# Cucumber

Location: Bloomington,

Date: Feb 26th 2018,

Author: Venkata Raghavendra Talluri

## What is Cucumber..?

1) In Simple words Cucumber is a tool for writing requirements in Gherkin language and running Automated tests.
2) Cucumber merges specification and test documentation into one cohesive whole.
3) Cucumber is designed to help build bridges between the technical and non-technical members of a software team.

## Cucumber Support

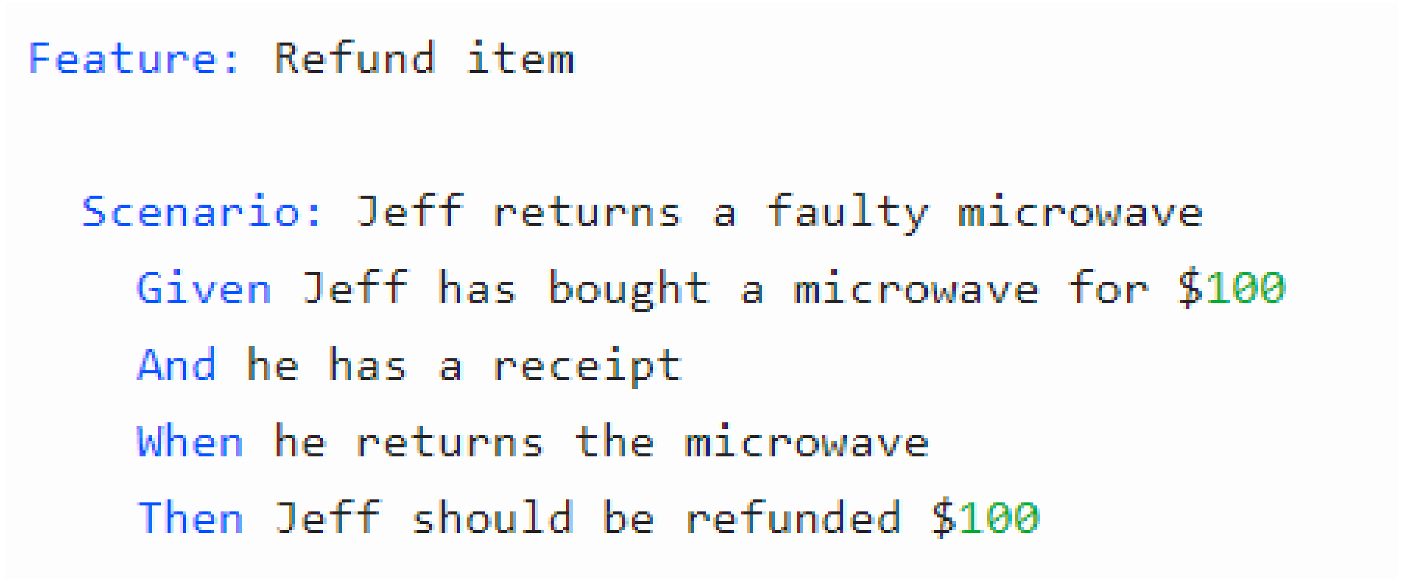
1) Cucumber supports over a dozen different software platforms.
2) Every Cucumber implementation provides the same overall functionality, but they also have their own installation procedure and platform-specific functionality.
3) Cucumber supports below software platforms.
   - Ruby/Jruby
   - Java
   - Groovy
   - JavaScript
   - Clojure
   - Gosu
   - Lua
   - .NET
   - PHP
   - Jython
   - C++
   - Tcl

## Cucumber Support and Framework Integration

1) Currently there are various free Frameworks available in the market for test Automation.
2) Cucumber is directly providing ways to integrate with some of those Frameworks.
3) Cucumber supports integration with below Frameworks
   - Rails (Ruby on Rails)
   - Selenium
   - PicoContainer
   - Spring Framework
   - Watir
   - Serenity (SerenityBDD)

## Gherkin

1) Gherkin is plain-text English (or one of 60+ other languages) with a little extra structure.
2) Gherkin is designed to be easy to learn by non-programmers, yet structured enough to allow concise description of examples to illustrate business rules in most real-world domains.
3) Here is a sample Gherkin document:



```
Feature: Refund item

  Scenario: Jeff returns a faulty microwave
    Given Jeff has bought a microwave for $100
    And he has a receipt
    When he returns the microwave
    Then Jeff should be refunded $100
```

## Gherkin

1) In Gherkin, each line that isn't blank has to start with a Gherkin *keyword*, followed by any text you like. The main keywords are:
   - Feature
   - Scenario
   - Given, When, Then, And, But (Steps)
   - Scenario Outline
   - Examples

