SIS-312
GESTION DE CALIDAD DE SISTEMAS

Behavior Driven Development

Why is focusing TESTING on

BEHAVIOUR

so important?

Why is a focus on BEHAVIOUR so important?

- **Users** usually don't care about **technical implementation**

they care about **BEHAVIOUR** of the software

The software was crap and never did what I wanted ...

... but I LOVE the way you implemented that CLASS!!

"...our clients don't value the code as such; they value the things that the code does for them." –

http://hotlard.files.wordpress.com/2009/07/thumbs-up-lol-pictures-1.jpg

Behavior Driven Development – BDD

Is a software development process where teams create simple steps on how an application should behave from the user's perspective

BDD is a process carefully developed while keeping the principles of Agile Manifesto in mind.

BDD's evolution

- **Dan North** first wrote a paper about his idea in **2003**
- November **2007**, **QCon Developer Conference (https://qconferences.com/)**, Dan North presents **Behavior Driven Development**
- BDD's definition: "**It's about focusing on the behavior of an application from the point of view of its stakeholders**"
- BDD envisioned **to provide a *language*, a *process*, and a *tool* that would provide a single source of truth of software behavior** for the audience of both **non-technical and technical project members**.
- BDD says you should elevate your mind to a level of **behavioral abstraction above the code implementation**

What is BDD?

- In Agile environments, **Behavior Driven Development (BDD)** plays a vital role because it **strongly encourages the use of Agile methodologies** during the development and testing.
- BDD brings **customers, end-users, BAs, QAs, and SEs** of the software product into **one table for effective sharing of knowledge** on the system and its testing requirements.
- Make sure everyone **speak about the system**, its requirements and its implementation, in the same way
- **Discovery, Formulation and Automation**
- Create a **shared understanding based** on examples that drive the development
- User stories **created** and **maintained** collaboratively by all stakeholders
- A **formalized template**, that is a way to create testable User Stories, since they are defined in a formal manner

What are the BDD benefits?

- The biggest advantage of BDD approach for software development might be that they describe a **set of functions that a user expects from a system in a very concrete and direct manner.** The sum of these **behaviors essentially document a contract** with the user/client. If any of the tests fail, this contract is not upheld.
- Define **verifiable, executable and unambiguous** requirements ,
- **Creates** and **shares** living **documentation**
- Developing features that truly **add business value**
  - Very often **business needs** are not well understood by the **people who build the software**
- Bring **QA involvement to the forefront**, great for team dynamics

In order to success you need to take a approach that involves:

PEOPLE

People

Tester

Analyst

Architect

Senior developer

LANGUAGE, STORIES

Business analyst

Product owner

Mid-developer

Junior developer

WHAT?

Let's imagine a scene at Delorean, the Uber for time travel, where you work. Your team is responsible for writing software systems that calculate the payment processing for your users who are hailing rides from your company's time-traveling ridesharing service.

PO: Our next big project is to update our driver app to show rider locations on the timeline map.

You: And when do these riders show up on the timeline map?

PO: When the driver turns on the app and signals that she's driving.

You: OK, so that means when the app boots up and the DriverStatus service receives a POST we'll need to simultaneously fetch references from the HailingUser service based on time locality.

PO: Um… I guess so?

PO: In this story, we're going to add a coupon box to the checkout flow.

You: [Thinking out loud] Hm… would that mean we add a `/coupon` *route to the checkout API?*

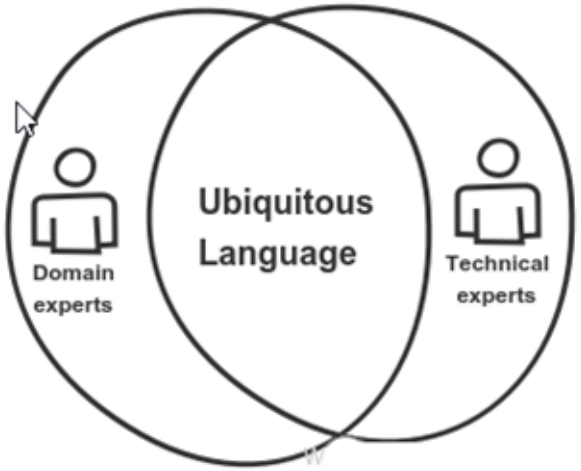
Teammate: Wait - I think we call them `Discounts` *in the backend. And the checkout flow is technically part of the* `RideCommerce` *service.*

You: Right - I mean let's call the route `/coupon` *but it'll create a* `Discount` *object. And in this story, let's just remember that the checkout API really refers to the* `RideCommerce` *service.*

PO: I'll add a note to the story.

Ubiquitous language

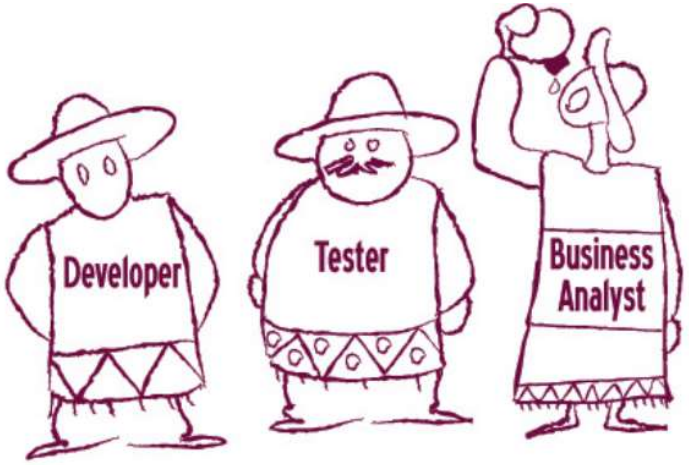
- Having a "Ubiquitous Language" means parties that communicate between each other **speak in the same language**
- It is **all about collaboration**
- Make sure everyone **speak** about the system, its requirements and its implementation, **in the same way**
- We want **to see the same terms** used to **discuss the system** to be present in the requirements, design documents, code, tests, etc.
- Review Ubiquitous Language & the joy of naming



