

TECHNOLOGY



Coding Bootcamp

TECHNOLOGY



Cucumber

Scenarios in Cucumber



Learning Objectives

By the end of this lesson, you will be able to:

- Explain scenarios in Cucumber testing to write clear and structured tests that accurately validate application behavior
- Comprehend scenario outlines with examples to test multiple data sets and ensure comprehensive test coverage efficiently
- List the types of scenarios to effectively manage and troubleshoot tests, including Undefined, Pending, and Failed scenarios
- Explore the environment variables in Cucumber and their uses to enhance flexibility and adaptability in dynamic test configurations

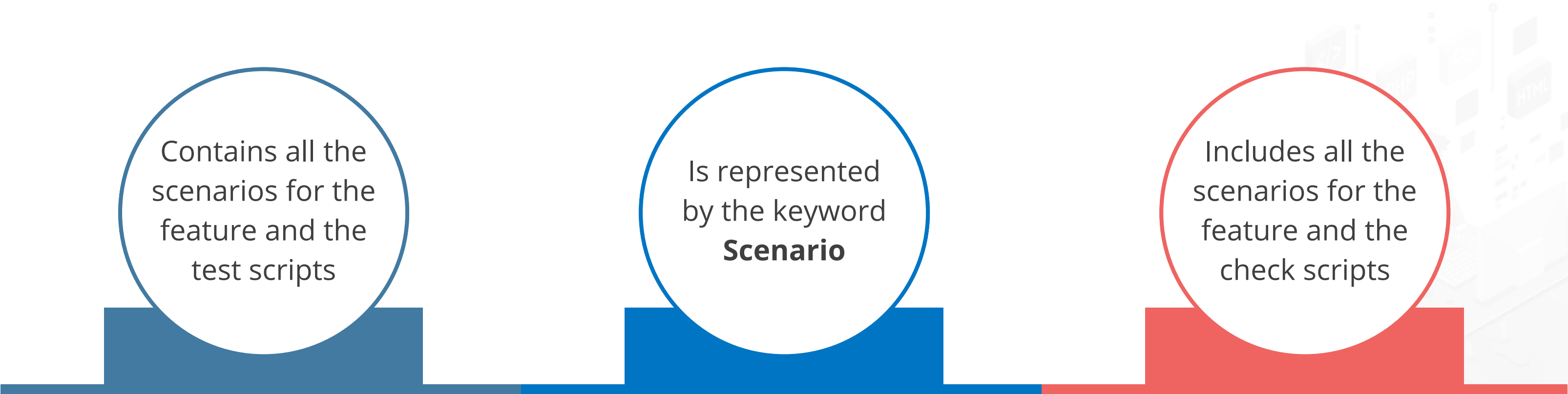


TECHNOLOGY

Scenario

Scenario in Cucumber Testing

Scenarios are fundamental to Cucumber testing as they describe the application's behavior in specific situations. It is also one of Gherkin's primary structures, which:



The diagram consists of three circular nodes arranged horizontally, each sitting on a colored rectangular base. The first node is blue and contains the text 'Contains all the scenarios for the feature and the test scripts'. The second node is blue and contains the text 'Is represented by the keyword **Scenario**'. The third node is red and contains the text 'Includes all the scenarios for the feature and the check scripts'. The bases of the circles are colored to match the circles: blue for the first two and red for the third. The background features a faint, stylized illustration of a city skyline with various building shapes and lines.

Contains all the scenarios for the feature and the test scripts

Is represented by the keyword **Scenario**

Includes all the scenarios for the feature and the check scripts

Scenario: Example

Here's a scenario for checking the feedback feature of a web application:

Scenario: Validate the feedback functionality.

Given: The user reaches xyz.com.

When: The user clicks on **Feedback**, and the Feedback page opens.

And: The user submits a feedback message.

Then: The user's feedback should be displayed on the admin page.

And: The administrator can respond to the user.

Every scenario uses Given, When, And, and Then.