2) There are few extra keywords.
   - """ (Doc Strings)
   - | (Data Tables)
   - @ (Tags)
   - # (Comments)

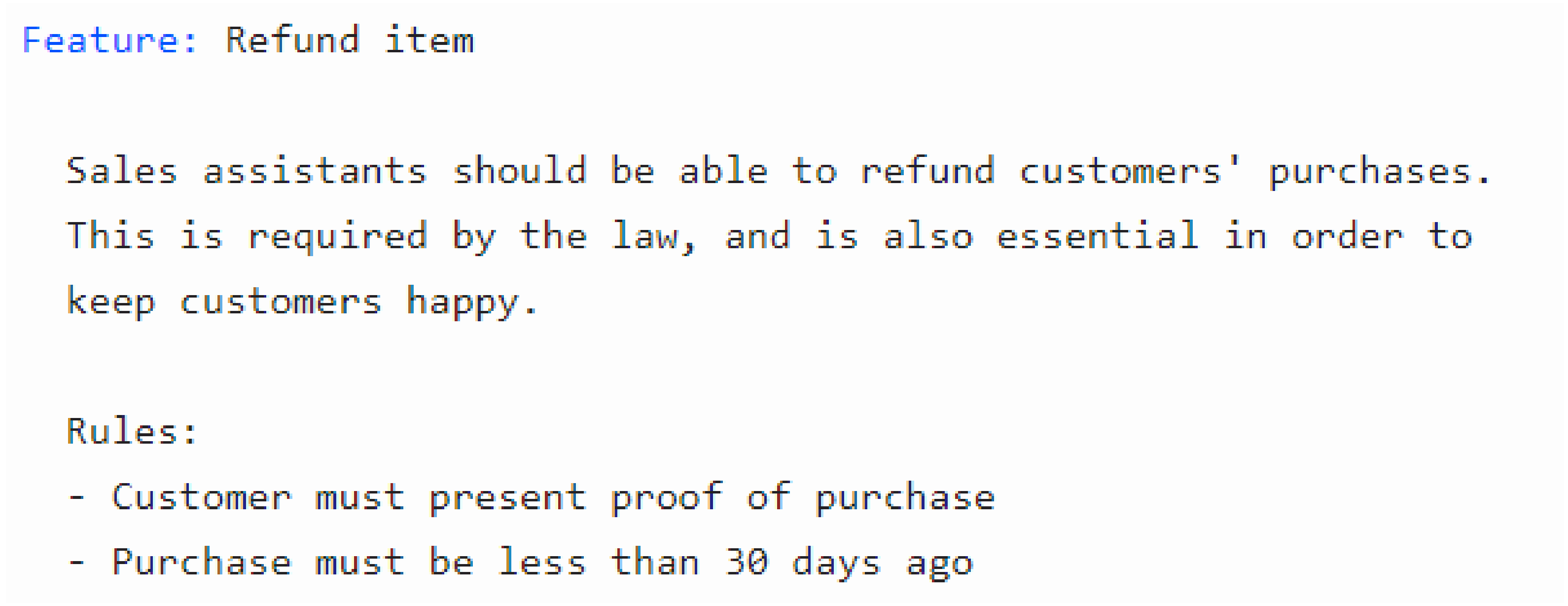
# Gherkin

## Feature

1) A .feature file is supposed to describe a single feature of the system, or a particular aspect of a feature.
2) It's just a way to provide a high-level description of a software feature, and to group related scenarios.
3) A feature has three basic elements
   - the Feature: keyword
   - a *name* (on the same line) and
   - an optional (but highly recommended) *description* that can span multiple lines.
4) Cucumber does not care about the name or the description.
5) The purpose is simply to provide a place where you can document important aspects of the feature.
6) Such as a brief explanation and a list of business rules (general acceptance criteria).

# Gherkin

## Feature Example With Description



```
Feature: Refund item

  Sales assistants should be able to refund customers' purchases.
  This is required by the law, and is also essential in order to
  keep customers happy.

  Rules:
  - Customer must present proof of purchase
  - Purchase must be less than 30 days ago
```

1) In addition to a name and a description, features contain a list of Scenarios or Scenario Outlines, and an optional Background.

2) Some parts of Gherkin documents do not have to start with a keyword.

3) On the lines following a Feature, Scenario, Scenario Outline or Examples you can write anything you like, as long as no line starts with a key a keyword.