PROCESS

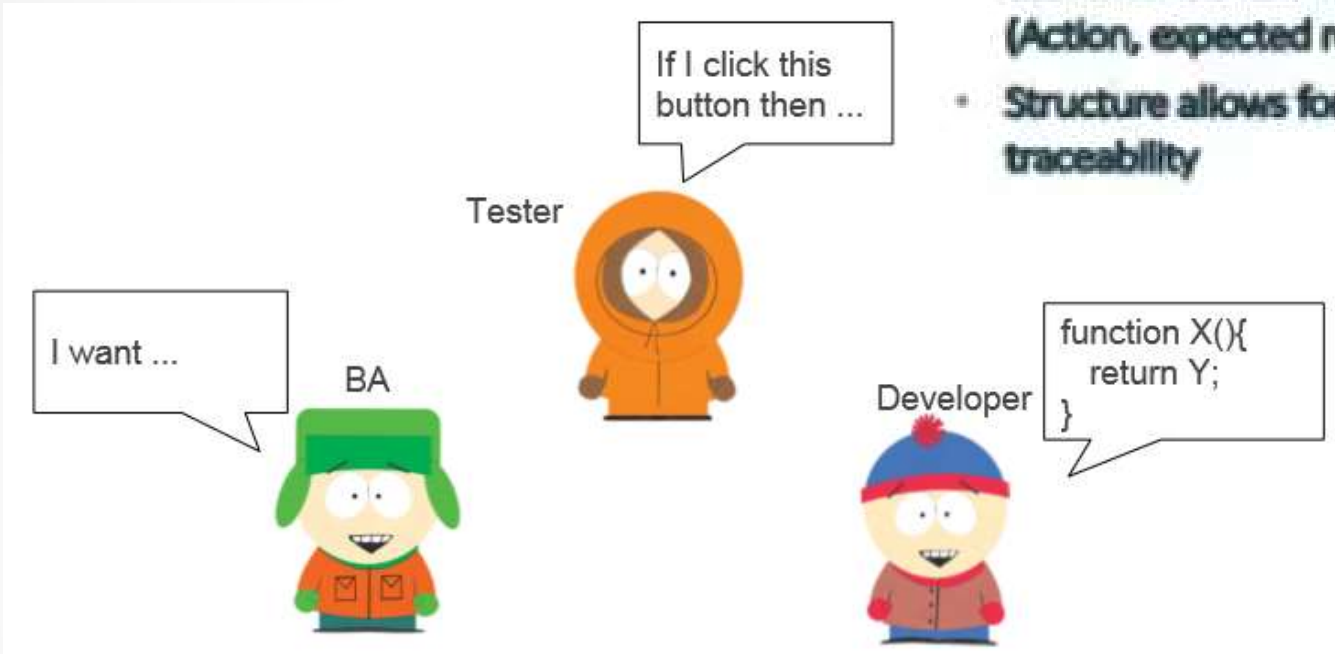
Three Amigos

- **Business Analysts**, '*Yes. I am confident that is what I need. Wait.. Let me think.. I guess so.* ' The focus for them would be acceptance criteria, look and feel of the product & core functionality.
- **Developers**, '*Three New programs and couple of new databases are required. It will take not less than 200 hours. Let me get back to my seat. There is lot of work.*' Their world is to somehow translate User stories to IF ELSE/SELECT statements.
- **QA**, will be running around with the smirk, '*Well, you missed the negative scenario. I am not going to signoff.*'



People Languages

- Scenarios contain Test definitions (Action, expected result)
- Structure allows for easy reporting and traceability



- Defines requirements in spoken-word scenarios
- Obtains feedback in clear, example heavy, and easy to understand language

- Making the Scenario "pass" or "work" guides development efforts and provides focus
- Scenario is easily translated directly into automated tests
- Given / When / Then structure provides simple ways to ask for clarification

Three Amigos

The people in charge of defining the requirements (business analysts / agile product owners) sit down with programmers and testers and discuss **features (**similar to **agile stories)** to be implemented.

- The **business person** specifies behaviors they want to see in the system.
- The **developer** asks questions based on their understanding of the system, while also writing down additional behaviors needed from a development perspective.
- The **testers** decides on which system behaviors the acceptance tests will be written.

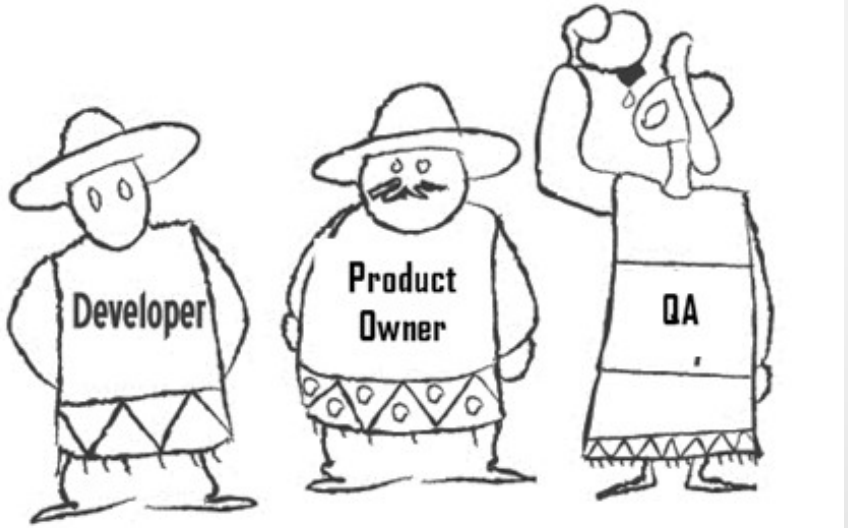
These **three amigos (business persons, developers, testers)** come up with **examples** of how the software should behave, and write them down as ***Cucumber Features and Scenarios***

Process

Goal

Learn as much possible before writing code

Have conversations about user stories and acceptance criteria using concrete examples



3 amigos meeting

- Keep them short ~ 25 minutes top
- User Stories
- Rules (Acceptance Criteria)
- Examples (Concrete examples covering each rules)
- Questions

Gherkin

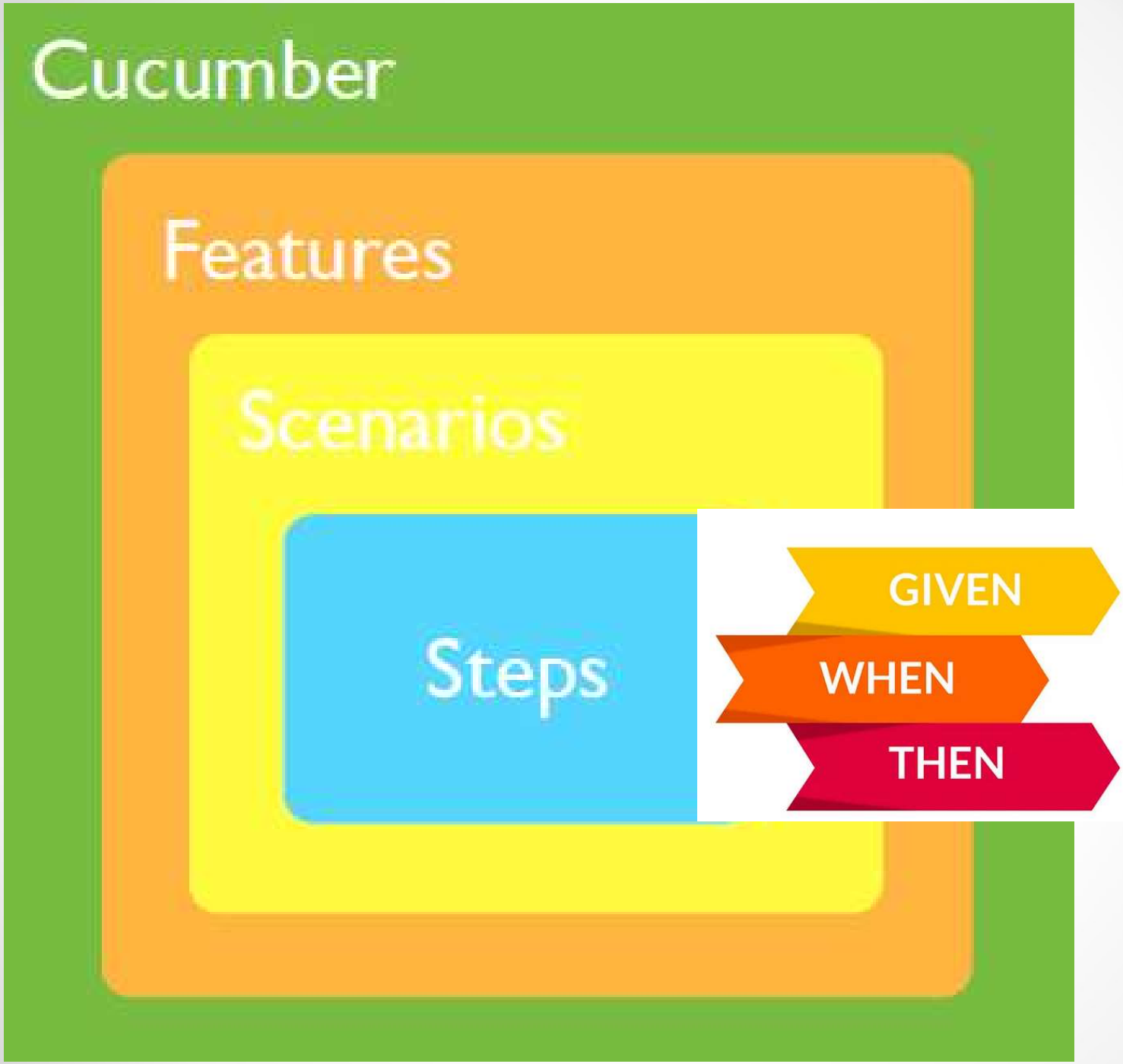
...

