Cucumber:

Cucumber is an automation testing tool used by Testers to execute acceptance tests in Behavior Driven Development (BDD) approach. This tool was developed using the Ruby programming language and was licensed by MIT. The test cases defined by Cucumber use Gherkin language which is a human-readable, non-technical and simple syntax of its language is used for documentation. Cucumber avoids more technical stack and will be easier to test without any programming skills. All the test cases can be written in plain English language.

What are the features in Cucumber?

Feature is keyword and also file which contains cucumber scenarios in GWT format

A feature can be defined as a unit or functionality or part of a project which is an independent functionality of the project. A feature contains a group of scenarios which are to be tested as a feature.

Eg. Feature files for Create Workflow, edit /view/ delete workflow

What are the different keywords used in feature file?

The different keywords used in the feature file are

1. Feature,
2. Background,

3. Scenario,
4. Scenario Outline,

5. Given,
6. When,
7. Then,
8. And
9. But.

7.What is Scenario Outline in Cucumber?

This is the most asked cucumber interview questions. A scenario outline replaces an identifier with the actual value from the table. Each row can be considered as a scenario. A feature file is more time taking a task and is more error-prone in testing. The same feature file can be reduced to less number of lines for execution in the scenario outline feature to increase the efficiency and decreases the runtime.

Which language is used to specify the scenarios in Cucumber?

Gherkin/GWT format

What is meant by a feature file?

Ans: 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' following 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.

What programming language is used by Cucumber?

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, Protractor Capybara etc.

What are the major advantages of Cucumber framework?

Ans: Given below are the advantages of Cucumber Gherkin framework that make Cucumber an ideal choice for rapidly evolving Agile methodology 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 and Capybara.

What is the limit for the maximum number of scenarios that can be included in the feature file?

Ans: 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.

What is the use of Background keyword in Cucumber?

Ans: 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.

Provide an example of Background keyword in Cucumber.

Ans:

Background: Given user is on the application login page.

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.

Provide an example of step definition file in Cucumber.

Ans: 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");
}
```

What is the purpose of Cucumber Options tag?

Ans: 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"})
```

How can Cucumber be integrated with Selenium WebDriver?

Ans: Cucumber can be integrated with Selenium webdriver by downloading the necessary JAR files./mentioning the below maven dependencies in pom.xml:

```
 <artifactId>selenium-java</artifactId>
```

For Junit:

```
<artifactId>cucumber-junit</artifactId>
<artifactId>junit</artifactId>
```

For Testng:

```
<artifactId>cucumber-testng</artifactId>
<artifactId>testng</artifactId>
```

What is the meaning of TestRunner class in Cucumber? //starting point of execution

Ans: TestRunner class is used to provide the link between feature file and step definition file. Below is the sample representation of how TestRunner class will look like. A TestRunner class is generally an empty class with no class definition.

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

Ans:

```
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
{
}
```

Should any code be written within TestRunner class?

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

What is the maximum number of steps that are to be written within a scenario?

Ans: The maximum number of steps to be written in a scenario is 3-4 steps.
Note-it can vary project to project. In our case Given & Then can have 3 steps, but When should have a single step

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

List out some of the main differences between Jbehave and Cucumber?
However, the Cucumber and Jbehave share the same perspective, but there are few key differences.

1- Jbehave is Java-based (Pure Java implementation) and Cucumber is Ruby-based.
2- Jbehave is story-driven whereas the Cucumber is feature-driven.

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

What are the apparent advantages of a test framework?
Following are the possible benefits of using a test framework.

1- It reduces the complexity of using a variety of technologies inculcated in a product.
2- It organizes the unit and functional testing efforts of a developer and tester.
3- Provides early feedback on the quality of the code.
4- Helps in tracking test coverage as well as code coverage.
5- Results in easy debugging and reduces chances of errors.

What are cucumber tags? And why do we use them?
Cucumber tags help in filtering the scenarios(manual,automated,smoke). We can tag the scenarios and then run them based on tags.

1- We can add tags to scenarios with the <@> symbol.

What is the purpose of cucumber dry-run?
We use to compile the cucumber feature files and step definitions. If there occur any compilation errors, then it shows them on the console.

And, But

If you have several `Given`'s, `When`'s, or `Then`s, you *could* write:

```
Example: Multiple Givens
  Given one thing
  Given another thing
```

```
  Given yet another thing
  When I open my eyes
  Then I should see something
  Then I shouldn't see something else
```

Or, you could make it read more fluidly by writing:

```
Example: Multiple Givens
  Given one thing
  And another thing
  And yet another thing
  When I open my eyes
  Then I should see something
  But I shouldn't see something else
```

Background

A `Background` allows you to add some context to the scenarios in the feature. It can contain one or more `Given` steps.

A `Background` is run before *each* scenario, but after any Before hooks. In your feature file, put the `Background` before the first `Scenario`.

You can only have one set of `Background` steps per feature. If you need different `Background` steps for different scenarios, you'll need to split them into different feature files.

Data Tables in Cucumber

Most of the people gets confused with Data tables & Scenario outline, but these two works completely differently.

Difference between Scenario Outline & Data Table

Scenario Outline:

- *This uses Example keyword to define the test data for the Scenario*

- *This works for the whole test*
- *Cucumber automatically run the complete test the number of times equal to the number of data in the Test Set*

Data Table:

- *No keyword is used to define the data table*
- *This works only for the single step, below which it is defined*
- *A separate code is need to understand the test data and then it can be run single or multiple times but again just for the single step, not for the complete test*

the *Data Tables* can be used in many ways because it has provided many different methods to use.