## Gherkin

### Scenario

1) A scenario is a concrete example that illustrates a business rule. It consists of a list of steps.
2) You can have as many steps as you like, but BDD recommend you keep the number at 3-5 per scenario.
3) If they become longer than that they lose their expressive power as specification and documentation.
4) In addition to being a specification and documentation, a scenario is also a *test*.
5) As a whole, your scenarios are an *executable specification* of the system.
6) Scenarios follow the same pattern:
   - Describe an initial context
   - Describe an event
   - Describe an expected outcome

# Gherkin

## Steps

1) A step typically starts with Given, When or Then.
2) If there are multiple Given or When steps underneath each other, you can use And or But.
3) Cucumber does not differentiate between the keywords, but choosing the right one is important for the readability of the scenario as a whole.

# Gherkin

## Given

1) Given steps are used to describe the initial context of the system → the *scene* of the scenario.
2) It is typically something that happened in the *past*.
3) When Cucumber executes a Given step it will configure the system to be in a well-defined state, such as creating and configuring objects or adding data to the test database.
4) It's ok to have several Given steps (just use And or But for number 2 and upwards to make it more readable).

# Gherkin

## When

1) “When” steps are used to describe an event, or an *action*.
2) This can be a person interacting with the system, or it can be an event triggered by another system.
3) It's strongly recommended you only have a single ”When” step per scenario.
4) If you feel compelled to add more it's usually a sign that you should split the scenario up in multiple scenarios.
5) In case if you think adding another “When” step is required then you can use “And” or “But” Steps after “When”.

# Gherkin

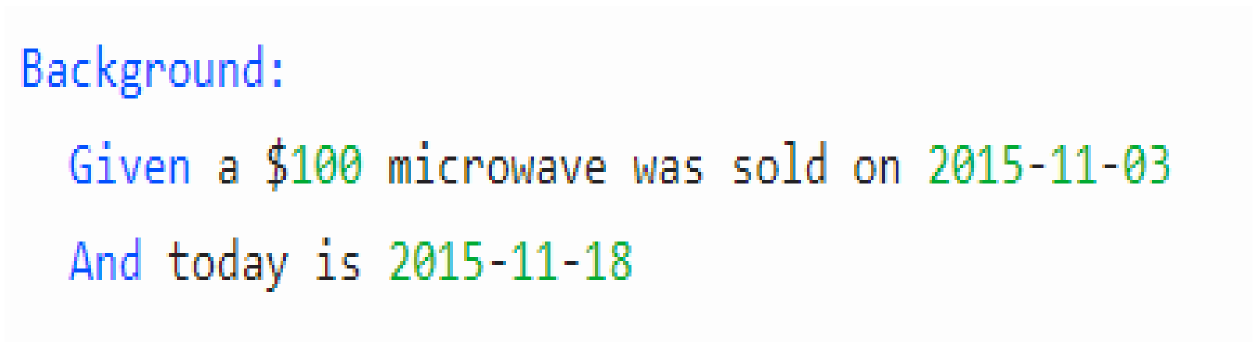
## Then

1) “Then” steps are used to describe an *expected* outcome, or result.
2) The step definition of a “Then” step should use an assertion to compare the actual outcome (what the system actually does) to the expected outcome (what the step says the system is supposed to do).
3) It's strongly recommended you only have a single ”Then” step per scenario.
4) In case if you have multiple assertions in the same scenario, you can use “And” or “But” Steps after “Then”.

# Gherkin

## Background

1) Occasionally you'll find yourself repeating the same Given steps in all of the scenarios in a feature file. Since it is repeated in every scenario it is an indication that those steps are not *essential* to describe the scenarios, they are *incidental details*.
2) You can literally move such Given steps to the background by grouping them under a Background section before the first scenario:

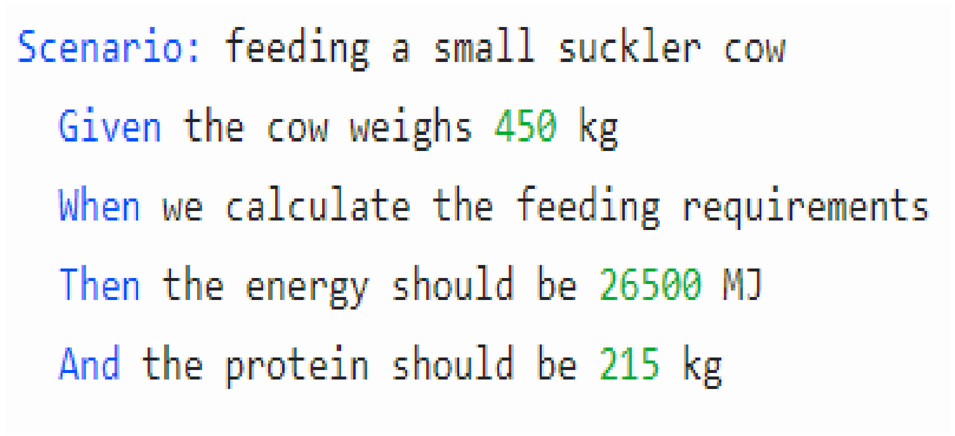


```
Background:
  Given a $100 microwave was sold on 2015-11-03
  And today is 2015-11-18
```

