* [Introduction to Cucumber](https://www.edureka.co/blog/cucumber-selenium-tutorial/" \l "IntroductiontoCucumber)
  + [Why Cucumber?](https://www.edureka.co/blog/cucumber-selenium-tutorial/#WhyCucumber?)
  + [Behavior Driven Development](https://www.edureka.co/blog/cucumber-selenium-tutorial/#BehaviorDrivenDevelopment)
* [Introduction to Selenium](https://www.edureka.co/blog/cucumber-selenium-tutorial/#IntroductiontoSelenium)
* [Why use Cucumber with Selenium?](https://www.edureka.co/blog/cucumber-selenium-tutorial/#WhyuseCucumberwithSelenium?)
* [Steps to create Cucumber application](https://www.edureka.co/blog/cucumber-selenium-tutorial/#StepstocreateCucumberapplication)

**Introduction to Cucumber**



[Cucumber](https://www.edureka.co/ruby-with-cucumber-sp) is a testing approach/tool that supports ***Behaviour Driven Development (BDD).*** It provides a way to write tests that anybody can understand, regardless of their extent of technical knowledge.

It explains the behavior of the application in a simple English text using ***Gherkin*** language. I hope that you got a glimpse about what is Cucumber. Now, let’s move further and understand some of the facts that depict the need for Cucumber in a testing framework.

**Why Cucumber?**

Well, Cucumber is one of the most popular tools because of the reasons listed below:

1. Cucumber is **open source** and hence, its free to use.
2. On using Cucumber, you can write your **test scripts** in multiple languages such as [Java](https://www.edureka.co/blog/java-tutorial/), [Ruby](https://www.edureka.co/blog/ruby-on-rails-tutorial/), .NET, [Python](https://www.edureka.co/blog/python-tutorial/), etc.
3. It also integrates with [Selenium](https://www.edureka.co/blog/selenium-webdriver-architecture/), [Ruby on Rails](https://www.edureka.co/blog/ruby-on-rails-tutorial/), Watir and other **web-based testing** tools.
4. Cucumber is one of the most widely used **BDD** tools.

These are some of the unique features that make Cucumber helpful for testing a website. Now that you know what is Cucumber and why do you need it, let’s understand one of the unique features of Cucumber that makes it easy for a non-techie to understand the test cases.

**Behavior Driven Development(BDD)**

In very simple terms, BDD or *Behavior-Driven Development* is a technique where your specifications or test cases are written in plain English like sentences. With this approach, the non-technical team members find it easy to understand the flow and collaborate more in the process of software development.

Let’s understand this with the help of a simple example. Consider a scenario where you want to test the Twitter website. One of the test scenarios is to verify the login credentials. With BDD, you can write this test scenario in this format:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | Feature: Test Twitter smoke scenario    Scenario: Test login with valid credentials  Given Open the firefox and start application  When I enter valid username and valid password  Then user should be able to login successfully |

In the above example, I have just written a few statements in simple English for defining the scenarios. I will be providing more insights into this scenario and its implementation in the later part of this article. For now, let’s move further in Cucumber Selenium Tutorial article and learn the fundamentals of Selenium.