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

Gherkin

- Is the **domain-specific language for writing behavior scenarios**. It is a simple programming language, and its "code" is written into **feature files** (text files with a ".feature" extension).
- The **official Gherkin language standard** is maintained by **Cucumber**, one of the most prevalent BDD automation frameworks
- The purpose of the language is to help us **write concrete requirements**, we want to remove as much ambiguity as possible in our requirements since ambiguity is a proven source of errors
- Gherkin has the following syntax:

```
Feature     Background        Scenarios
Given       When              Then
And         But
Examples    Scenario Outline
```



Cucumber
Features
Scenarios
Steps
GIVEN
WHEN
THEN

Feature, Scenario & Steps

```
1: Feature: Descriptive text of what is desired
2: Any additional information that will make the feature
easier to understand
3:
4: Scenario: Some determinable business situation
5:     Given some precondition
6:       And some other precondition
7:     When some action by the actor
8:       And some other action
9:       And yet another action
10:    Then some testable outcome is achieved
11:      And something else we can check happens too
12:
13: Scenario: A different situation
14: ...
```


Feature

- The **feature** keyword is used to group a set of tests (scenarios)
- Description of a **requirement** and its **business benefit**, and a set of **criteria** by which we **all agree** that it is "**done**"

Flavor 1:

As a [User/Role]
I want [Behaviour]
so that [I receive benefit]

Flavor 2:

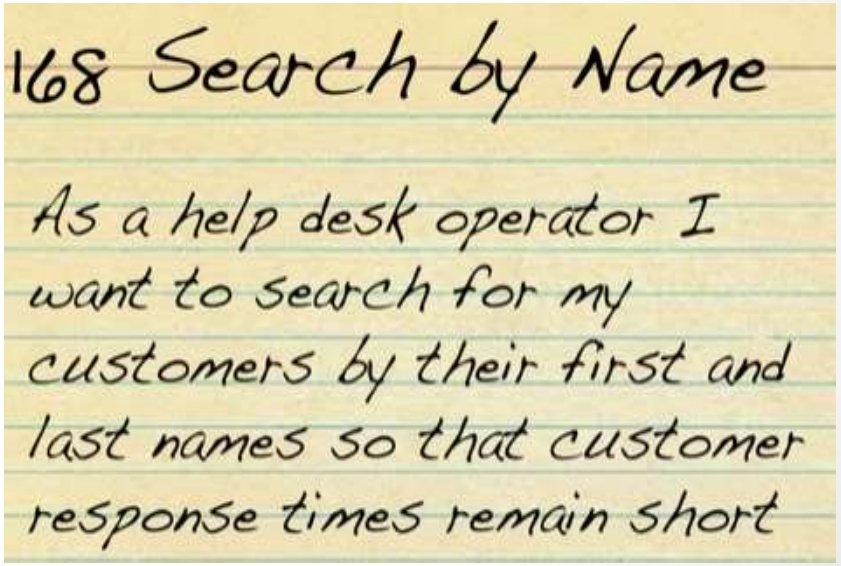
In order to [business value]
As a [role]
I want to [some action]

Feature Examples

Feature: Withdraw money
In order to avoid going to the bank
As a customer
I want to withdraw money from an ATM

As a buyer
I want to save my cart
So I can complete checkout later

Feature: Google Searching
As a web surfer,
I want to search Google,
so that I can learn new



Feature: Registered user
As a registered user
I want to log in
so I can access subscriber-only content

Scenario

Gherkin scenarios are **meant to be short** and to **sound like plain English.** Each scenario has the following structure:

- **Given** some initial state
- **When** an action is taken
- **Then** verify an outcome

The essential idea is to break down writing a scenario into three sections:

- The **given** part describes the state of the world before you begin the behavior you're specifying in this scenario. You can think of it as the **pre-conditions** to the test.
- The **when** section is **that behavior or action that you're specifying**, (e.g., push a button)
- Finally **then** section describes **the changes you expect due to the specified behavior**. Validates that the right thing happened

Scenario

- Using the **And/But** keywords you can add additional steps to the context, action, and outcome sections
- Used **instead of repeating Given, When, or Then**

  example: Given-Given-When-Then

  Using And : Given-And-When-Then
- These keywords help increase the **expressiveness** of the scenario
- Scenarios are the core of Cucumber. In the **"3 Amigos" meeting,** it is important to **have all three perspectives** to guarantee a full range of scenarios, both happy and unhappy paths

Scenario Examples

Scenario: Withdraw money from account

Given I have $100 in my account

When I request $20

Then $20 should be dispensed

Scenario: Log into the system

Given I visit the login page

And I enter a valid username and password

When I press login button

Then I should see a welcome message

Scenario Examples

```
Scenario: Save cart for later
  Given I have items in the cart
  When I click 'Save for later'
  Then the cart should be stored for future access
```

```
Scenario: Clicking the login button
  Given I click the login field
  And I type the username
  And I click the password field
  And I type the password
  And I click the login button
  Then I see the dashboard
```

```
Feature: Google Searching
  As a web surfer, I want to search Google, so that I can learn new things.

  Scenario: Simple Google search
    Given a web browser is on the Google page
    When the search phrase "panda" is entered
    Then results for "panda" are shown
    And the related results include "Panda Express"
    But the related results do not include "pandemonium"
```

Scenario Examples

```
Feature: Google Searching
  As a web surfer, I want to search Google, so that I can learn new things.

  Scenario: Simple Google search
    Given a web browser is on the Google page
    When the search phrase "panda" is entered
    Then results for "panda" are shown
    And the result page displays the text
      """
      Scientific name: Ailuropoda melanoleuca
      Conservation status: Endangered (Population decreasing)
      """
```

```
Feature: Google Searching
  As a web surfer, I want to search Google, so that I can learn new things.

  Scenario: Simple Google search
    Given a web browser is on the Google page
    When the search phrase "panda" is entered
    Then results for "panda" are shown
    And the following related results are shown
      | related       |
      | Panda Express |
      | giant panda   |
      | panda videos  |
```