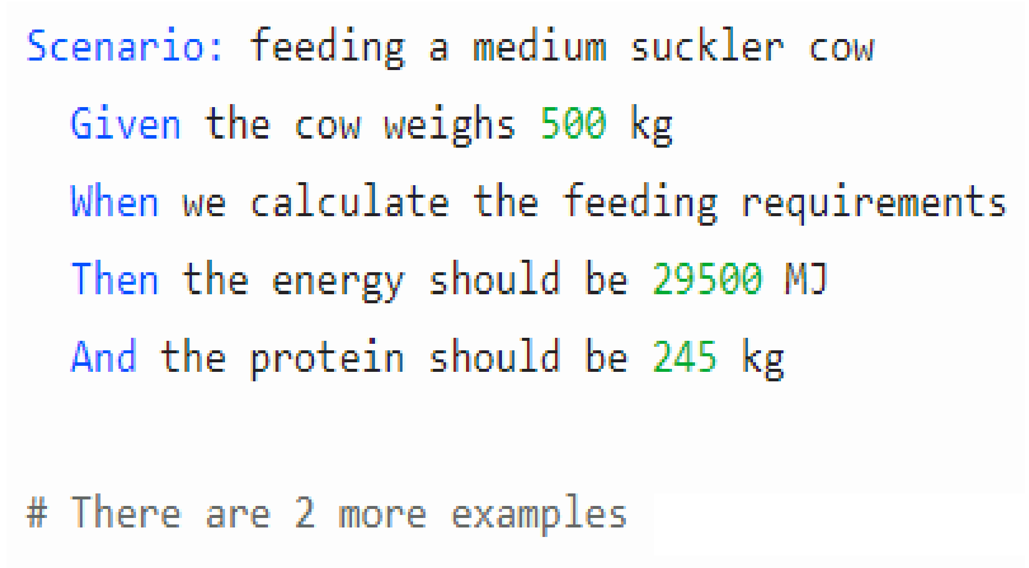
## Gherkin

### Scenario Outline

1) When you have a complex business rule with severable variable inputs or outputs you might end up creating several scenarios that only differ by their values.



```
Scenario: feeding a small suckler cow
  Given the cow weighs 450 kg
  When we calculate the feeding requirements
  Then the energy should be 26500 MJ
  And the protein should be 215 kg
```



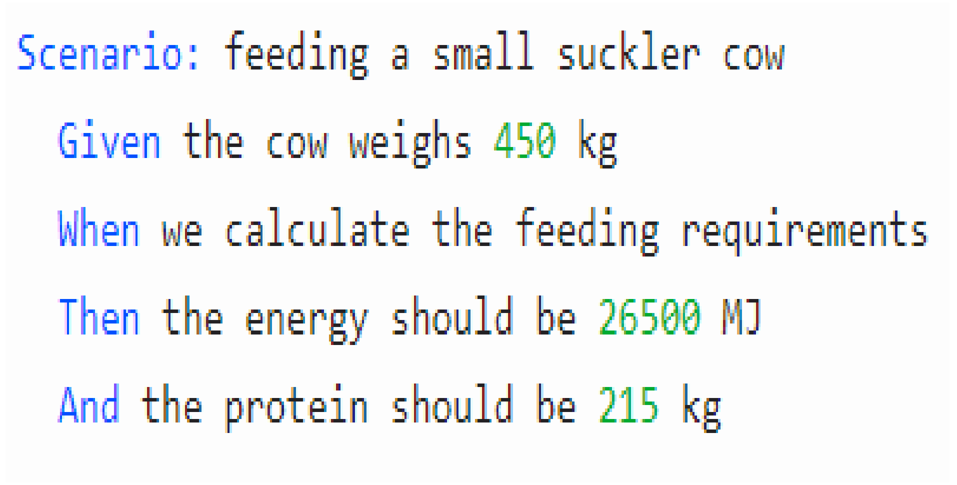
```
Scenario: feeding a medium suckler cow
  Given the cow weighs 500 kg
  When we calculate the feeding requirements
  Then the energy should be 29500 MJ
  And the protein should be 245 kg

# There are 2 more examples
```

