1. **Feature File:**

Every .feature file conventionally consists of a single feature. A line starting with the keyword **Feature** followed by free indented text starts a feature. A feature usually contains a list of scenarios. You can write whatever you want up until the first scenario, which starts with the word **Scenario** (or localized equivalent; Gherkin is localized for [dozens of languages](https://github.com/cucumber/cucumber/wiki/Spoken-languages)) on a new line. You can use [tagging](https://github.com/cucumber/cucumber/wiki/Tags) to group features and scenarios together independent of your file and directory structure.

Every scenario consists of a list of steps, which must start with one of the keywords **Given**, **When**, **Then**, **But** or **And**. Cucumber treats them all the same, but you shouldn’t. Here is an example:

Feature: Serve coffee

Coffee should not be served until paid for

Coffee should not be served until the button has been pressed

If there is no coffee left then money should be refunded

Scenario: Buy last coffee

Given there are 1 coffees left in the machine

And I have deposited 1$

When I press the coffee button

Then I should be served a coffee

1. **writting test cases in BDD format (given, when, and then)**

**Feature**: Login Functionality Feature

In order to ensure Login Functionality works,  
I want to run the cucumber test to verify it is working

@positiveScenario  
**Scenario**: Login Functionality

**Given** user navigates to SOFTWARETETINGHELP.COM  
**When** user logs in using Username as “USER” and Password “PASSWORD”  
**Then** login should be successful

@negaviveScenario  
**Scenario**: Login Functionality

**Given** user navigates to SOFTWARETETINGHELP.COM  
**When** user logs in using Username as “USER1” and Password “PASSWORD1”  
**Then** error message should throw

1. Scenario outline:

Scenario outlines are used when same test has to be performed with different data set. Let’s take the same example. We have to test login functionality with multiple different set of username and password.

**Feature**: Login Functionality Feature

In order to ensure Login Functionality works,  
I want to run the cucumber test to verify it is working

**Scenario Outline**: Login Functionality

**Given** usernavigates to SOFTWARETESTINGHELP.COM  
**When** user logs in using Username as <**username**> and Password <**password**>  
**Then** login should be successful

**Examples:**  
|username         |password         |  
|Tom                     |password1       |  
|Harry                   |password2       |  
|Jerry                    |password3        |

1. Scenario:

Basically a scenario represents a particular functionality which is under test. By seeing the scenario user should be able to understand the intent behind the scenario and what the test is all about. Each scenario should follow given, when and then format. This language is called as “gherkin”.

1. **Given:**As mentioned above, given specifies the pre-conditions. It is basically a known state.
2. **When**: This is used when some action is to be performed. As in above example we have seen when the user tries to log in using username and password, it becomes an action**.**
3. **Then:**The expected outcome or result should be placed here. For Instance: verify the login is successful, successful page navigation.
4. **Background:**Whenever any step is required to perform in each scenario then those steps needs to be placed in Background. For Instance: If user needs to clear database before each scenario then those steps can be put in background.
5. **And**: And is used to combine two or more same type of action.

5. **back ground**:

Background allows you to add some context to the scenarios in a single feature. A Background is much like a scenario containing a number of steps. The difference is when it is run. The background is run before each of your scenarios but after any of your Before [Hooks](https://github.com/cucumber/cucumber/wiki/Hooks).

Example:

Feature: Multiple site support

As a Mephisto site owner

I want to host blogs for different people

In order to make gigantic piles of money

Background:

Given a global administrator named "Greg"

And a blog named "Greg's anti-tax rants"

And a customer named "Dr. Bill"

And a blog named "Expensive Therapy" owned by "Dr. Bill"

1. **create step definition file**:

A step definition is analogous to a method definition / function definition in any kind of OO/procedural programming language. Step definitions can take 0 or more arguments, identified by groups in the Regexp (and an equal number of arguments to the Proc).

Some people are uncomfortable with Regular Expressions. It’s also possible to define Step Definitions using strings and $variables like this:

Given "I have $n cucumbers in my belly" do |cukes|

# Some Ruby code here

end

In this case the String gets compiled to a Regular Expression behind the scenes: /^I have (.\*) cucumbers in my belly$/.