To use it we may need to deal with a ***list of maps*** or a ***map of lists***, or list of lists etc

One way is to pass test data using *data table* and handle it with using ***Raw()*** method. When No header is provided and data is smaller

```
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:

```
@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();
}
```

Maps in Data Tables

a little complex scenario where a good amount of data is required to pass in the step. Or what is there are multiple columns of test data is present. *How would you handle it?* The answer is to make a Use of ***Maps in Data Tables***.

In the previous chapter of ***Data Tables in Cucumber***, we pass *Username & Password* without Header, due to which the test was not much readable. What if there will be many columns. The basic funda of BDD test is to make the Test in Business readable format, so that business users can understand it easily.

```
Scenario: Successful Login with Valid Credentials
 Given User is on Home Page
 When User Navigate to LogIn Page
 And User enters Credentials to LogIn
 | Username   | Password |
   | testuser_1 | Test@153 |
 Then Message displayed Login Successfully
```

```
        @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<Map<String,String>> data = usercredentials.asMaps(String.class,String.class);

            driver.findElement(By.id("log")).sendKeys(data.get(0).get("Username"));
            driver.findElement(By.id("pwd")).sendKeys(data.get(0).get("Password"));
            driver.findElement(By.id("login")).click();
        }
```

```
Scenario: Successful Login with Valid Credentials
        Given User is on Home Page
        When User Navigate to LogIn Page
        And User enters Credentials to LogIn
        | Username   | Password |
   | testuser_1 | Test@153 |
   | testuser_2 | Test@154 |
        Then Message displayed Login Successfully
```

```
        @When("^User enters Credentials to LogIn$")
        public void user_enters_testuser_and_Test(DataTable usercredentials) throws Throwable {

                //Write the code to handle Data Table
                for (Map<String, String> data : usercredentials.asMaps(String.class, String.class)) {
                        driver.findElement(By.id("log")).sendKeys(data.get("Username"));
                   driver.findElement(By.id("pwd")).sendKeys(data.get("Password"));
                   driver.findElement(By.id("login")).click();
                        }

        }
```

More on data tables:

https://github.com/cucumber/cucumber/tree/master/datatable

Hooks

Hooks are blocks of code that can run at various points in the Cucumber execution cycle. They are typically used for setup and teardown of the environment before and after each scenario.

Where a hook is defined has no impact on what scenarios or steps it is run for. If you want more fine-grained control, you can use tagged hooks.

You can declare hooks in any class.

Scenario hooks

Scenario hooks run for every scenario.

Before

`Before` hooks run before the first step of each scenario.

Annotated method style:

```
@Before
public void doSomethingBefore() {
}
```

Lambda style:

```
Before(() -> {
});
```

Think twice before you use Before

Whatever happens in a `Before` hook is invisible to people who only read the features. You should consider using a background as a more explicit alternative, especially if the setup should be readable by non-technical people. Only use a `Before` hook for low-level logic such as starting a browser or deleting data from a database.

You can specify an explicit order for hooks if you need to.

Annotated method style:

```
@Before(order = 10)
```

```
public void doSomething(){
    // Do something before each scenario
}
```

Lambda style:

```
Before(10, () -> {
    // Do something before each scenario
});
```

After

`After` hooks run after the last step of each scenario, even when steps are `failed`, `undefined`, `pending`, or `skipped`.

Annotated method style:

```
@After
public void doSomethingAfter(Scenario scenario){
    // Do something after after scenario
}
```

Lambda style:

```
After((Scenario scenario) -> {
});
```

The `scenario` parameter is optional. If you use it, you can inspect the status of the scenario.

For example, you can take a screenshot with WebDriver for failed scenarios and embed them in Cucumber's report.

See the browser automation page for an example on how to do so.

Global Hooks

Cucumber-JVM does not support global hooks.

Running a hook only once

Cucumber-JVM does not support running a hook only once.

AfterConfiguration

Cucumber-JVM does not support `AfterConfiguration` hooks.

Automation Testing With Selenium, Cucumber & TestNG

Three major parts of the cucumber framework i.e. Feature file, Step Definitions, and Test Runner file.

Test Runner file defined the location of step definition and primarily provides all the metadata information required for test execution. It makes use of **'@RunWith()'** annotation from JUnit framework for execution.

Secondarily you would be making use of **'@CucumberOptions'** annotation to define the location of feature files, step definitions location through glue, what reporting integrations to use etc.

'@RunWith()' is required only when running with Junit, not required when running with TestNG.

To work with testing runner class extends "AbstarctTestNGCucumberTests" class

'@RunWith()' is used with a class to make it a (custom) runner class

```
  @RunWith(Suite.class)
   @SuiteClasses({ATest.class, BTest.class, CTest.class})
   public class ABCSuite {
}
```

```
package amazonTest;

import org.junit.runner.RunWith;
import cucumber.api.CucumberOptions;
import cucumber.api.junit.Cucumber;

@RunWith(Cucumber.class)
@CucumberOptions(features = "src/test/Feature"  ,glue={"src/main/stepDefinition"}
      )