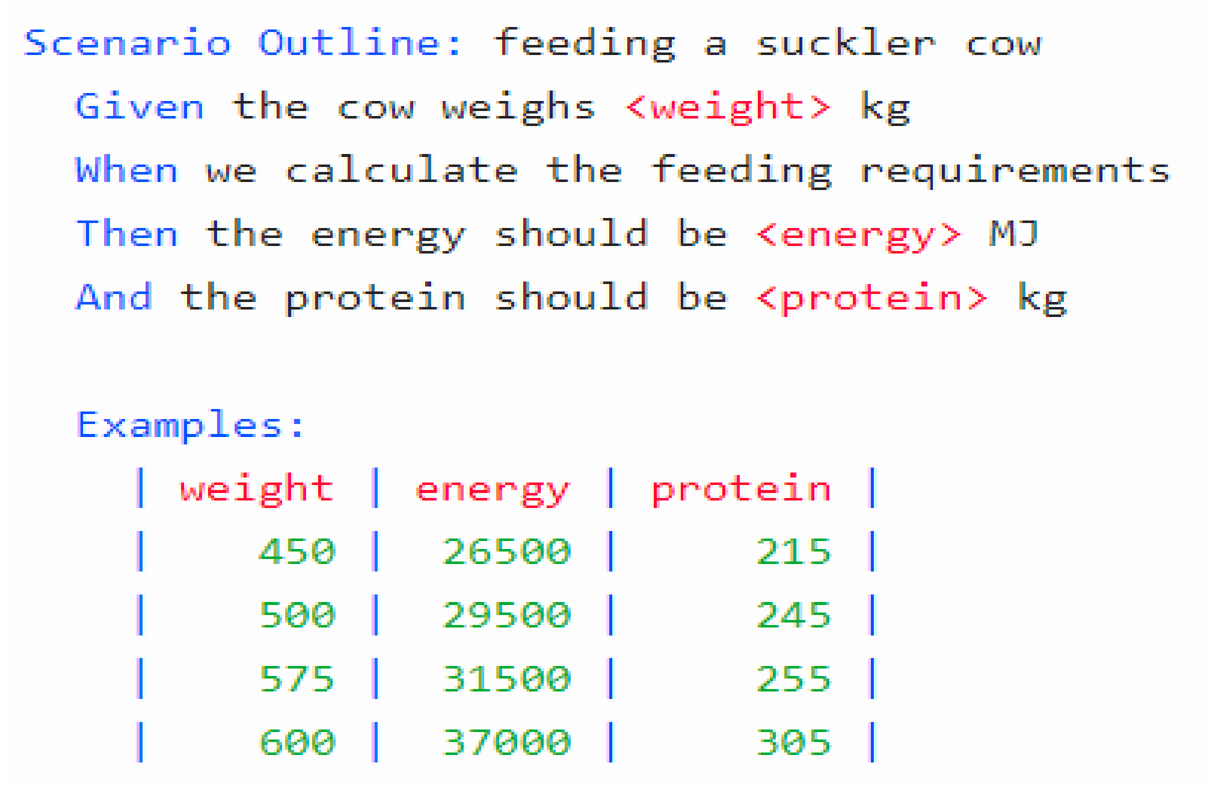
# Gherkin

## Scenario Outline

1) If there are many examples, this becomes tedious. We can simplify it with a Scenario Outline:



```
Scenario: feeding a small suckler cow
  Given the cow weighs 450 kg
  When we calculate the feeding requirements
  Then the energy should be 26500 MJ
  And the protein should be 215 kg
```



```
Scenario Outline: feeding a suckler cow
  Given the cow weighs <weight> kg
  When we calculate the feeding requirements
  Then the energy should be <energy> MJ
  And the protein should be <protein> kg

  Examples:
    | weight | energy | protein |
    |    450 |  26500 |     215 |
    |    500 |  29500 |     245 |
    |    575 |  31500 |     255 |
    |    600 |  37000 |     305 |
```

2) This is much easier to read.

3) Variables in the Scenario Outline steps are marked up with < and >.

# Gherkin

## Examples

1) A Scenario Outline section is always followed by one or more Examples sections, which are a container for a table.
2) The table must have a header row corresponding to the variables in the Scenario Outline steps.
3) Each of the rows below will create a new Scenario, filling in the variable values.

