } |

### **27) What is a test harness?**

Answer: A test harness is a framework that contains test cases, test data, certain parameters and configurations, collects test results and records them.

### **28) How to generate reports with Cucumber?**

We can generate the output/report of the cucumber using different cucumber commands.

>cucumber adding.feature –format HTML  
>cucumber adding.feature –out report.html  
>cucumber adding.feature –format pretty

The report file will be stored in the project folder itself.

^^^^^^^^^^^^^^

Interview questions

1) What is Cucumber? Why is it used?

Cucumber is a testing tool based on Behavior Driven Development (BDD) framework. It is used to run functional tests written in plain text and develop test cases for software functionality. It plays a supporting role in automated testing.

In other words, we can say that "Cucumber is a software tool used by the testers to develop test cases for the testing of behavior of the software."

2) In which language is Cucumber software written?

Cucumber software is written in a Ruby programming language.

3) What is the main aim of the Behavior Driven Development (BDD) framework?

The main aim of the Behavior Driven Development framework is to make various project roles such as Business Analysts, Quality Assurance, Developers, etc., understand the application without diving deep into the technical aspects.

4) What language is used by the Cucumber tool?

The Cucumber tool uses the Gherkin language, a simple English representation of the application behavior. The Gherkin language uses several keywords to describe the behavior of applications such as Feature, Scenario, Scenario Outline, Given, When, Then, etc.

5) What are the two files required to execute a Cucumber test scenario?

Following are the two files required to execute a Cucumber test scenario:

Features

Step Definition

6) What do you understand by a feature file?

A feature file is used to provide a high-level description of an Application Under Test (AUT). The first line of the feature file must start with the keyword 'Feature' followed by the description of the application under test. A feature file may include multiple scenarios within the same file, and the extension of the feature file must be ".feature."

7) What are the various keywords used in the Cucumber tool for writing a scenario?

Following are the keywords that are used for writing a scenario in the Cucumber tool:

Given

When

Then

And

8) What is the use of the Background keyword in Cucumber?

In the Cucumber tool, the Background keyword is used to group multiple given statements into a single group. The keyword is mostly used when the same set of given statements are repeated in each scenario of the feature file.

9) What is the usage of a Scenario Outline in the Cucumber tool?

In Cucumber, a Scenario outline is used as a parameter of scenarios. This is used when the same scenario needs to be executed for multiple sets of data; however, the test steps remain the same. Scenario Outline must be followed by the keyword 'Examples', which specify the set of values for each parameter.

10) What do you understand by the term step definition in Cucumber?

In Cucumber, a step definition is the actual code implementation of the feature mentioned in the feature file.

11) Which programming languages are supported by Cucumber?

The Cucumber tool supports multiple programming languages such as Java, .Net, Ruby, etc. You can also integrate this tool easily with multiple tools such as Selenium, Capybara, etc.

12) What are the differences between Jbehave and Cucumber?

Although Cucumber and Jbehave are designed for the same purpose, the most distinctive difference between them is that Jbehave is based on stories while Cucumber is based on features.

13) What do you understand by regular expressions?

A regular expression is a pattern used to describe a certain amount of text. The most basic regular expression consists of a single literal character.

14) What software is used to run a Cucumber Web Test case?

Following software is used to run a Cucumber Web Test case:

Cucumber

Ruby and its Development Kit

IDE like ActiveState

Watir ( To simulate browser)

Ansicon and RSpec (if required)

15) What are the most significant advantages of the Cucumber framework?

Following is the list of advantages of the Cucumber framework that makes Cucumber an ideal choice for rapidly evolving agile methodology in today's corporate world:

Cucumber is an open-source and free-to-use tool.

It is easy to use. Even non-technical users can also understand the scenarios because of its plain text representation.

It bridges the communication gap between various project stakeholders such as Business Analysts, Developers, and Quality Assurance personnel.

With the Cucumber tool, we can develop automation test cases that are easier to maintain and understand.

It is easy to integrate with other tools such as Selenium and Capybara.

16) What do you understand by test harness in Cucumber?

In Cucumber, the test harness allows for separating responsibility between setting up the context and interacting with the browser, and cleaning up the step definition files. It collects stubs, drivers, and other supporting tools required to automate test execution in testing.

17) What is the difference between RSpec and Cucumber? When should we use RSpec and when to use Cucumber?

RSpec and Cucumber both are the example of testing frameworks. RSpec uses traditional Unit Testing. It means it uses testing a class or part of the application in isolation from the rest of the application. So your model does what your model is supposed to do, the controller does what it is supposed to do, etc. RSpec and Cucumber both are used for Acceptance Testing, also called ATDD, BDD, etc.

Difference between RSpec and Cucumber

The main difference between RSpec and Cucumber is the business readability factor.

RSpec is mainly used for Unit Testing. On the other hand, Cucumber is mainly used for Behavior-driven development. We can also use it for System and Integration Testing.

In Cucumber, the specifications or features are separate from the test code, so the product owners can provide or review the specification without going through the code. These are the .feature files that you make in Cucumber.