public class TestRunner {

}
```

```
@RunWith(Cucumber.class)
//@CucumberOptions(plugin = {"pretty"}, features = { "classpath:cucumber_testng" })
@CucumberOptions(plugin = {"pretty"}, features =
"src/test/resources/com/ct/cucumber_testng")
public class RunCucumberTest {
}
```

Classes annotated with `@RunWith(Cucumber.class)` will run a Cucumber Feature.

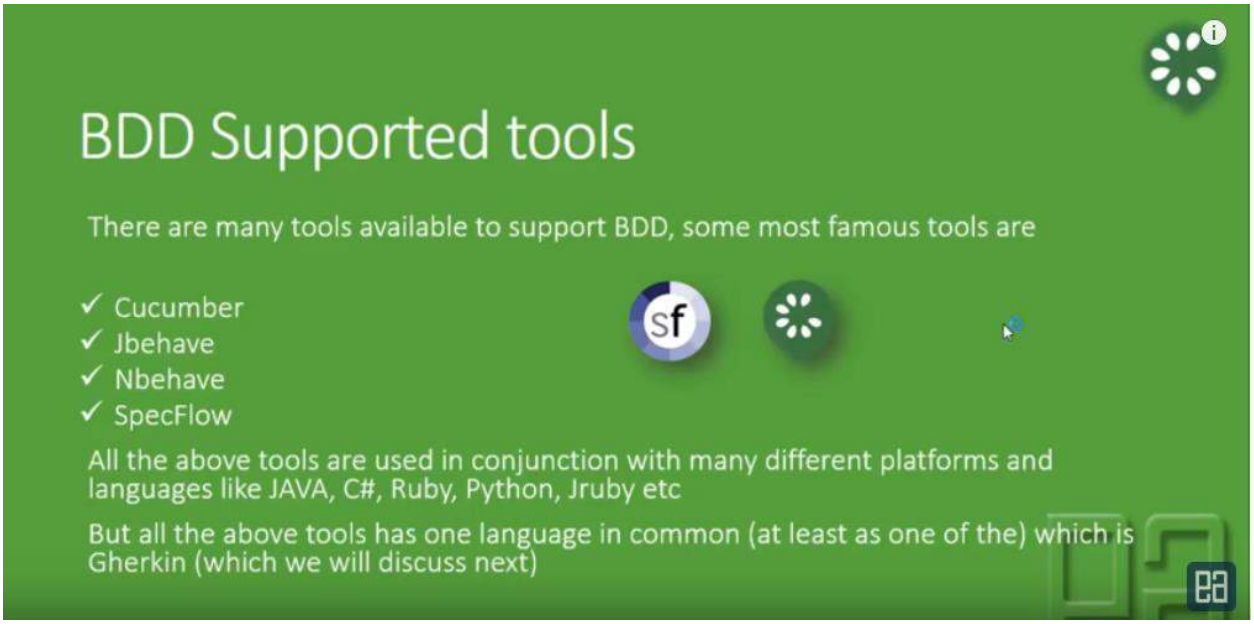
In general, the runner class should be empty without any fields or methods. For example:

```
@RunWith(Cucumber.class)
@CucumberOptions(plugin = "pretty")
public class RunCukesTest {
}
```

Cucumber will look for a `.feature` file on the classpath, using the same resource path as the annotated class

Additional hints can be given to Cucumber by annotating the class with `CucumberOptions`.

Cucumber:

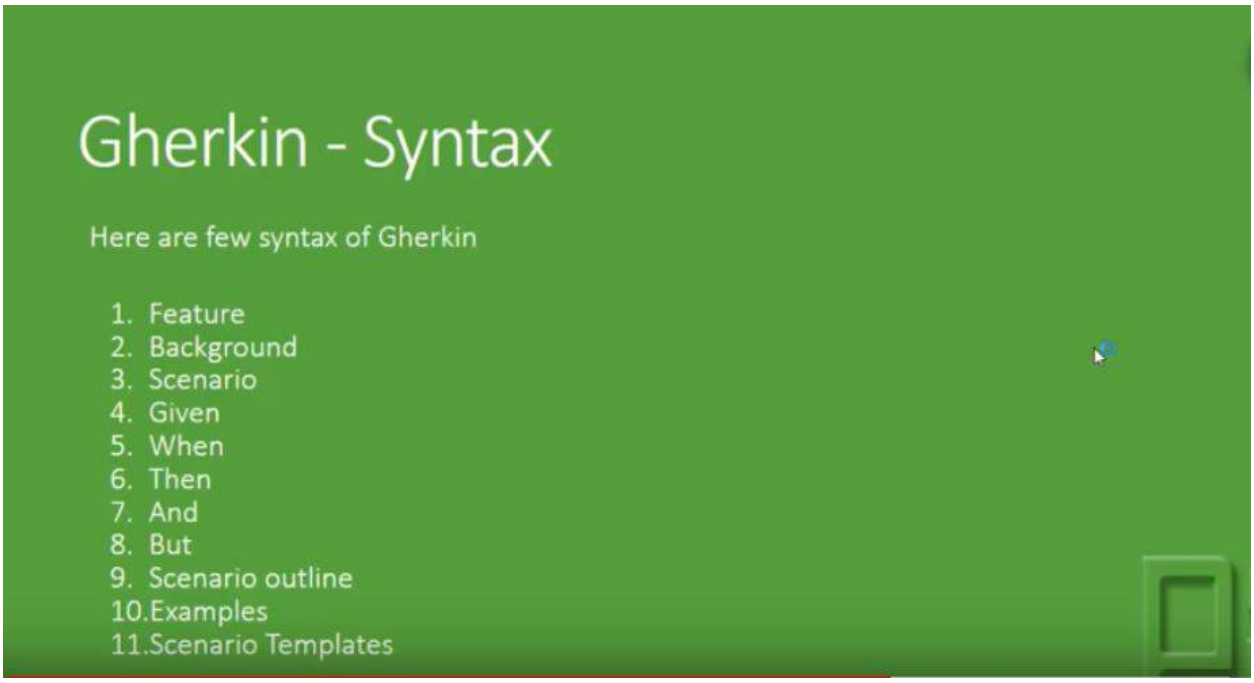


Cucumber-with java, python, ruby etc