## Gherkin

### Scenario Outlines and UI

1) Automating Scenario Outlines using UI automation such as Selenium WebDriver is considered a bad practice.
2) The only good reason to use Scenario Outlines is to validate the implementation of business rule that behaves differently based on several input parameters.
3) Validating a business rule through a UI is slow, and when there is a failure it is difficult to pinpoint where the error is.
4) The automation code for Scenario Outlines should communicate directly with the business rule implementation, going through as few layers as possible. This is fast, and errors become easy to diagnose fix.

# Gherkin

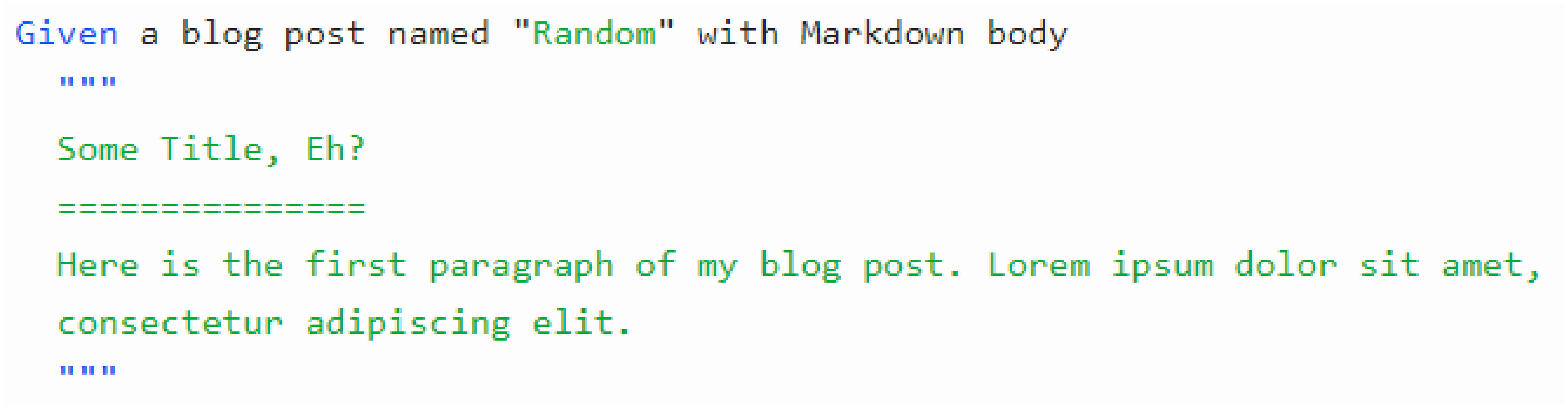
## Step Arguments

1) In some cases you might want to pass a larger chunk of text or a table of data to a step → something that doesn't fit on a single line.
2) For this purpose Gherkin has “Doc Strings” and “Data Tables”.

# Gherkin

## Doc Strings

1) Doc Strings are handy for passing a larger piece of text to a step definition. The syntax is inspired from Python's Docstring syntax.
2) The text should be offset by delimiters consisting of three double-quote marks on lines of their own:



```
Given a blog post named "Random" with Markdown body
  """
  Some Title, Eh?
  ===============
  Here is the first paragraph of my blog post. Lorem ipsum dolor sit amet,
  consectetur adipiscing elit.
  """
```