Scenario Outline

Scenario Outline

A scenario outline runs the same scenario multiple times with different sets of values.



Scenario outlines are helpful when performing the same test with various data combinations. In Cucumber, scenario outlines are defined using the *Scenario Outline* keyword, followed by *Examples* tables that list the different data sets. Each row in the table represents a separate test iteration.

A scenario template is another term for a scenario outline.

Scenario Outline: Example

Case: To test the login functionality for multiple customers

Use a scenario outline to execute the login capability multiple times.



Use <username> and <password> in Gherkin sentences.



Scenario Outline: Example

Case: To test the login functionality for multiple customers

Test script



Scenario outline: Successful login

Given: The user navigates to the login page.

When: The user enters <username> in the **Login Window** as UserA.

And: The user enters <password> as PassA.

Then: The user should be successfully logged in.

Scenario Outline: Example

Case: To test login functionality with various types of usernames and passwords

Test script

Feature: Login Functionality

Scenario Outline: Login Functionality

User		Pass
User1		PassA
User2		PassB
User3		PassC
User4		PassD

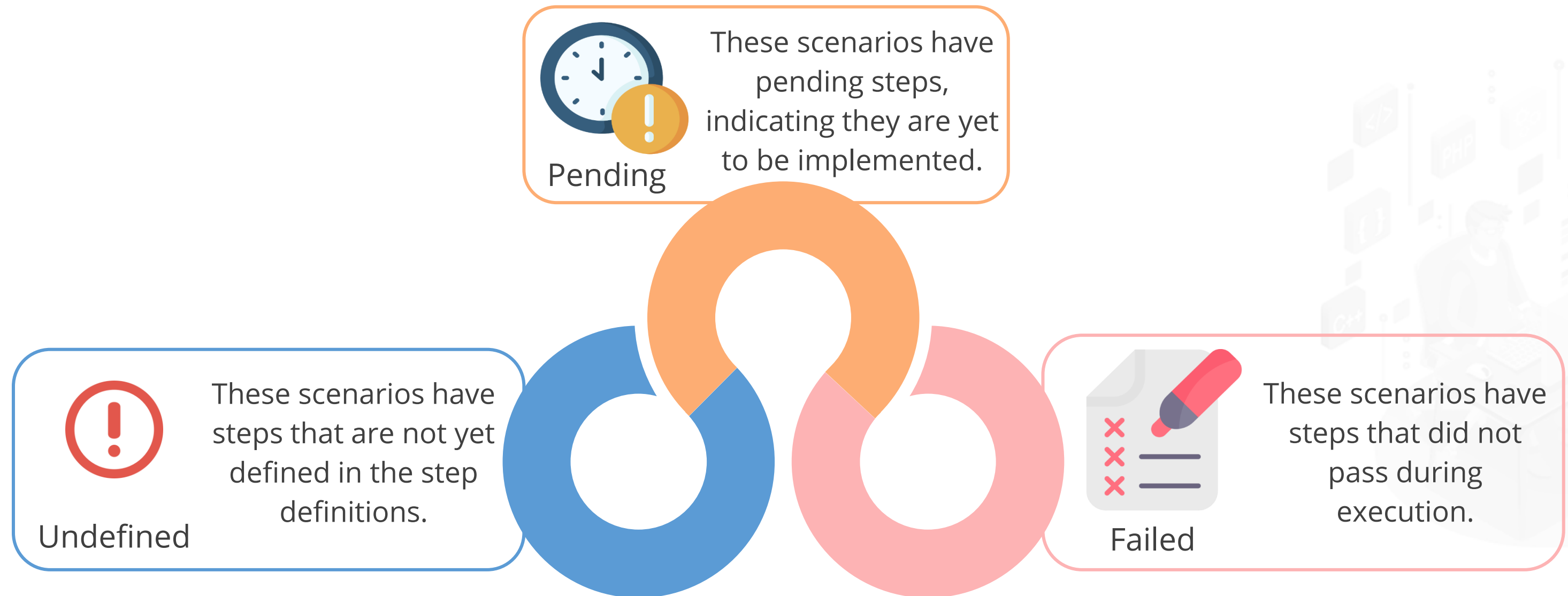
Once the user is executed with a pass, the test will automatically pick the second iteration with another input value.