Cucumber-jvm: java implementation of

Cucumber js- javascript implementation of cucumber

SpecFlow- C# implementation of cucumber



BDD vs traditional

BDD	Traditional Automation
Its plain text and easy to understand	Its full of code and hard to understand
Its easy to understand by BA/QA/DEV/Automation Test Engineer and all will be in same page	The code are understood only by Automation test engineer (Some times Dev)
Since its plain text format, BDD can be shared even to stakeholders	Impossible
Easy to learn and implement	More knowledge is required while designing

Cucumber-JVM
Java
official

Cucumber.js
Node.js and browsers
official

Cucumber.rb
Ruby, Ruby on Rails
official

Cucumber.ml
OCaml
official

Cucumber.cpp
C++
official

Cucumber-Lua
Lua
official

Android™
Java
official

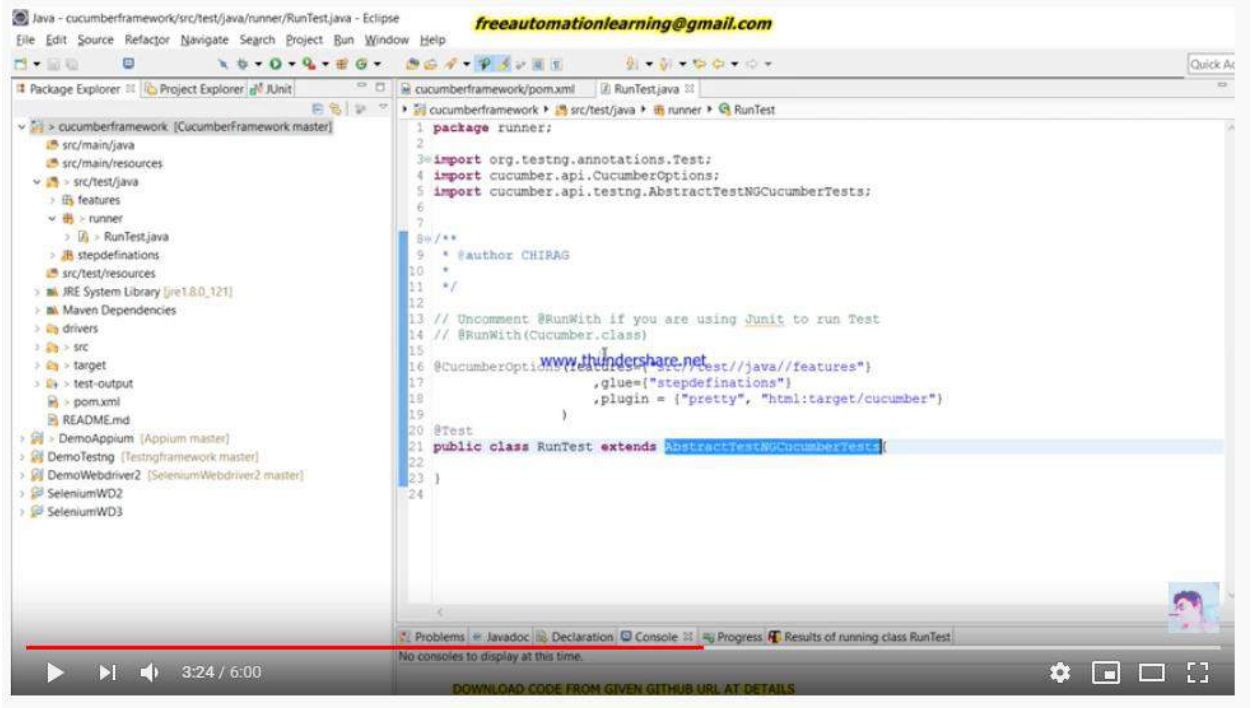