## Gherkin

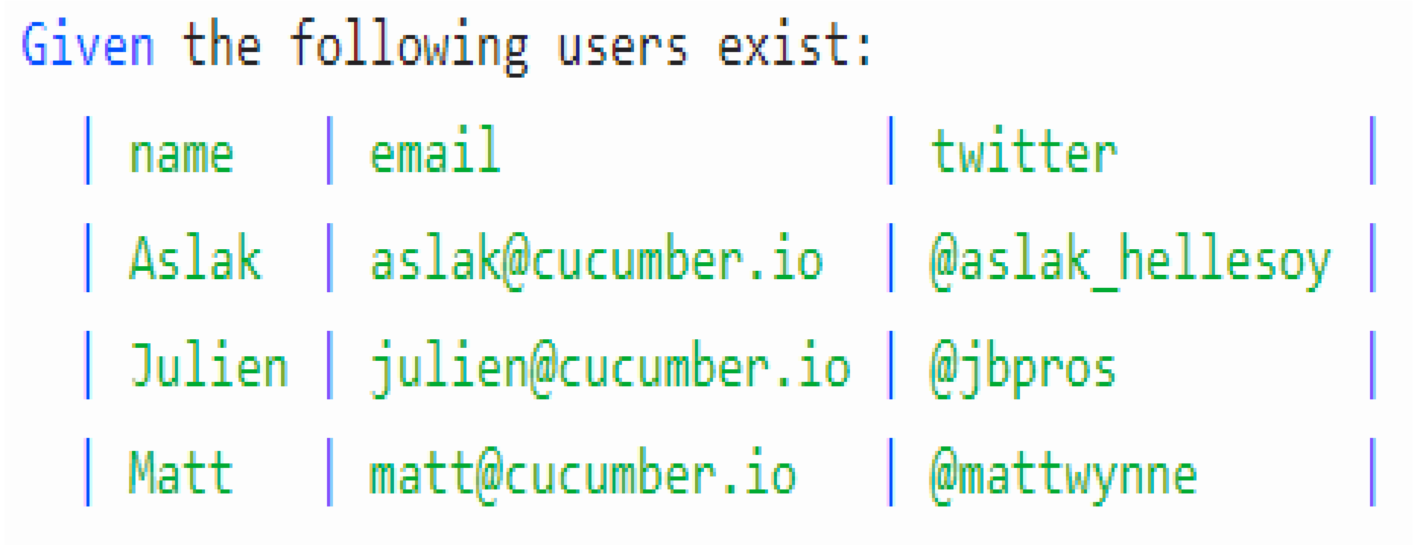
### Doc Strings

1) In your Step Definition, there’s no need to find this text and match it in your pattern. It will automatically be passed as the last parameter in the step definition.
2) Indentation of the opening """ is unimportant, although common practice is two spaces in from the enclosing step.
3) The indentation inside the triple quotes, however, *is* significant.
4) Each line of the Doc String will be de-indented according to the opening """.
5) Indentation beyond the column of the opening """ will therefore be preserved.

## Gherkin

### Data Tables

1) Data Tables are handy for passing a list of values to a step definition:



```
Given the following users exist:
  | name   | email              | twitter         |
  | Aslak  | aslak@cucumber.io  | @aslak_hellesoy |
  | Julien | julien@cucumber.io | @jbpros         |
  | Matt   | matt@cucumber.io   | @mattwynne      |
```

2) Just like Doc Strings, Data Tables will be passed to the Step Definition as the last argument.
3) The type of this argument will be “DataTable”. See the API docs for more details about how to access the rows and cells.

# Gherkin

## Tags

1) Tags are a way to group Scenarios.
2) They are @-prefixed strings and you can place as many tags as you like above Feature, Scenario, Scenario Outline or Examples keywords.
3) Space character are invalid in tags and may separate them.
4) Tags are inherited from parent elements.
5) For example, if you place a tag above a Feature, all scenarios in that feature will get that tag.
6) Similarly, if you place a tag above a Scenario Outline or Examples keyword, all scenarios derived from examples rows will inherit the tags.
7) You can tell Cucumber to only run scenarios with certain tags, or to exclude scenarios with certain tags.
8) Cucumber can perform different operations before and after each scenario based on what tags are present on a scenario.

# Gherkin

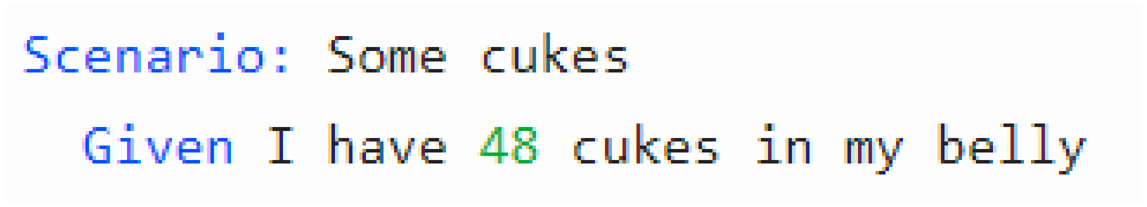
## Comments

1) Gherkin provides lots of places to document your features and scenarios.
2) The preferred place is descriptions. Choosing good names is also useful.
3) If none of these places suit you, you can start a line with a # to tell Cucumber that the remainder of the line is a comment, and shouldn't be executed.