Types of Scenarios

Types of Scenarios

The following are the different types of scenarios encountered in Cucumber testing:



Undefined Scenario

Undefined scenarios are those that Cucumber cannot execute because they lack step definitions.

hellocucumber.RunCucumberTest

Feature: Is it Friday yet?

#Everyone wants to know when it's Friday

Scenario: Sunday isn't Friday #hellocucumber/is_it_Friday_yet.feature:4

Given: Today is Sunday #null

When: I ask whether it's Friday yet #null

Then: I should be told "Nope" #null

Undefined scenario:

Hellocucumber/is_it_Friday_yet.feature:4 #Sunday isn't Friday

1 Scenario (1 undefined)

3 steps (3 undefined)

0m0.040s

Undefined Scenario

Ensure that the path-to-door definitions (glue direction) are targeted correctly.

Implement missing steps with the snippets below:

```
@Given("today is sunday")
```

```
Public void today_is_Sunday(){
```

```
    //Write code here that turns the phrase above into concrete actions
```

```
    //throw new io.cucumber.java.PendingException();
```

```
}
```

```
@When("I ask whether it's Friday yet")
```

```
Public void i_as_whether_it_s_Friday_yet(){
```

```
    //Write code here that turns the phrase above into concrete actions
```

```
    //throw new io.cucumber.java.PendingException();
```

```
}
```

```
@Then("I should be told {string}")
```

```
Public void i_should_be_told(String string){
```

```
    //Write code here that turns the phrase above into concrete actions
```

```
    //throw new io.cucumber.java.PendingException();
```

```
}
```



Pending Scenario

Pending scenarios are those that have not yet been implemented.

To create a pending step, add a step to one of the step definition files.

```
ruby

Given /^PENDING/ do
  pending
end
```

Use it as:

Scenario: Login with valid credentials

Given PENDING, I have valid credentials.



Failed Scenario

Failed scenarios are those that do not pass during execution.

Step 1: In the Runner File inside the plugin, write rerun: **rerun/failed_scenarios.txt**

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

@RunWith(Cucumber.class)
@CucumberOptions(
    features = {"C:\\User\\ankur.jain\\eclipse-
workspace\\Cucumber_Test\\src\\main\\java\\com\\qa\\FeatureFile\\Login.feature"},
    glue = {"com.qa.stepDefinition"},
    plugin = {"pretty", "html:target/site/cucumber-pretty", "json:target/cucumber.json",
"rerun:rerun/failed_scenario.txt"},
    monochrome = true,
    strict = true,
    dryRun = false
)
public class TestRunner {}
```

Failed Scenario

Step 2: Create another Runner file and name it **ReRunRunner.java**. Inside the features option, add the name of the *failed_scenario.txt* file prefixed with @

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

@RunWith(Cucumber.class)
@CucumberOptions(
    features = {"@rerun/failed_scenario.txt"},
    glue = {"com.qa.stepDefinition"},
    plugin = {"pretty", "html:target/site/cucumber-pretty", "json:target/cucumber.json",
"junit:target/cucumber.xml",
"com.cucumber.listener.ExtentCucumberFormatter:target/html/ExtentReport.html"},
    monochrome = true,
    strict = true,
    dryRun = false
)
public class ReRunRunner {}
```

Failed Scenario

If a scenario fails, a ***failed_scenarios.txt*** file is created in the ***rerun*** folder.



Using the ***ReRunRunner.java*** file, the failed scenarios are rerun in Cucumber-JUnit.

Writing Scenario Using the Gherkin Syntax



Problem Statement:

You have been asked to write a scenario using the Gherkin Syntax.

Outcome:

By completing this task, you will gain practical experience in writing scenarios using Gherkin Syntax.