Then there are Steps. Steps are declared in your features/\*.feature files. Here is an example:

Given I have 93 cucumbers in my belly

A step is analogous to a method or function invocation. you’re “calling” the step definition above with one argument — the string “93”. Cucumber matches the Step against the Step Definition’s Regexp and takes all of the captures from that match and passes them to the Proc.

Step Definitions start with a [preposition](http://www.merriam-webster.com/dictionary/given) or an [adverb](http://www.merriam-webster.com/dictionary/when) (**Given**, **When**, **Then**, **And**, **But**), and can be expressed in any of Cucumber’s supported [Spoken languages](https://github.com/cucumber/cucumber/wiki/Spoken-languages). All Step definitions are loaded (and defined) before Cucumber starts to execute the plain text.

When Cucumber executes the plain text, it will for each step look for a registered Step Definition with a matching Regexp. If it finds one it will execute its Proc, passing all groups from the Regexp match as arguments to the Proc.

1. **Tags:**

Cucumber by default runs all scenarios in all the feature files. In real time projects there could be hundreds of feature file which are not required to run at all times.

**For instance**: Feature files related to smoke test need not run all the time. So if you mention a tag as smokeTest in each feature file which is related to smoke test and run cucumber test with @SmokeTest tag . Cucumber will run only those feature files specific to given tags. Please follow the below example. You can specify multiple tags in one feature file.

**Example of use of single tags:**

**Feature**: Login Functionality Feature

In order to ensure Login Functionality works,  
I want to run the cucumber test to verify it is working

**Scenario Outline**: Login Functionality

**Given** user navigates to SOFTWARETESTINGHELP.COM  
**When** user logs in using Username as <**username**> and Password <**password**>  
**Then** login should be successful

**Example of use of multiple tags:**

As shown in below example same feature file can be used for smoke test scenarios as well as for login test scenario. When you intend to run your script for smoke test then use @SmokeTest. Similarly when you want your script to run for Login test use @LoginTest tag.

Any number of tags can be mentioned for a feature file as well as for scenario.

**@SmokeTest @LoginTest**

**Feature**: Login Functionality Feature

In order to ensure Login Functionality works,  
I want to run the cucumber test to verify it is working

webhooks

The setup for utilizing this collector relies heavily on the output of the Cucumber test cases. All languages that utilize cucumber should allow setting the format of the output to JSON and set the name of the output file. For the usage of this collector, all output files should be entitled "cucumber.json". Since cucumber tests can be written in many languages, there are many examples of how to enable the output format. Here are a couple of examples for setting these options in Java and Ruby: For Cucumber-JVM, the Cucumber Options should be similar to the following code snippet:

amoolya

java

@RunWith(Cucumber.class)

@CucumberOptions( features = "src/test/features/com/sample",

tags = { "~@wip", "@executeThis" }, monochrome = true,

format = { "pretty", "html:target/cucumber", "json:target/cucumber.json" } )

public class RunCukeTest {

}

1. **how to connect feature and step definition file**

open the .feature file you want to navigate

Keep the control button pressed  hover your mouse pointer over a step. The step turns to a hyperlink, and its reference information is displayed at the tooltip:

Click that hyperlink

1. **Running feature file/files from command line**

mvn -Dcucumber.options="from/the/root/of/module/to/the/feature/MyFeature.feature" test

1. **Generating Reports**

**Step 1** − Create a Maven project named **cucumberReport** in Eclipse.

**Step 2** − Create a package named **CucumberReport** under **src/test/java**

**Step 3** − Create a feature file named **cucumberReport.feature**

Write the following text within the file and save it.

**Feature** − Cucumber Report

#This is to check test result for Pass test case

**Scenario:** Login functionality exists

Given I have opened the browser

When I open Facebook website

Then Login button should exist

#This is to check test result for Failed test case

**Scenario:** Forgot password exists

Given I have open the browser

When I open Facebook website

Then Forgot password link should exist

**Note** − Here scenario first will pass, whereas the second scenario will fail. So that we can witness how the pass and failed report looks like.

**Step 4** − Create a step definition file.

* Select and right-click on the package outline.
* Click on ‘New’ file.
* Give the file name as **cucumberReport.java**
* Write the following text within the file and save it.