## Cucumber

### Step Definitions

1) Cucumber doesn't know how to execute your scenarios out-of-the-box.
2) It needs *Step Definitions* to translate plain text Gherkin steps into actions that will interact with the system.
3) When Cucumber executes a Step in a Scenario it will look for a matching Step Definition to execute.
4) A Step Definition is a small piece of code with a pattern attached to it.
5) The pattern is used to link the step definition to all the matching Steps, and the code is what Cucumber will execute when it sees a Gherkin Step.
6) To understand how Step Definitions work, consider the following Scenario:



```
Scenario: Some cukes
  Given I have 48 cukes in my belly
```

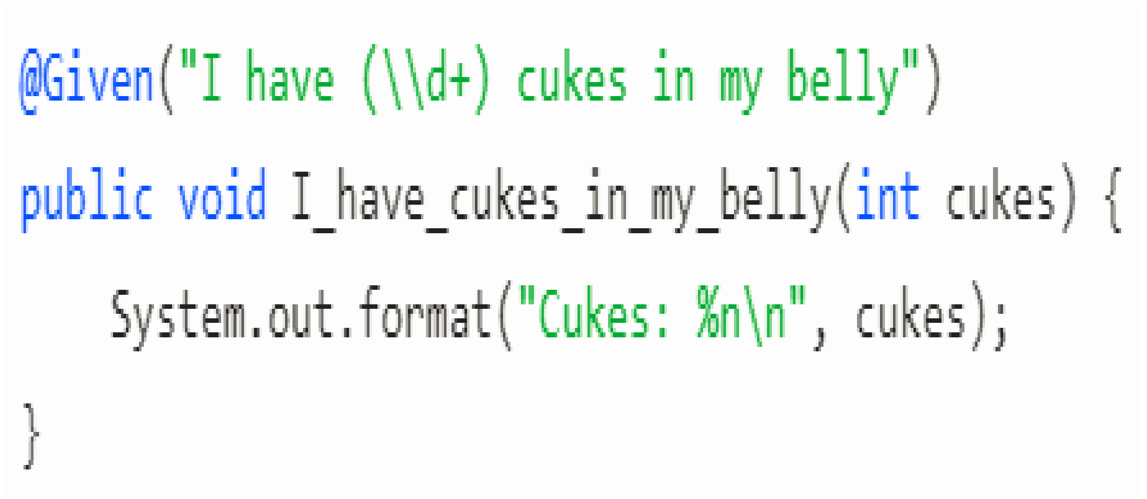
# Cucumber

## Step Definitions

1) The I have 48 cukes in my belly part of the step (the text following the Given keyword) will match the Step Definition below:

2) When Cucumber matches a Step against a pattern in a Step Definition, it passes the value of all the capture groups to the Step Definition's arguments.

3) Capture groups are strings (even when they match digits like \d+).



```
@Given("I have (\\d+) cukes in my belly")
public void I_have_cukes_in_my_belly(int cukes) {
    System.out.format("Cukes: %n\n", cukes);
}
```

4) For statically typed languages, Cucumber will automatically transform those strings into the appropriate type.

5) For dynamically typed languages, no transformation happens by default, as there is no type information.

6) Cucumber does not differentiate between the five step keywords Given, When, Then, And and But.

# Cucumber

## Reports

1) Cucumber can report results in several different formats, using *formatter plugins*. The available formatters plugins are:
   - **Pretty** - Prints the Gherkin source to STDOUT along with additional colours and stack traces for errors.
   - **HTML** - Generates a HTML file, suitable for publishing.
   - **JSON** - Generates a JSON file, suitable for post-processing to generate custom reports.
   - **Progress** - This report prints results to STDOUT, one character at a time.
   - **Usage** - Prints statistics to STDOUT. Programmers may find it useful to find slow or unused Step Definitions.
   - **Junit** - Generates XML files just like Apache Ant's junitreport task. This XML format is understood by most Continuous Integration servers, who will use it to generate visual reports.
   - **Rerun** - The rerun report is a file that lists the location of failed Scenarios.
2) Note that some Cucumber implementations might not provide all of these formatter plugins, and some implementations might provide additional ones.
3) This is useful while fixing broken scenarios, as only the scenarios that failed in the previous run will be run again.
4) This can reduce time spent fixing a bug when running all scenarios is time-consuming.

# Cucumber

## Browser Automation

1) Cucumber is not a Browser Automation tool, but it works well with Browser Automation tools such as:
   - **Selenium WebDriver**
   - **Capybara**
   - **Watir**
   - **Serenity**

## References:

https://cucumber.io/docs

https://cucumber.io/docs/reference#gherkin

how
which
why
when
what
where
who

**People matter, results count.**

## About Capgemini

A global leader in consulting, technology services and digital transformation, Capgemini is at the forefront of innovation to address the entire breadth of clients' opportunities in the evolving world of cloud, digital and platforms. Building on its strong 50-year heritage and deep industry-specific expertise, Capgemini enables organizations to realize their business ambitions through an array of services from strategy to operations. Capgemini is driven by the conviction that the business value of technology comes from and through people. It is a multicultural company of 200,000 team members in over 40 countries. The Group reported 2016 global revenues of EUR 12.5 billion.

Learn more about us at

www.capgemini.com