Kotlin
Cucumber-JVM with Kotlin
official

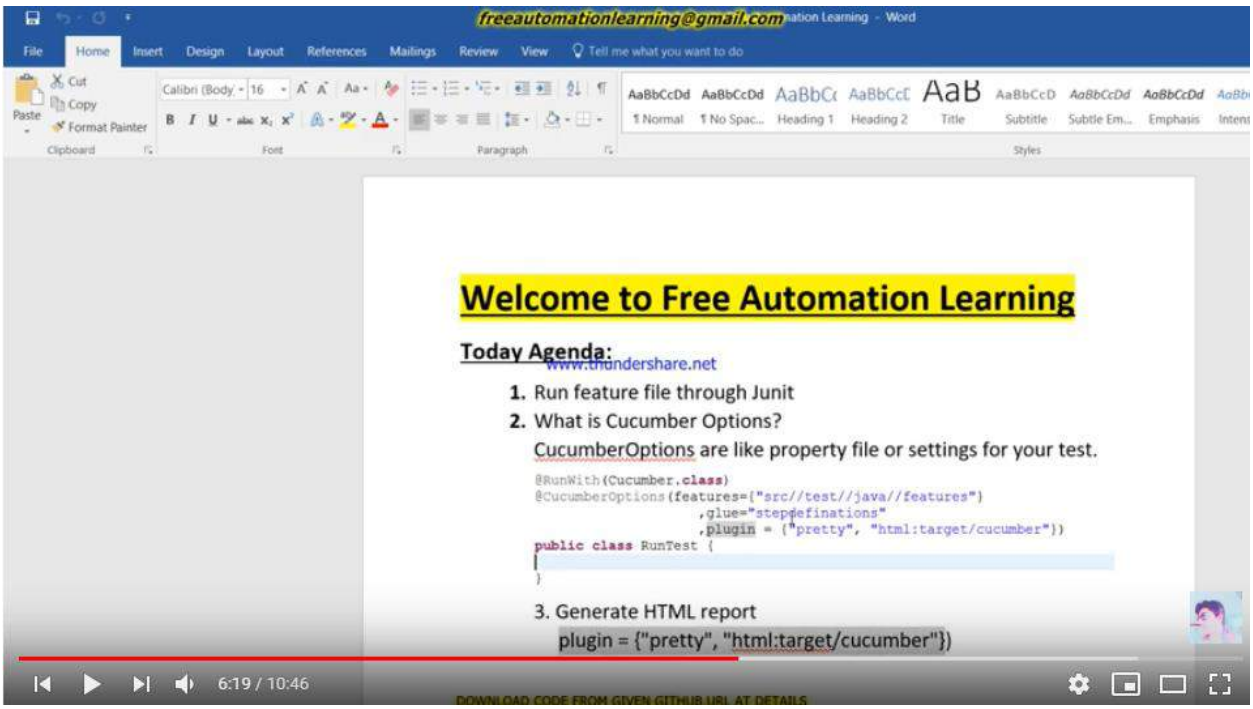
Cucumber-Tcl
Tcl
official

SpecFlow
C#, F#, VB.NET
semi-official

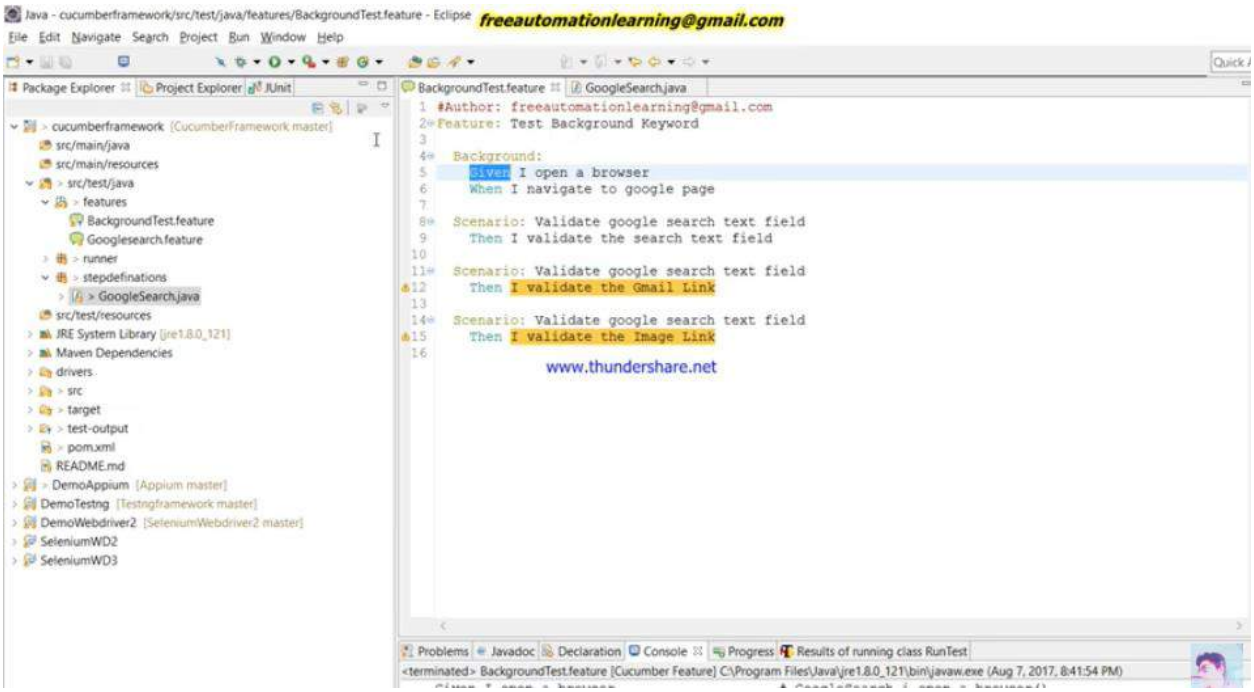
Behat
PHP
semi-official

Behave
Python
semi-official

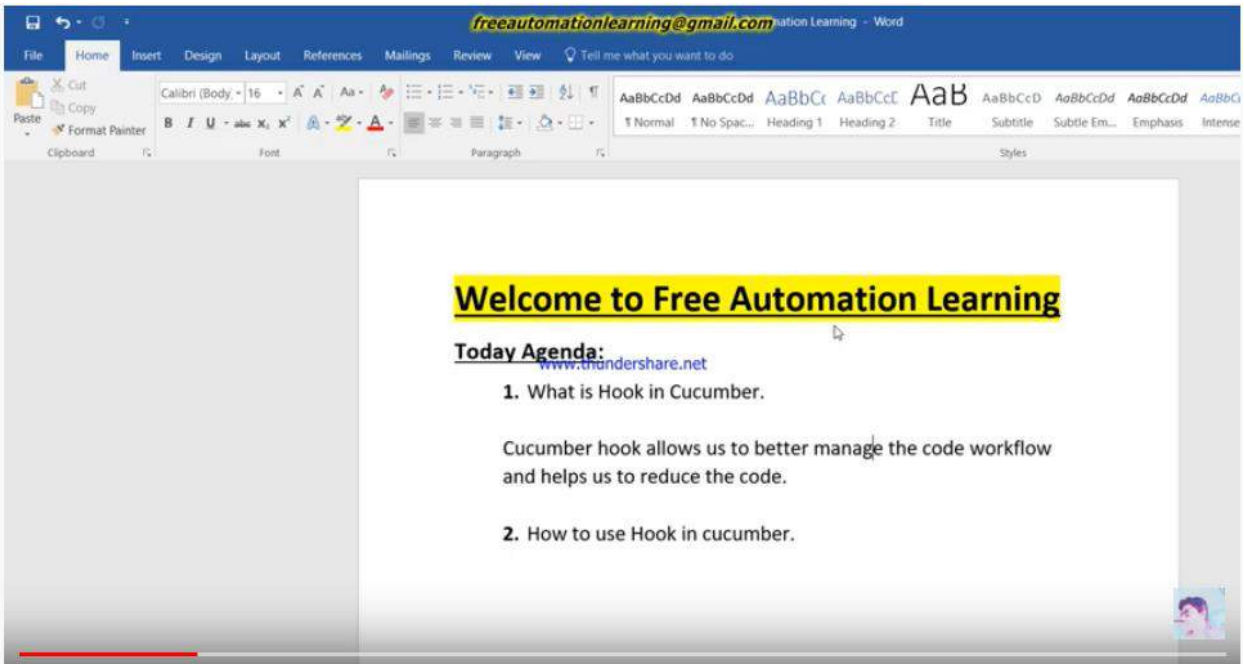


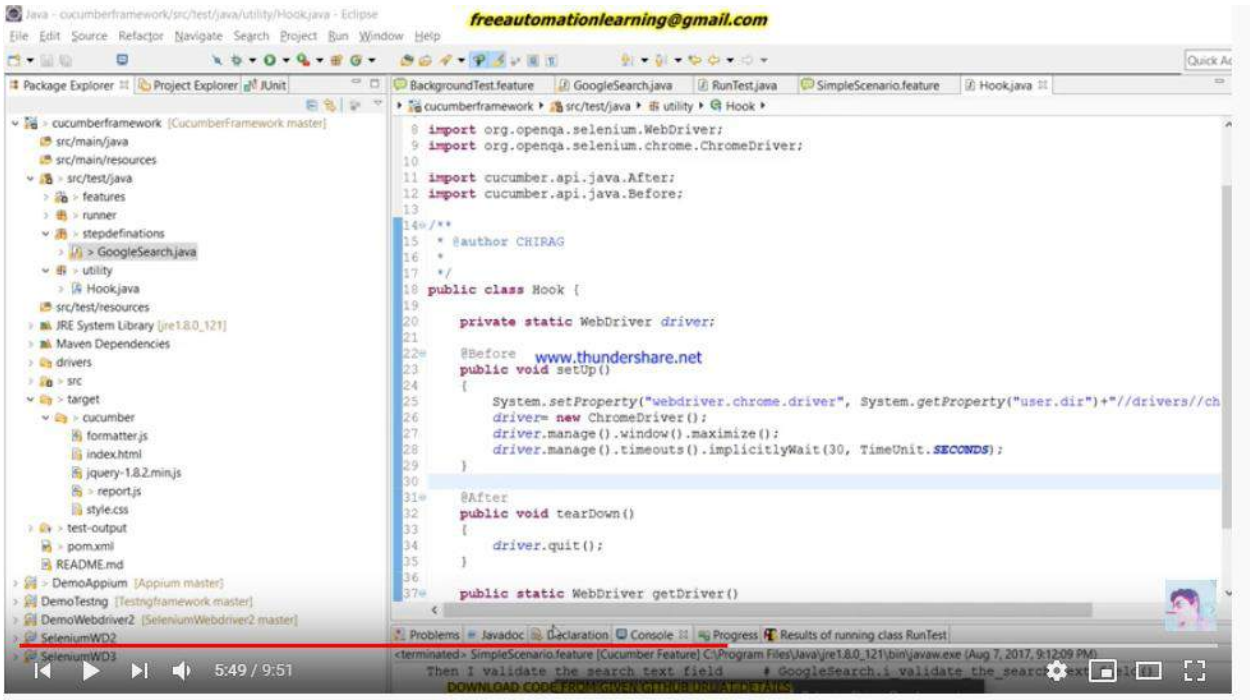


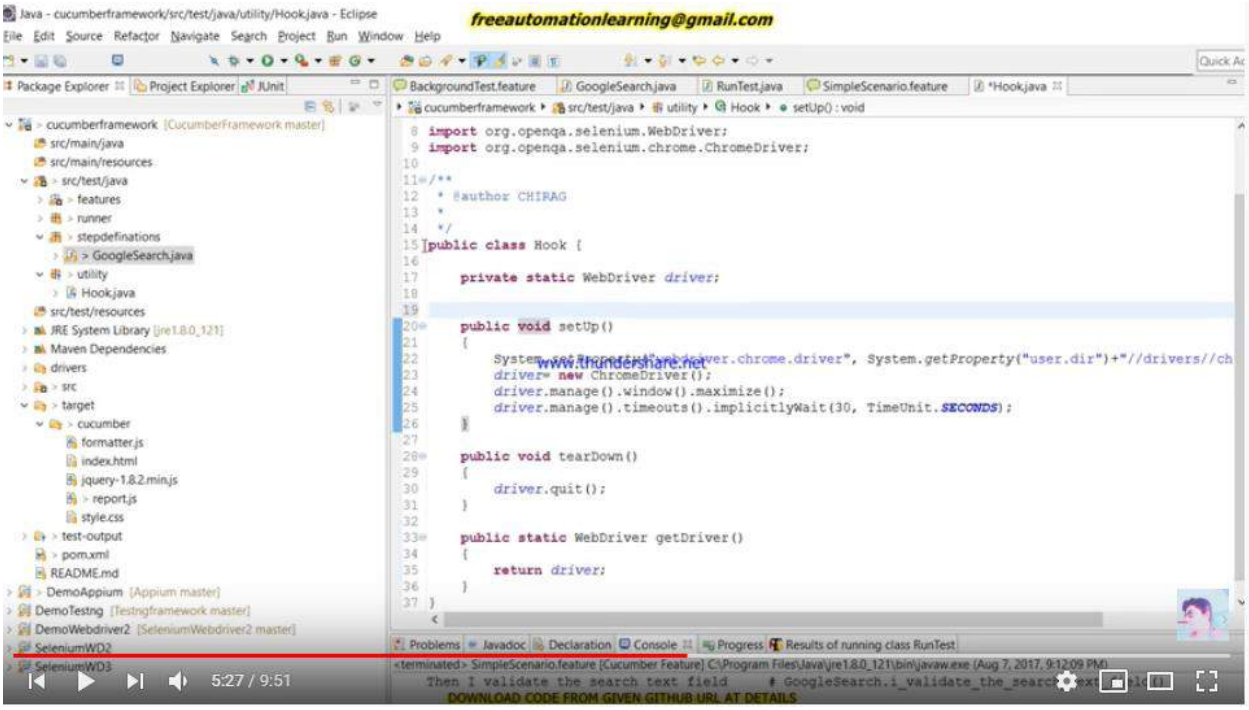
Background keyword:



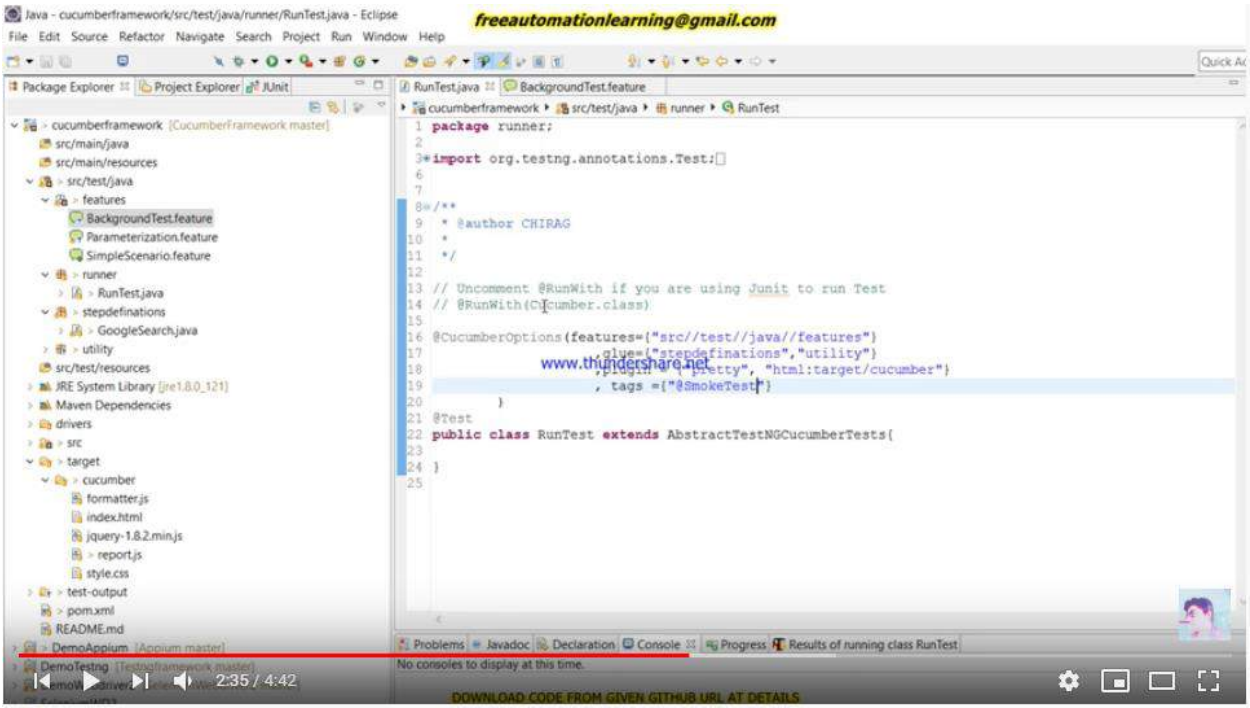
Hook in cucumber:



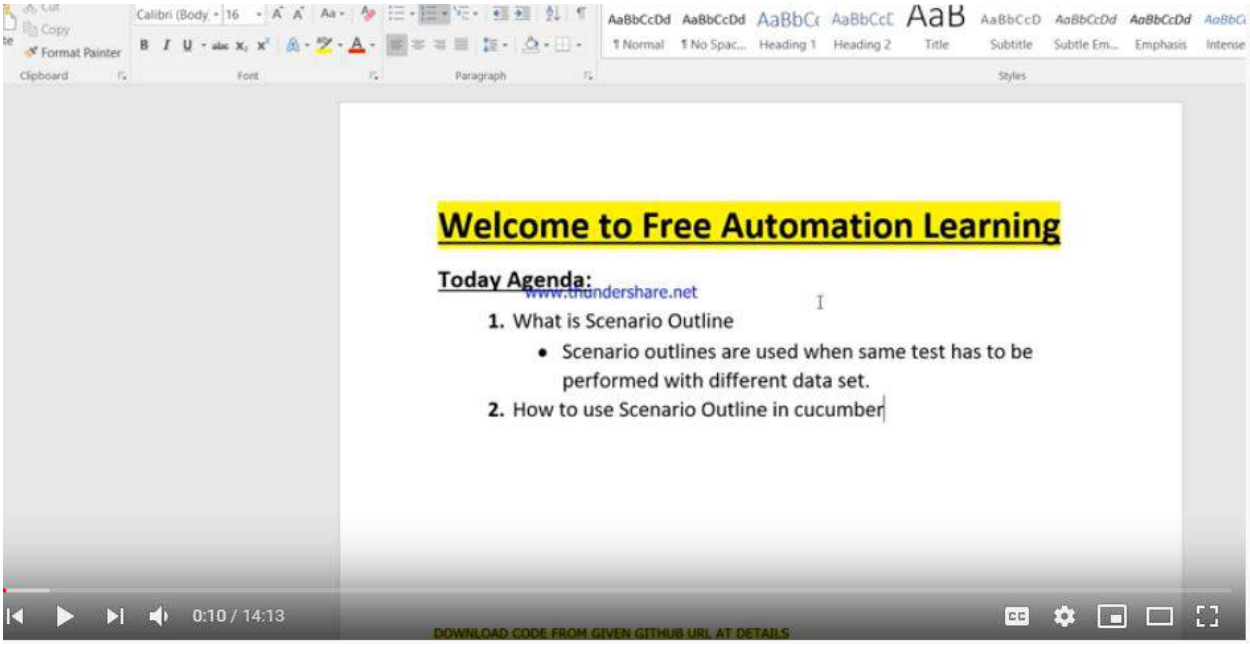


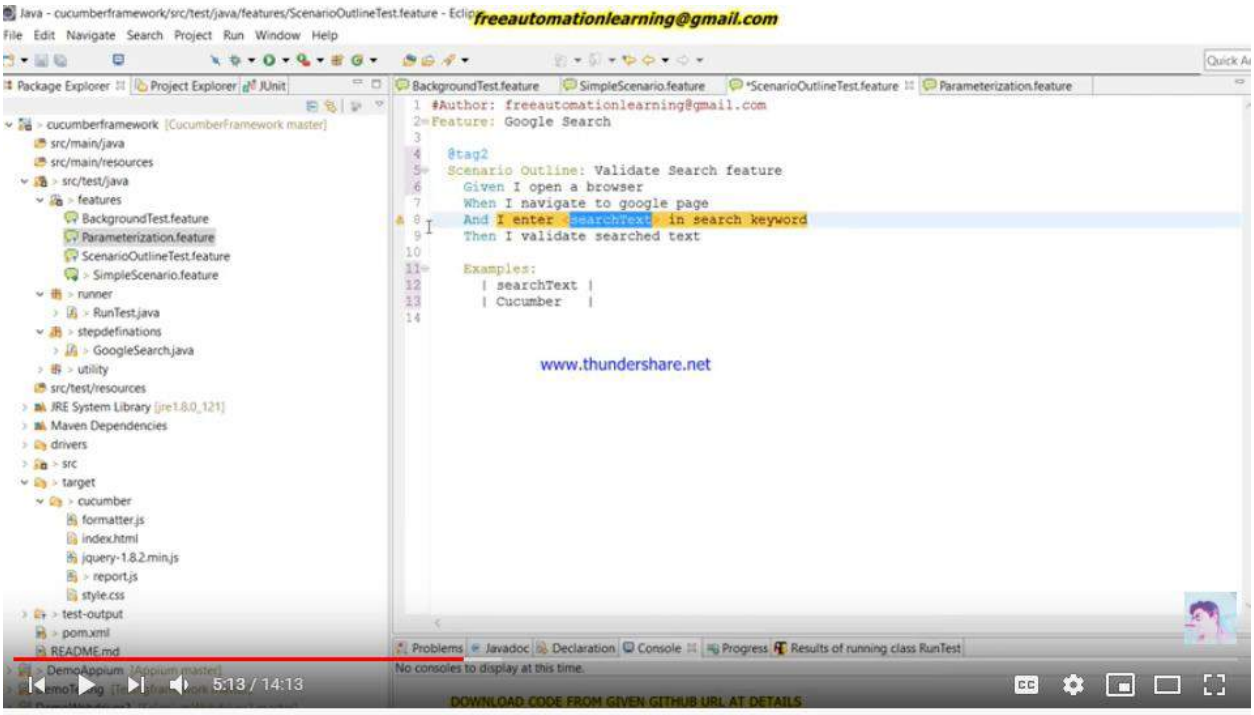


Using tags to run scenarios:

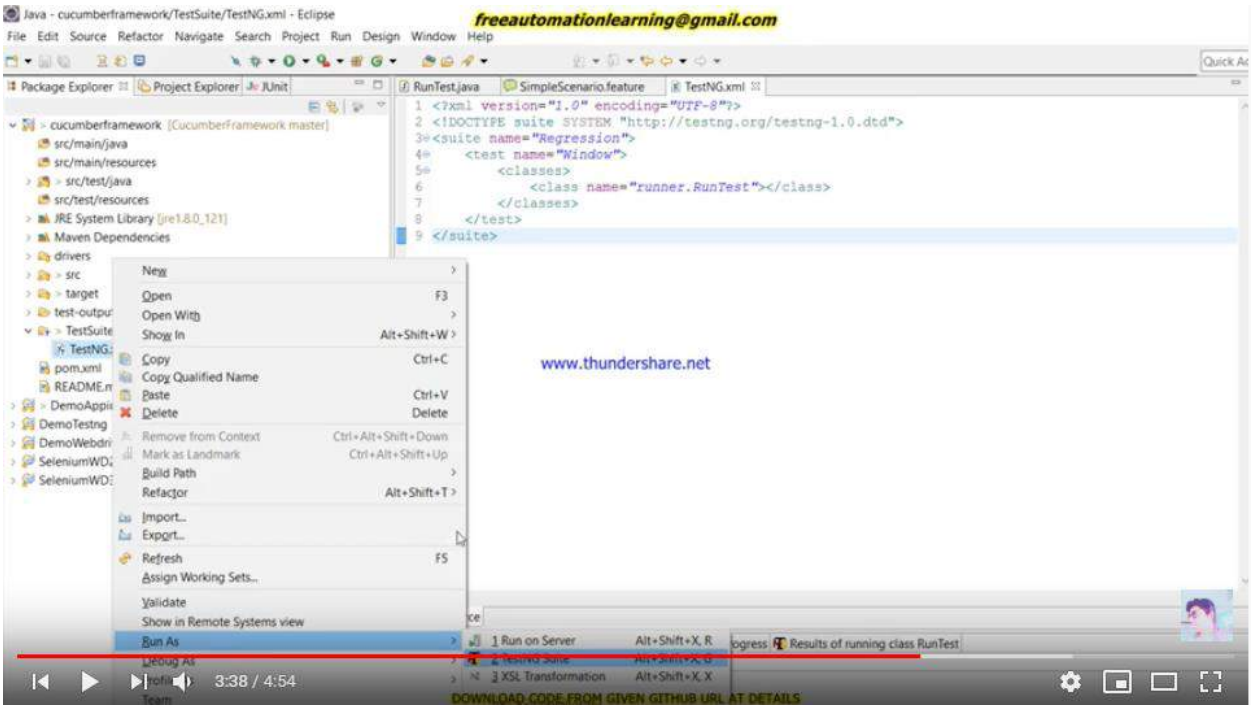


Data driven testing : Scenario outline: Examples keyword has to be used with table.





Running with tesng.xml:



Running testing suite with maven commands or with pom.xml:

Pom.xml-> Run As maven test