RSpec also has a similar mechanism, but instead of describing a step with a Describe or Context, it uses the business specification to execute that statement. This approach is a little easier for developers to work with but a bit harder for non-technical guys.

Which should we use?

For a core developer, it is the best choice to use RSpec. It is easier to understand for a technical person and offers a few advantages in keeping things scoped and under control because you don't have to mess up with RegExs for test steps.

If you are building this for a client, you should choose Cucumber for Acceptance Testing and use RSpec for Unit Testing.

18) What is Selenium?

Selenium is a web browser automation tool widely used for Functional Testing of web-based applications. Selenium supports different programming languages such as Java, Python, Ruby, C#, etc.

19) What is the difference between Selenium and Cucumber?

Selenium and Cucumber are both open-source testing tools, and both are used for functional testing. But there are some differences between them.

Following are some critical differences between Selenium and Cucumber:

Selenium is a web browser automation tool for web apps, while Cucumber is an automation tool for behavior-driven development that can be used with Selenium (or Appium).

Selenium is used for automated UI testing, while Cucumber is used for acceptance testing.

Selenium is preferred by technical teams (SDETs/programmers), while Cucumber is typically preferred by non-technical teams (business stakeholders and testers).

Selenium can work independently of Cucumber. Cucumber depends on Selenium or Appium for step-definition implementation.

In Selenium, the script creation is complex, while Cucumber is simpler than Selenium.

20) Why we have to use Cucumber with Selenium?

Cucumber and Selenium are both testing frameworks and prevalent technologies. Many organizations use Selenium for functional testing. Along with Selenium, these organizations integrate Cucumber with Selenium as Cucumber makes it easy to read and understand the application flow. The most significant benefit of using Cucumber with Selenium is that it facilitates developers to write test cases in simple feature files easily understood by managers, non-technical stakeholders, and business analysts. It provides the facility to write tests in a human-readable language called Gherkin. The Selenium-Cucumber framework supports programming languages such as Java, .NET, PHP, Python, Perl, etc.

21) What do you understand by the Step Definition?

Step definition is used to map the Test Case Steps in the feature files to code. It executes the steps on Application under Test and checks the results. To execute step definition, it must match the given component in a feature.

22) What are the maximum numbers of scenarios that we can includ in the feature file?

In Cucumber, a feature file can contain a maximum of 10 scenarios. This number can vary from project to project and from one organization to another organization. It is the best practice to limit the number of scenarios included in the feature file.

23) What is the purpose of the behavior-driven development (BDD) methodology in the real world?

Behavior Driven Development or BDD is a methodology used to understand the functionality of an application in the simple plain text representation. The primary purpose of the Behavior Driven Development framework is to make various project roles such as Business Analysts, Quality Assurance, Developers, Support Teams understand the application without diving deep into the technical aspects.

24) What do you understand by TDD, and what are the different processes used in TDD?

TDD is an acronym that stands for Test-Driven Development. This is a software development technique used to create the test cases first and then write the code underlying those test cases. Although TDD is a development technique, it can also be used for automation testing development. TDD takes more time for development because it tends to find very few defects. The result provided by the TDD development technique has improved the quality of code, and that can be more reusable and flexible. TDD also helps developers to achieve high test coverage of about 90-100%. The only disadvantage for developers following TDD is to write their test cases before writing the code.

Following is the list of simple 6 step process used by TDD methodology:

First, write the test case: You have to write an automated test case according to your requirements.

Run all the test cases: Now, run these automated test cases on the currently developed code.

Develop the code for that test case: In this process, you must write the code to make that test case work as expected if the test case fails.

Run test cases again: Now, you have to rerun the test cases and check if all the test cases developed so far are implemented.

Refactor your code: This is an optional step. But, it is advised to refactor your code to make it more readable and reusable. That's why it is essential.

Repeat steps 1- 5 for new test cases: This is the last step. Here, you have to repeat the cycle for the other test cases until all the test cases are implemented.

25) What are the similarities between BDD and TDD?

TDD stands for Test-Driven Development, and BDD stands for Behavior Driven Development. Both are two software development techniques.

BDD and TDD are both very similar as they are both testing strategies for a software application. In both cases, the developers have to write the test before writing the code to pass the test. The second main similarity between them is in both cases; the tests can be used as part of an automated testing framework to prevent bugs.

26) What are the main differences between TDD and BDD?

Following is the list of main differences between TDD and BDD:

TDD BDD

TDD stands for Test-Driven Development. It is a test-centered development process. This means the developers have first to write the test cases then code. BDD stands for Behavior Driven Development. It is a Behavior centered development process.

In TDD, writing a test fails because the specified functionality doesn't exist, then writing the most straightforward code that can make the test pass, then refactoring to remove duplication, etc. In BDD, creating an executable specification that fails because the feature doesn't exist, then writing the most straightforward code that can make the spec pass. You repeat this until a release candidate is ready to ship.