Scenario Full Example

```
Account Holder withdraws cash

As an Account Holder
I want to withdraw cash from an ATM
So that I can get money when the bank is closed

Scenario 1: Account has sufficient funds
Given the account balance is \$100
 And the card is valid
 And the machine contains enough money
When the Account Holder requests \$20
Then the ATM should dispense \$20
 And the account balance should be \$80
 And the card should be returned

Scenario 2: Account has insufficient funds
Given the account balance is \$10
 And the card is valid
 And the machine contains enough money
When the Account Holder requests \$20
Then the ATM should not dispense any money
 And the ATM should say there are insufficient funds
 And the account balance should be \$20
 And the card should be returned

Scenario 3: Card has been disabled
Given the card is disabled
When the Account Holder requests \$20
Then the ATM should retain the card
And the ATM should say the card has been retained
```

Scenarios Outline

- Scenario outlines are parameterized using **Examples** tables. Each **Examples** table has a title and uses the same format as a step table.
- Each row in the table **represents one test instance** for that particular combination of parameters.
- In the example above, there would be two tests for this **Scenario Outline**. The table values are **substituted** into the steps above wherever the column name is surrounded by the "<" ">" symbols.

```
Scenario Outline: Simple Google searches
  Given a web browser is on the Google page
  When the search phrase "<phrase>" is entered
  Then results for "<phrase>" are shown
  And the related results include "<related>"

  Examples: Animals
    | phrase   | related       |
    | panda    | Panda Express |
    | elephant | Elephant Man  |
```

Scenarios Outline

Scenario: Withdraw less money ...
  Given I have 200 SEK on my account
  When I withdraw 100 SEK
  Then I get 100 SEK from the ATM

Scenario: Withdraw more money ...
  Given I have 50 SEK on my account
  When I withdraw 100 SEK
  Then I get 0 SEK from the ATM

Scenario: Withdraw money from the account
  Given I have <balance> SEK on my account
  When I withdraw <withdraw> SEK
  Then I get <received> SEK from the ATM

Examples:

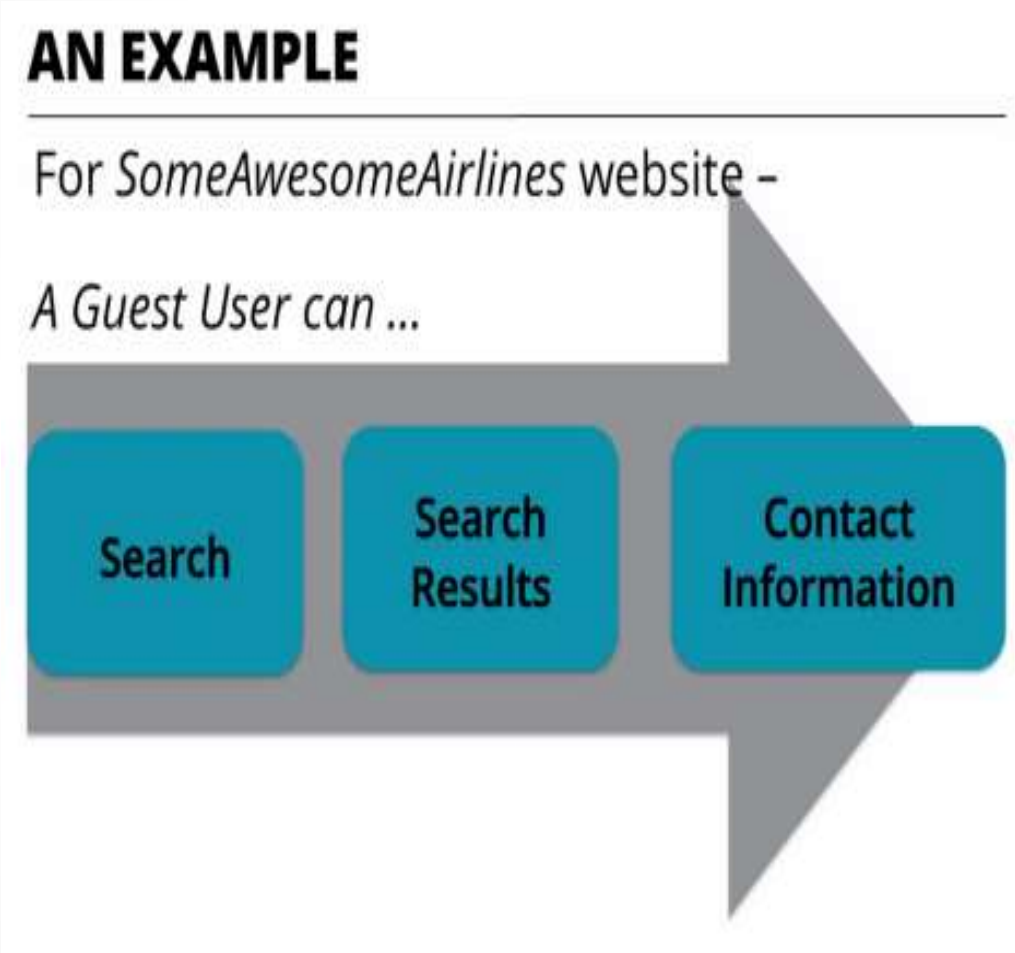
balance	withdraw	received
200	100	100
50	100	0

```
Feature: Order a pizza
     As a hungry person
     I want to order a pizza to be delivered to me
     So that I don't starve

Scenario Outline: Place Order
     Given I'm viewing the ordering page
     When I select toppings: <toppings>
     And I specify my address as <address>
     And I submit my order
     Then my order is confirmed
     And I have an option to return to the ordering page

     Examples:
       | toppings                          |  address                 |
       | sauce,   Limburger,  Gummy worms| Building 225, room A350 |
       | noSauce, Cheddar,    Spam       | Building 225, room A350 |
```

Two styles of writing a scenario



Two styles of writing a scenario

Long scenarios with low level steps describing **how to navigate the user interface**

The scenario is **business user oriented, very abstract and gets rid of the implementation** noise totally. It simply describes user interaction at a high level and the expected outcome.

IMPERATIVE STYLE

Given I am a guest user on the home page

And I choose "round" trip option

And I select "Chicago" from the origin dropdown

And I select "San Francisco" from the destination dropdown

And I select departure date as "5 December 2013"

And I select returning date as "25 December 2013"

When I click on Search

Then I should see the search results page

And I should see at least 1 option for my criteria

...