**Scenario** **Outline**: Check if String is Palindrome

**Given** I entered word <wordToTest>

**When** I test it for Palindrome

**Then** the output should be <output>

**Examples**:

    | wordToTest | output  |

    | "Refer"    | "true"  |

    | "Coin"     | "false" |

    | "Space"    | "false" |

    | "racecar"  | "true"  |

**Introduction to Selenium**

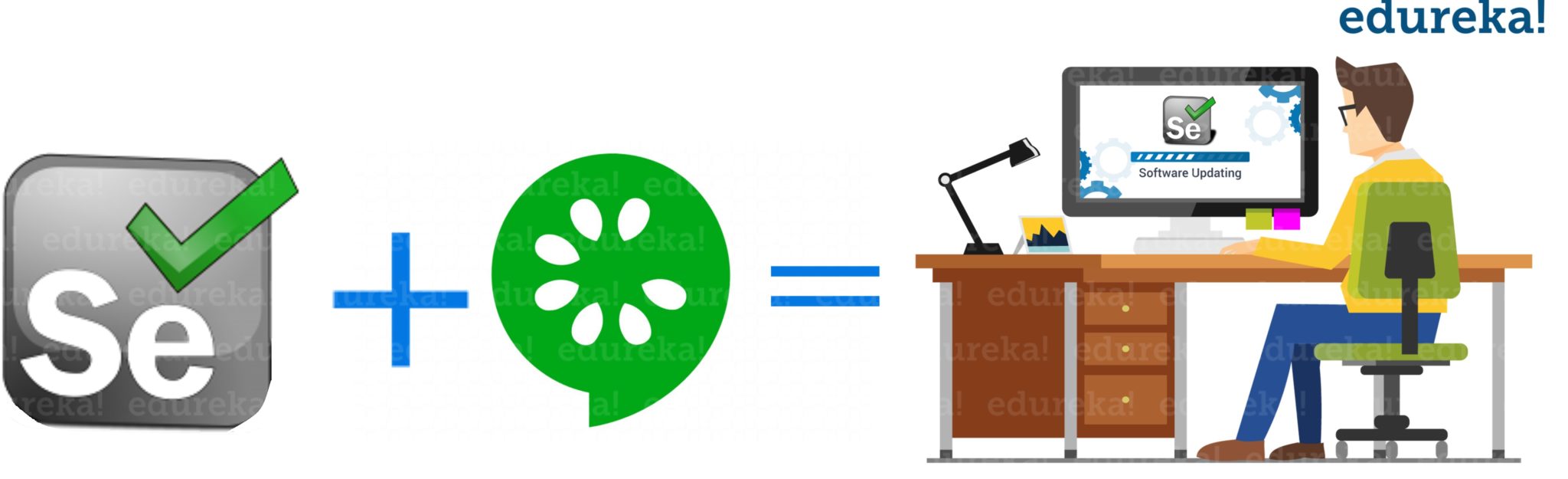


[Selenium](https://www.edureka.co/blog/what-is-selenium/) is the preferred tool when it comes to automating the tests which are carried out on web browsers. It is useful only for testing of web applications. Any desktop (software) application or mobile application cannot be tested using Selenium. It is very helpful in writing functional test cases. It also provides reliable performance with ‘**n’**number of test cases and it is obviously the best suitable automation tool for web applications.

Now that you know what is Selenium, let’s move further in this article on Cucumber Selenium Tutorial and understand why use Cucumber with Selenium?

**Why use Cucumber with Selenium?**

Many organizations use [Selenium](https://www.edureka.co/blog/handle-multiple-windows-in-selenium/) for [functional and regression testing](https://www.edureka.co/blog/regression-testing). Selenium and Cucumber are a great combination when it comes to web application automation, as Cucumber allows you to write your tests quickly in English-like language and Selenium allows you to run on various combinations of browsers.

Cucumber tool is based on the Behavior Driven Development framework that *acts as the bridge*between the Software Engineer and Business Analyst and also between Manual Tester and Developers.

When using Selenium with Cucumber for test automation, tests are written in feature files that can be understood by various stakeholders in an agile environment such as Business Analysts. Cucumber also comes with its ability to support multiple scripts and programming languages and JUnit is used to execute these scripts and generate the output.

Having understood this, let’s now see the various steps to create Cucumber application and run the test cases.

**Steps to create Cucumber Application**

The various steps involved in creating a Cucumber application are as follows:

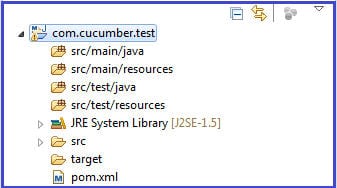
1. Download the Jar files of Cucumber and Selenium and configure the build path.
2. Add Cucumber Plugin from Eclipse Marketplace.
3. Create a feature file and add scenarios.
4. Implement the steps for the scenarios.
5. Write the runner class and execute the code.