TDD tests are written using programming languages such as Java, .Net, Python, Ruby, etc. BDD tests are written in a human-readable format using Given-When-Then steps. These tests are readable and understandable by non-technical persons also.

TDD tests are difficult to read by non-programmers as they are written in specific programming languages. BDD tests are readable by non-programmers also as they are written in a human-readable format.

The critical difference between TDD and BDD is the scope. TDD is a development practice. On the other hand, BDD is a team methodology.

In TDD, the developers write the test cases. In BDD, the automated specifications are created by users or testers then the developers wiring them to the code under test.

27) What do you understand by cucumber dry run?

Cucumber dry run is used to compile cucumber features files and step definitions. It is run to find any compilation errors. If it finds anyone, it will show when we use dry run.

28) What do you understand by the TestRunner class in the Cucumber testing approach? Explain with example.

In the Cucumber testing approach, the TestRunner class provides the link between the feature file and the step definition file. The TestRunner class is generally an empty class with no class definition.

Example of a TestRunner class in Cucumber:

Package com.sample.TestRunner

Import org.junit.runner.RunWith;

Import cucumber.api.CucumberOptions;

Import cucumber.api.junit.Cucumber;

@RunWith(Cucumber.class)

@CucumberOptions(features="Features",glue={"StepDefinition"})

public class Runner

{

}

29) Should we write code within the TestRunner class?

It is advised not to write code under the TestRunner class. It should include the tags @RunWith and @CucumberOptions.

30) What is the starting point of execution for feature files?

When Cucumber is integrated with Selenium, the starting point of execution must be from the TestRunner class.

31) How can you use the Options tag in the Cucumber framework?

In the Cucumber framework, the Options tag is a part of the TestRunner file and comes in the form of an annotation called @CucumberOptions. It contains two parameters feature and glue.

Feature parameter: The feature parameter is used to specify the path of the feature file.

Glue parameter: The glue parameter is used to specify the path of the step definition file.

See the code implementation of TestRunner file with Option tag:

import org.junit.runner.RunWith;

import cucumber.api.CucumberOptions;

import cucumber.api.junit.Cucumber;

@RunWith (Cucumber.class)

@CucumberOptions (

features = "src/test/java/features ",

glue = {"stepDefinitions"}

)

public class TestRunner {

}

We have to import org.junit.runner.RunWith for the @RunWith annotation and cucumber.api.CucumberOptions for the @CucumberOptions annotation.

32) What is the use of features property under the Cucumber Options tag?

In the Cucumber framework, the features property is used to identify the location of the feature files.

33) What is the use of glue property under the Cucumber Options tag?

The Glue property is used to facilitate the Cucumber framework to identify the location of step definition files.

34) What are the two build management tools that can be integrated with Cucumber?

Following are the two build management tools that can be integrated with Cucumber:

Gradle

Maven

35) What is the use of hooks in the Cucumber framework?

In the Cucumber framework, the hooks are used to control the flow of the program and optimize lines of code. A block of code tagged with hooks in Cucumber can run before or post a scenario with the help of @Before and @After annotations.

Some scenarios may require certain preconditions for execution, such as launching the application, establishing a database connection, configuring the test data, and so on. Also, certain postconditions should be executed, such as terminating database connection, closing the browser, refreshing test data, application log out, and so on. All these conditions are handled in Cucumber with the help of the hooks. The @Before hook executes before the actual scenario, and the @After hook executes after the actual scenario even if the test corresponding to the actual scenario fails.

For the @Before annotation, we have to import cucumber.api.java.en.Before and for the @After annotation, we have to import cucumber.api.java.en.After.

See the code implementation with hooks:

The feature file: In this feature file, we have included two scenarios:

Feature: Invoice Generation.

Scenario 1: Verify the invoice generates in pdf format.

Given User navigates to Invoice Page.

Then User should be able to open the invoice in pdf format.

Scenario2: Verify the invoice fields.

Given User opens the invoice.

Then User should be able to see the payment amount in invoice.

See the step definition file:

import cucumber.api.java.en.Given;

import cucumber.api.java.en.Then;

import cucumber.api.java.en.Before;

import cucumber.api.java.en.After;

public class InvoiceValidation {

@Given ("^User navigates to Invoice Page$")

public void navigate\_to\_invoice () {

System.out.println ("Navigation to Invoice Page");

}

@Then ("^User should be able to open the invoice in pdf format$")

public void open\_invoice\_pdf () {

System.out.println ("Invoice opened in pdf format ");

}

@Given ("^User opens the invoice$")

public void open\_invoice () {

System.out.println ("Invoice is opened");

}

@Then ("^User should be able to see the payment amount in invoice$")

public void verify\_payment\_invoice () {

System.out.println ("Invoice contains payment amount");

}

@Before

public void precondition () {

System.out.println ("The method runs before all scenarios");

}

@After

public void postcondition () {

System.out.println ("The method runs after all scenarios ");

}

}

In the above example, you can see that we have used two scenarios, the test method precondition with @Before annotation and the test method postcondition with @After annotation.