DECLARATIVE STYLE

Given I am a guest user

When I search for flight options for a "one-way" trip for "1" "Adult" from "Chicago" to "San Francisco"

And I select the "first" flight

And I enter "valid" contact details for "traveller1"

Then I am able to Save and Continue

Two styles of writing a scenario

From these examples, its easy to see that the **imperative style** offers **more details surrounding the actions** involved in the scenario and also **touches on the functionality/fields involved**

```
Given I open a browser
And I navigate to http://example.com/login
When I type in the username field bob97
And I type in the password field F1d0
And I click on Submit button
Then I should see the message Welcome Back Bob
```

The **declarative method** seems to **focus more on the scenario itself**, with the actions and the functionality not really being described.

```
Given I am on the Login Page
When I sign in with correct credentials
Then I should see a welcome message
```

Let's summarize it

From the previous slides, we can see that:

- features are explained by **scenarios**
- scenarios consist of **steps**

The spec is written in natural language in a plain-text file (low entry barrier), but **the spec is executable**!

Cucumber can guide us into turning the language of each step into an executable test case that calls our systems and can then either pass or fail. Let's see how it works!

Exercise

Visit

https://www.saucedemo.com/

Start writing your feature files and scenarios

Later we will automate !

SIS-312
GESTION DE CALIDAD DE SISTEMAS

Behavior Driven Development Technology


TECHNOLOGY

Components that enable BDD

Cucumber

https://cucumber.io/

Cucumber lets software development teams describe how software should behave in plain text. The text is written in a business readable domain specific-language and serves as documentation, automated tests and development-aid - all rolled into one format.

Ruby

https://www.ruby-lang.org/en/

A dynamic, open source programming language with a focus on simplicity and productivity. It has an elegant syntax that is natural to read and easy to write.

Capybara

https://teamcapybara.github.io/capybara/

Capybara is a library written in the Ruby programming language which makes it easy to simulate how a user interacts with your application.
Capybara can talk with many different drivers which execute your tests through the same clean and simple interface. You can seamlessly choose between Selenium, Webkit or pure Ruby drivers.

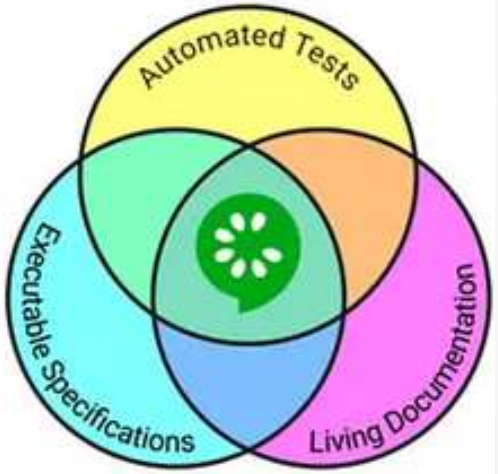
Cucumber – a tool used in BDD to write acceptance tests.

Cucumber

Behaviour Driven Development
with elegance and joy

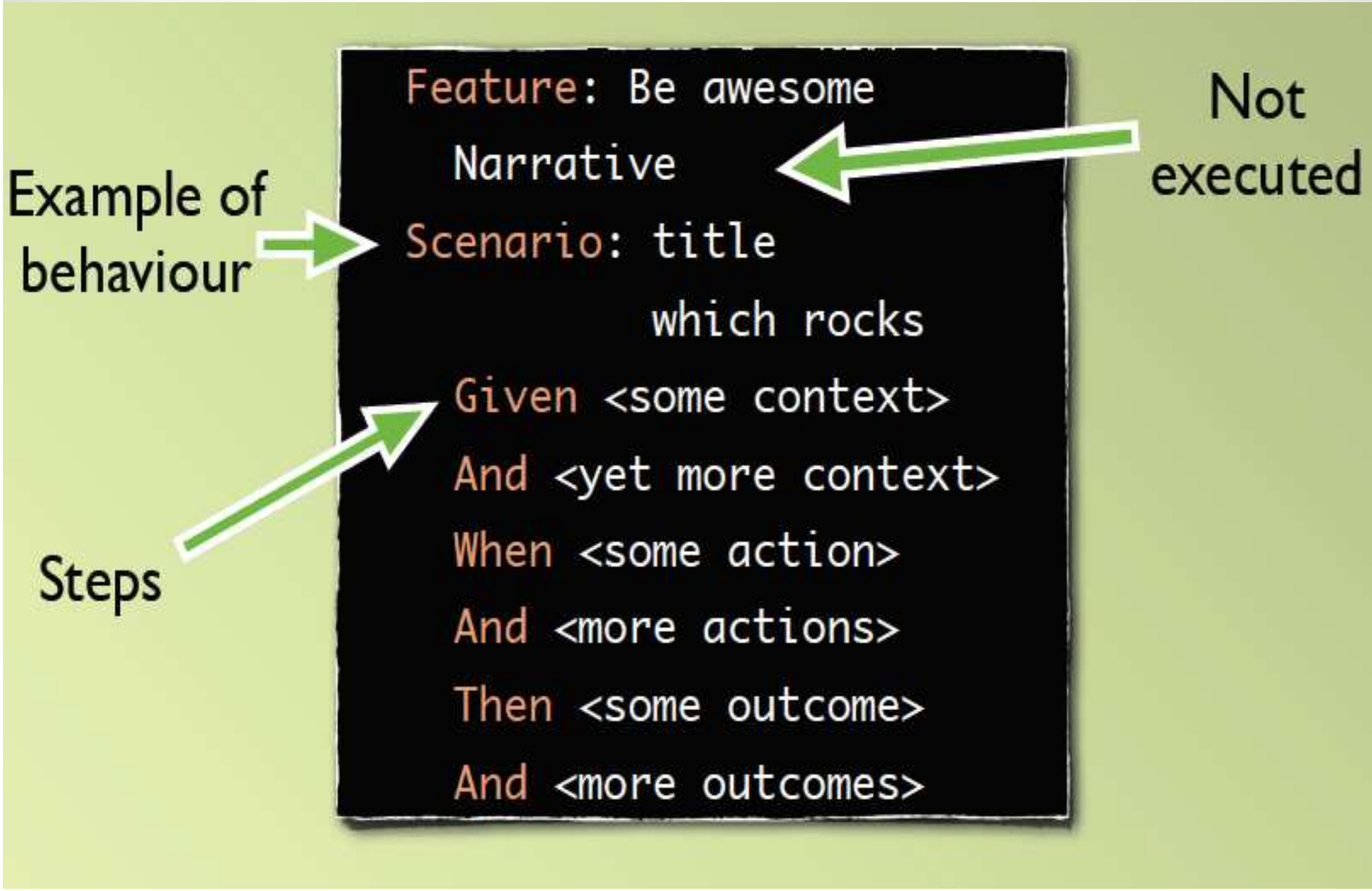
```
Feature: Addition
  In order to avoid silly mistakes
  As a math idiot
  I want to be told the sum of two numbers

  Scenario: Add two numbers
    Given I have entered 50 into the calculator
    And I have entered 70 into the calculator
    When I press add
    Then the result should be 120 on the screen
```