Setup for Cucumber Java Maven Project

**Cucumber Project Setup**

**Step #1:** Create a New Maven Project:  
*Right Click -> New -> Others -> Maven -> Maven Project -> Next*

**Step #2:** Now the project will look like this:



**Step #3**: Add below dependencies in pom.xml

|  |
| --- |
| <**dependencies**>  <**dependency**>        <**groupId**>info.cukes</**groupId**>        <**artifactId**>cucumber-java</**artifactId**>        <**version**>1.0.2</**version**>        <**scope**>test</**scope**>    </**dependency**>  <**dependency**>       <**groupId**>info.cukes</**groupId**>       <**artifactId**>cucumber-junit</**artifactId**>       <**version**>1.0.2</**version**>       <**scope**>test</**scope**>  </**dependency**>  <**dependency**>       <**groupId**>junit</**groupId**>       <**artifactId**>junit</**artifactId**>       <**version**>4.10</**version**>       <**scope**>test</**scope**>  </**dependency**>  </**dependencies**> |

**Step #4**: Create a sample.feature file under src/test/resources.

@smokeTest  
**Feature**: To test my cucumber test is running  
I want to run a sample feature file.

**Scenario**: cucumber setup

**Given** sample feature file is ready  
**When** I run the feature file  
**Then** run should be successful

**Step #5**: Create a class under src/test/java which will implement all the steps.

|  |
| --- |
| **public** **class** stepDefinition {         @Given("^sample feature file is ready$")  **public** **void** givenStatment(){                System.out.println("Given statement executed successfully");         }         @When("^I run the feature file$")  **public** **void** whenStatement(){                System.out.println("When statement execueted successfully");         }         @Then("^run should be successful$")  **public** **void** thenStatment(){                System.out.println("Then statement executed successfully");         }  } |

**Step #6**: Create a JUnit runner to run the test.

|  |
| --- |
| @RunWith(Cucumber.**class**)  @Cucumber.Options(format={"pretty","html:reports/test-report"},tags= "@smokeTest")  **public** **class** CucumberRunner {  } |

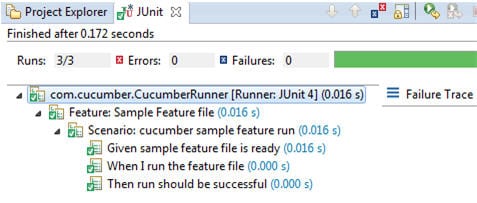
Provide the path of the report as given here. The reports will store in ‘test-report’ folder under project folder and “pretty” format specifies the type of report.

**Step #7**: Junit Result and Test Report:

Below is the report when the cucumber test is successful. The green bar in Junit describes the test is passed. Similarly, red bar describes that test has failed.

If we want to use default reporting then navigate the path mentioned in Junit Runner. In this case, we have given path as *reports->test-reports->index.html.*

Open this report in Internet Explorer or in Firefox to verify the result. Below is the sample of the report:



**Cucumber Selenium WebDriver Integration**

Cucumber framework can be used to test the web-based applications along with Selenium WebDriver. The test cases are written in simple feature files which are easily understood by managers, non-technical stakeholders and business analysts. And those feature file steps are implemented in step definition file. If you are using maven then you have to add dependencies for Cucumber and WebDriver.

So here is the sample test case we have implemented using Cucumber and WebDriver. As given below, the scenario in feature file is self-explanatory.

**Feature: Login Feature File**  
@selenium  
**Scenario**: Login scenario test for Gmail

**Given** navigate to Gmail page  
**When** user logged in using username as “userA” and password as “password”  
**Then** home page should be displayed

**WebDriver Implementation in Cucumber stepDefinitions:**

|  |
| --- |
| **public** **class** stepDefinition {  WebDriver dr;  @Given("^navigate to gmail page$")  **public** **void** navigate(){         dr=**new** FirefoxDriver();         dr.get("http://www.gmail.com");         }  @When ("^user logged in using username as \"(.\*)\" and password as \"(.\*)\"$")  **public** **void** login(String username,String password){         dr.findElement(By.xpath("//\*[@id='Email']")).sendKeys(username);         dr.findElement(By.xpath("//\*[@id='Passwd']")).sendKeys(password);         dr.findElement(By.xpath("//\*[@id='signIn']")).click();         dr.manage().timeouts().implicitlyWait(20, TimeUnit.SECONDS);         }  @Then("^home page should be displayed$")  **public** **void** verifySuccessful(){        String expectedText="Gmail";        String actualText=         dr.findElement(By.xpath("//\*[@id='gbq1']/div/a/span")).getText();        Assert.assertTrue("Login not successful",expectedText.equals(actualText));        }  } |

In this test, we have used the Firefox as the browser to test the Gmail login functionality.  
Clearly, WebDriver object is a class variable and used across the class.