Note: Refer to the demo document for detailed steps:
01_Write_Scenario_Using_the_Gherkin_Syntax

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed are:

1. Add dependencies to the pom.xml file
2. Create feature files
3. Run Cucumber tests



Environment Variables

Environment Variables

Cucumber uses an environment variable to enable features.



Note

Environment variables cannot be defined globally for security reasons.

Environment Variables

Users of Unix-based operating systems should not define environment variables in *~/.bashrc*, *~/.bash_profile*, *~/.zshrc*, or */etc/profile*.



Note

Windows users should not declare environment variables in the **Control Panel** or using **Setx**.



Refactoring in Cucumber

Refactoring

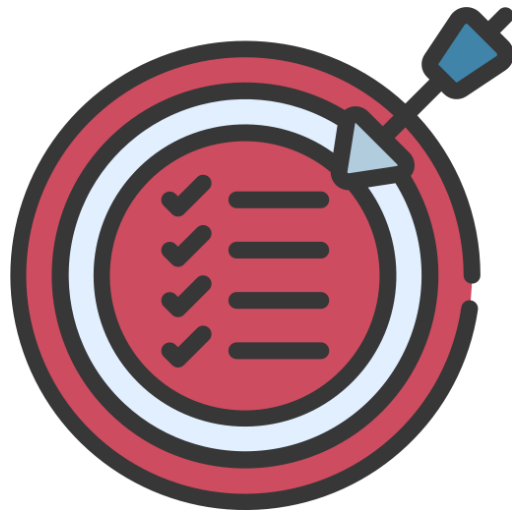
Refactoring involves organizing all the step definition files into groups.

Cucumber evaluates **<any-extension>** ending in * inside the feature folder.



Best Practices to Refactor

Group step definitions



Step definition files are recommended to be grouped based on each domain concept.

The thumb rule is to have one file for each domain object.



Best Practices to Refactor

Write step definitions

Avoid including the steps that are absent in any of the scenarios in the step definition file.

Always implement the required step definitions.

Best Practices to Refactor

Avoid duplication

Avoid writing similar step definitions, as they may create confusion.

It is advisable to execute helper methods for abstracting while documenting the steps.

Avoid Duplication: Example

Instead of writing duplicate step definitions, use helper methods to consolidate similar steps.

- Given I go to the Dashboard page
- Given I check the first page of the website
- Given I get the phone number and email to communicate

Helper method



Given I go to the {} page

The step definitions file:

```
@Given("I go to the {string} page")
public void navigateToWebPage(String page) {
    webpageFactory.openPage(page);
}
```

Writing Step Definitions in Java



Problem Statement:

You have been asked to write Step Definitions in Java.

Outcome:

By completing this task, you will learn how to bridge the gap between Gherkin syntax and executable code in Java

Note: Refer to the demo document for detailed steps:
02_Write_Step_Definition_in_Java

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed are:

1. Write a step definition in Java



Executing Test Cases with Cucumber



Problem Statement:

You have been asked to execute test cases with Cucumber.

Outcome:

By completing this task, you will learn how to integrate Cucumber with your test automation framework.

Note: Refer to the demo document for detailed steps:
[03_Execution_of_Test_Cases_with_Cucumber](#)

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed are:

1. Add dependencies to the pom.xml file
2. Add annotations to the AppTest class
3. Execute the configuration



Key Takeaways

- A scenario outline allows you to run the same scenario multiple times with different data sets. This is useful for efficiently testing various combinations of inputs.
- Undefined scenarios do not have corresponding step definitions in Cucumber. They must be defined before they can be executed.
- Scenarios can be marked as pending to indicate that they are not yet implemented. This helps prioritize which steps need to be completed.
- If a scenario fails, a failed_scenarios.txt file is created in the rerun folder. This allows you to rerun failed scenarios to ensure issues are resolved.
- Refactoring involves organizing step definition files into logical groups. This process helps maintain a clean and manageable codebase, ensuring clarity and readability.



TECHNOLOGY

Thank You