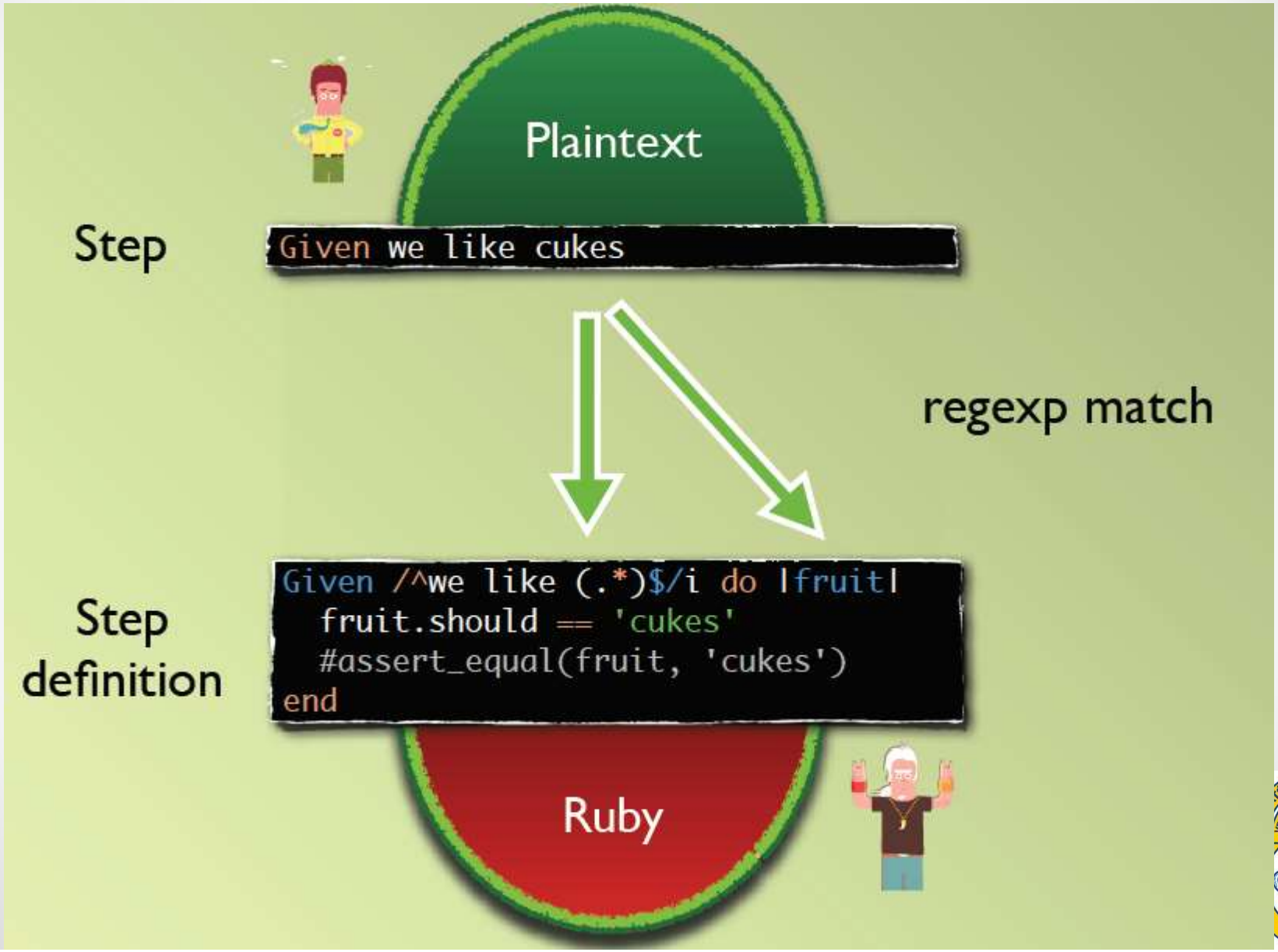


- Describes **how software should behave in plain text using Gherkin**
- Easy to set up
- The killer feature of Cucumber is the **ease of web workflow testing**
- Multiple **report** formats
- Supports writing specifications in about 40 spoken languages
- Integrates nicely with the rest of the **development ecosystem**. (Plain text)
- You can use **Cucumber** with **.NET and JVM languages** almost natively
- **Integrated** with all the **most popular web** testing libraries

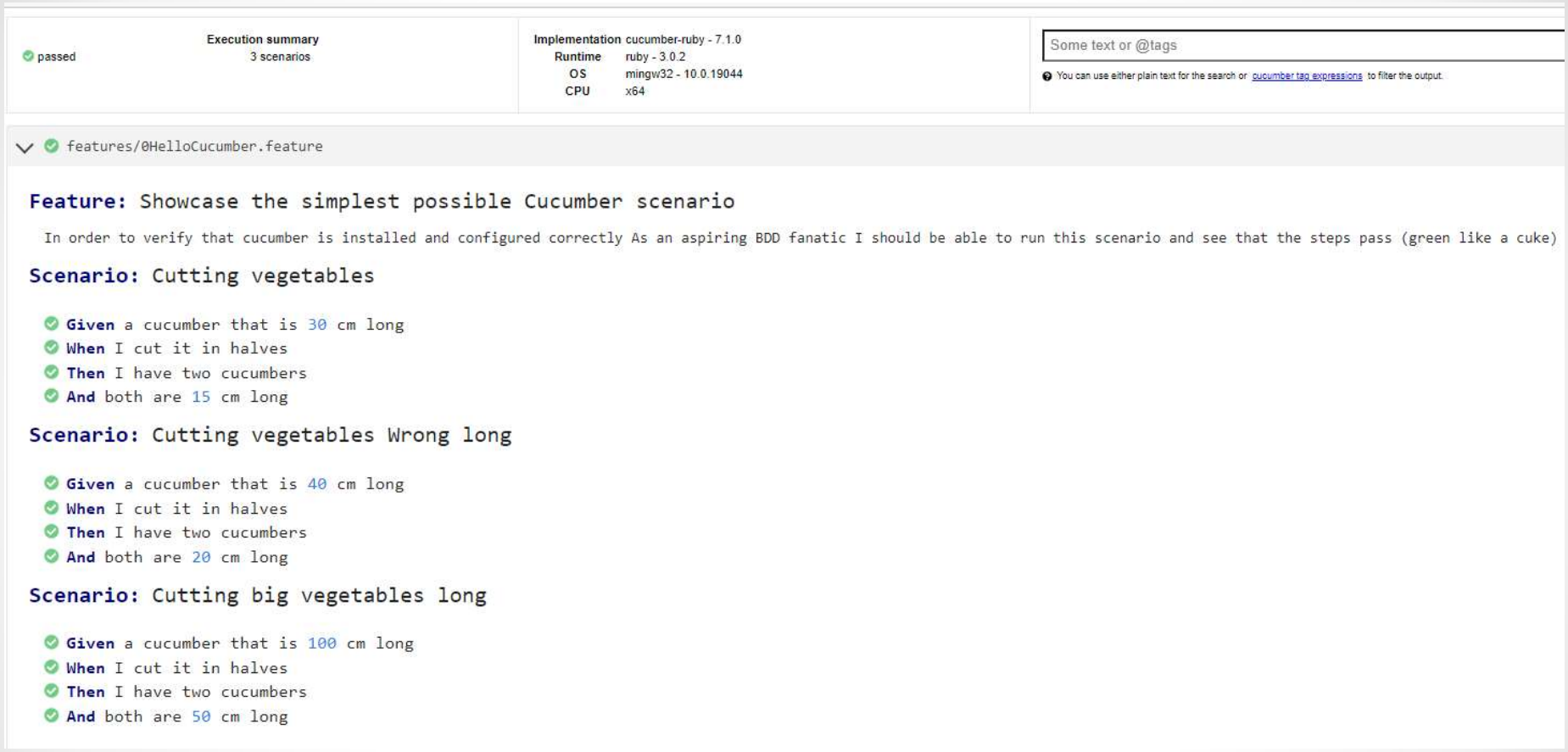
Cucumber feature



Steps and Step definition



Cucumber Report



passed

Execution summary
3 scenarios

Implementation cucumber-ruby - 7.1.0
Runtime ruby - 3.0.2
OS mingw32 - 10.0.19044
CPU x64