**Given** statement initializes the browser and navigates to the page.  
**When** statement logs into the application using the username as “userA” and password as “password”. Both the values ‘username’ and ‘password’ are passed from feature file and both the values to be used in the same order.  
**Then** Statement only validates the conditions after logging into the application.

This is a sample test describing the usage of Cucumber and Selenium. You can create multilayer architecture depending upon your project requirement.

Cucumber TestNG Runner file

@CucumberOptions(features=’’src/test/resources/features”,glue={“steps”}, plugin={“html:target/cucumber-reports/cucumber-html-reports.html”, “com.aventstack”})

)

Public class RunCuke Extends AbstractTestNGCucumberTests{

}

How to use Extent reports

Extent reports

1. Add dependency extentreports-cucumber7-adapter
2. Add a extent.properties file in resources

Extent.reporter.spark.start=true

Extent.reporter.spark.out=target/cucumber-reports/extent.html

1. Add a plugin in runner file

Cucumber tags

@Scenario

Scenario Outline

Background

Run Scenarios based on tags

Cucumber hooks

In the hooks.java file you can write and keep this in the same folder that contains your feature files. Now when you run features the code under @Before gets executed before each scenario. Code Under @After gets execute after each scenario.

@Before

@After

@BeforeStep

@AfterStep

Create Maven project.

Add all dependencies

Add cucumber plugin

Pom.xml for jdk 1.8 and old cucumber versions

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>org.example</groupId>  
 <artifactId>Cucumberfour</artifactId>  
 <version>1.0-SNAPSHOT</version>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.seleniumhq.selenium</groupId>  
 <artifactId>selenium-java</artifactId>  
 <version>3.4.0</version>  
 </dependency>  
 <dependency>  
 <groupId>info.cukes</groupId>  
 <artifactId>cucumber-java</artifactId>  
 <version>1.2.5</version>  
 <scope>compile</scope>  
 </dependency>  
 <dependency>  
 <groupId>info.cukes</groupId>  
 <artifactId>cucumber-junit</artifactId>  
 <version>1.2.5</version>  
 <scope>compile</scope>  
 </dependency>  
 <dependency>  
 <groupId>junit</groupId>  
 <artifactId>junit</artifactId>  
 <version>4.12</version>  
 <scope>test</scope>  
 </dependency>  
  
 <dependency>  
  
 <groupId>junit</groupId>  
 <artifactId>junit</artifactId>  
 <version>RELEASE</version>  
 <scope>compile</scope>  
 </dependency>  
 <dependency>  
 <groupId>io.github.bonigarcia</groupId>  
 <artifactId>webdrivermanager</artifactId>  
 <version>5.0.3</version>  
 <scope>test</scope>  
 </dependency>  
  
 </dependencies>  
  
  
 <properties>  
 <maven.compiler.source>8</maven.compiler.source>  
 <maven.compiler.target>8</maven.compiler.target>  
 </properties>  
  
</project>

A screenshot of a computer program

Description automatically generated with medium confidence

To use extent reports in cucumber test



Rest assured code inside cucumber test

A screenshot of a computer code

Description automatically generated with low confidence

Cucumber

Hooks- @Before @After tag used to run statements for each scenario ( similar to suites in TestNG)

@BeforeStep @AfterStep for each step of scenario

A screenshot of a computer

Description automatically generated with medium confidence

References

https://www.softwaretestinghelp.com/cucumber-bdd-tool-selenium-tutorial-30/

<https://www.softwaretestinghelp.com/selenium-webdriver-cucumber-selenium-tutorial-31/#Cucumber_Selenium_WebDriver_Integration>

Cucumber 7 BDD

https://www.way2automation.com/cucumber-setup-in-eclipse/