Some text or @tags

You can use either plain text for the search or cucumber tag expressions to filter the output.

features/0HelloCucumber.feature

Feature: Showcase the simplest possible Cucumber scenario

In order to verify that cucumber is installed and configured correctly As an aspiring BDD fanatic I should be able to run this scenario and see that the steps pass (green like a cuke)

Scenario: Cutting vegetables

- **Given** a cucumber that is 30 cm long
- **When** I cut it in halves
- **Then** I have two cucumbers
- **And** both are 15 cm long

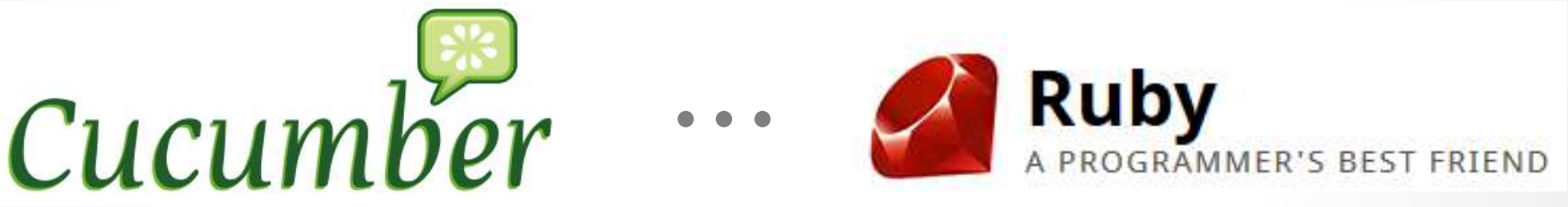
Scenario: Cutting vegetables Wrong long

- **Given** a cucumber that is 40 cm long
- **When** I cut it in halves
- **Then** I have two cucumbers
- **And** both are 20 cm long

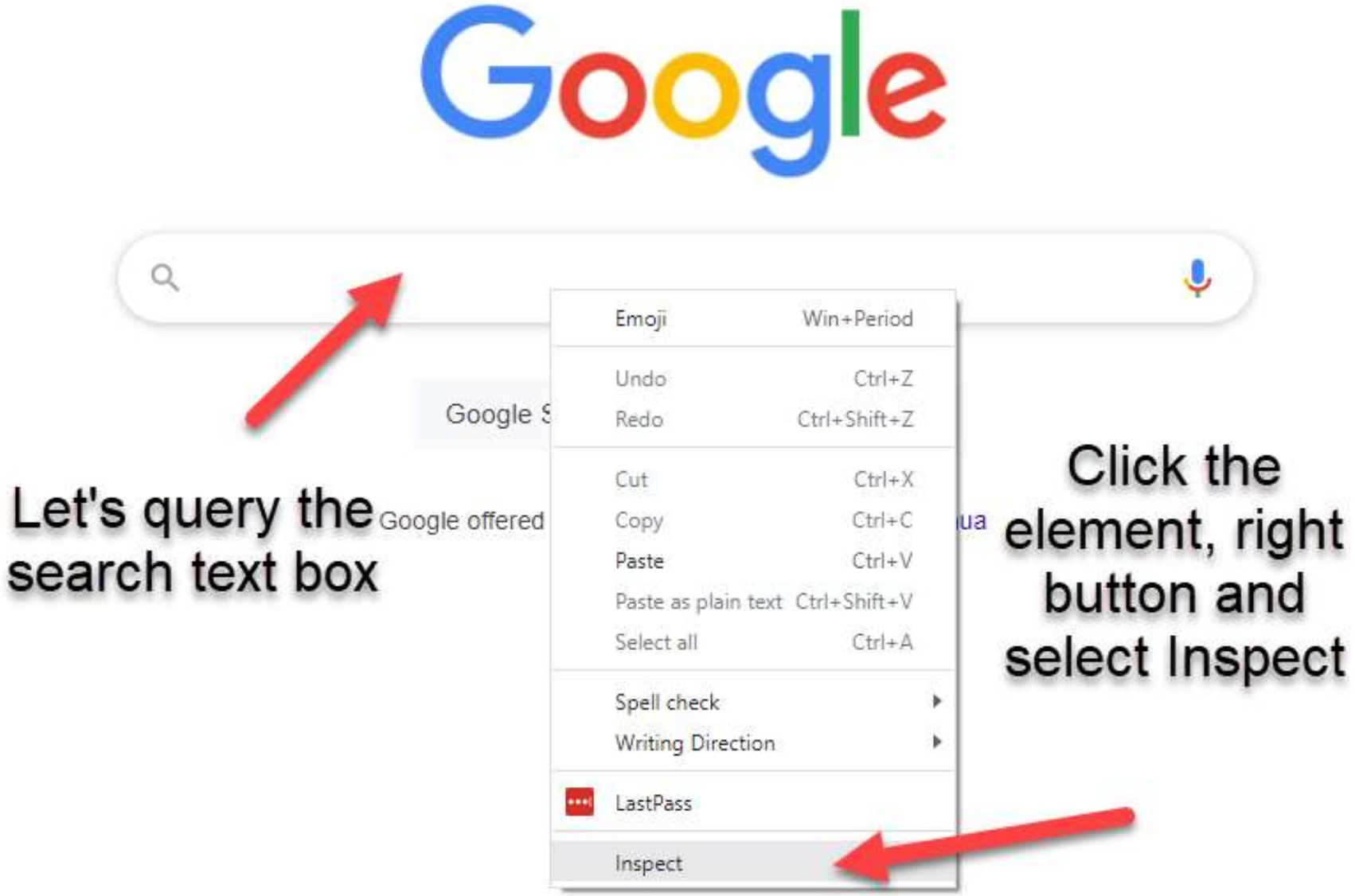
Scenario: Cutting big vegetables long

- **Given** a cucumber that is 100 cm long
- **When** I cut it in halves
- **Then** I have two cucumbers
- **And** both are 50 cm long

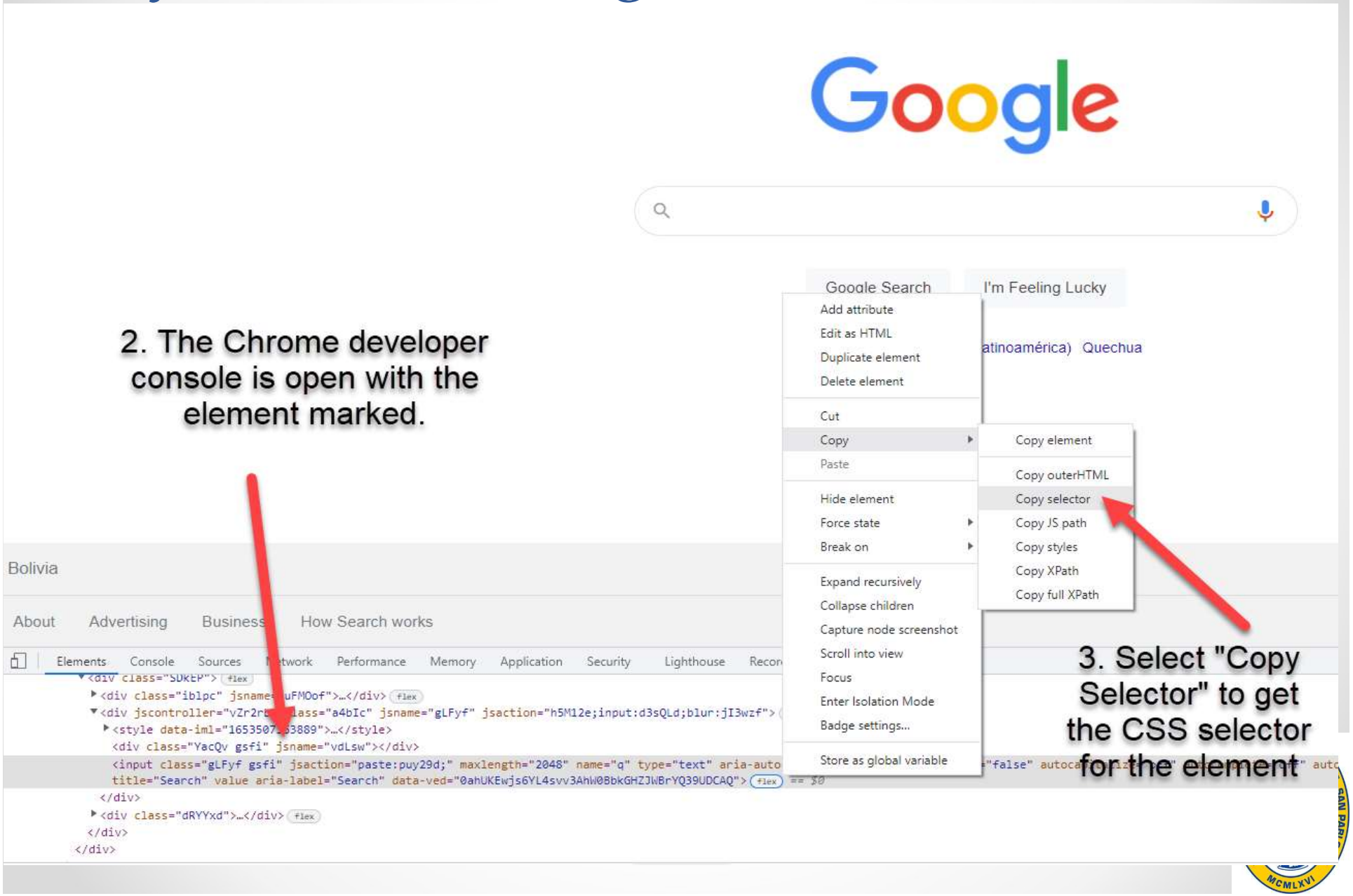
Cucumber and Ruby Demo



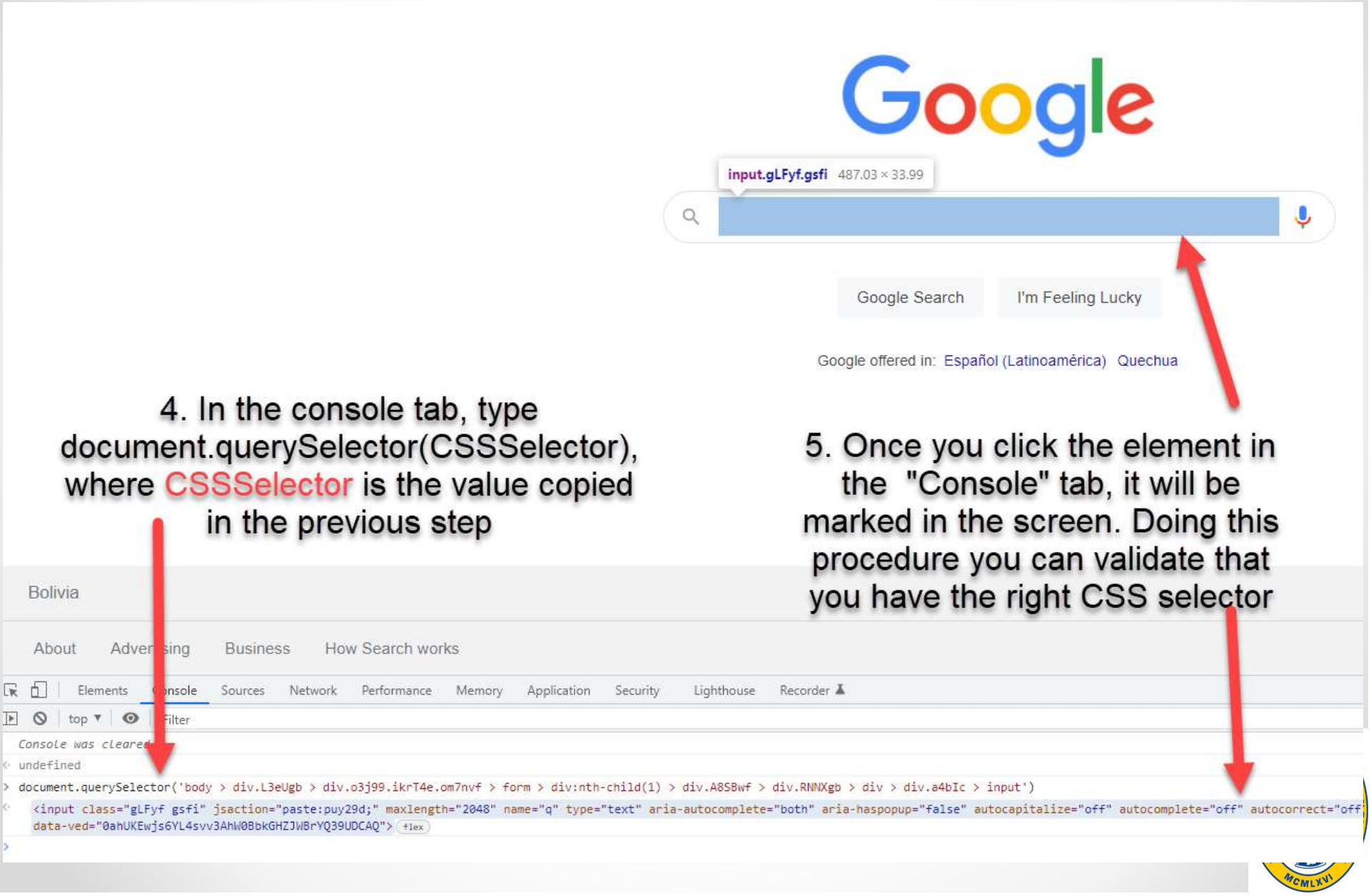
Query elements using Dev Console



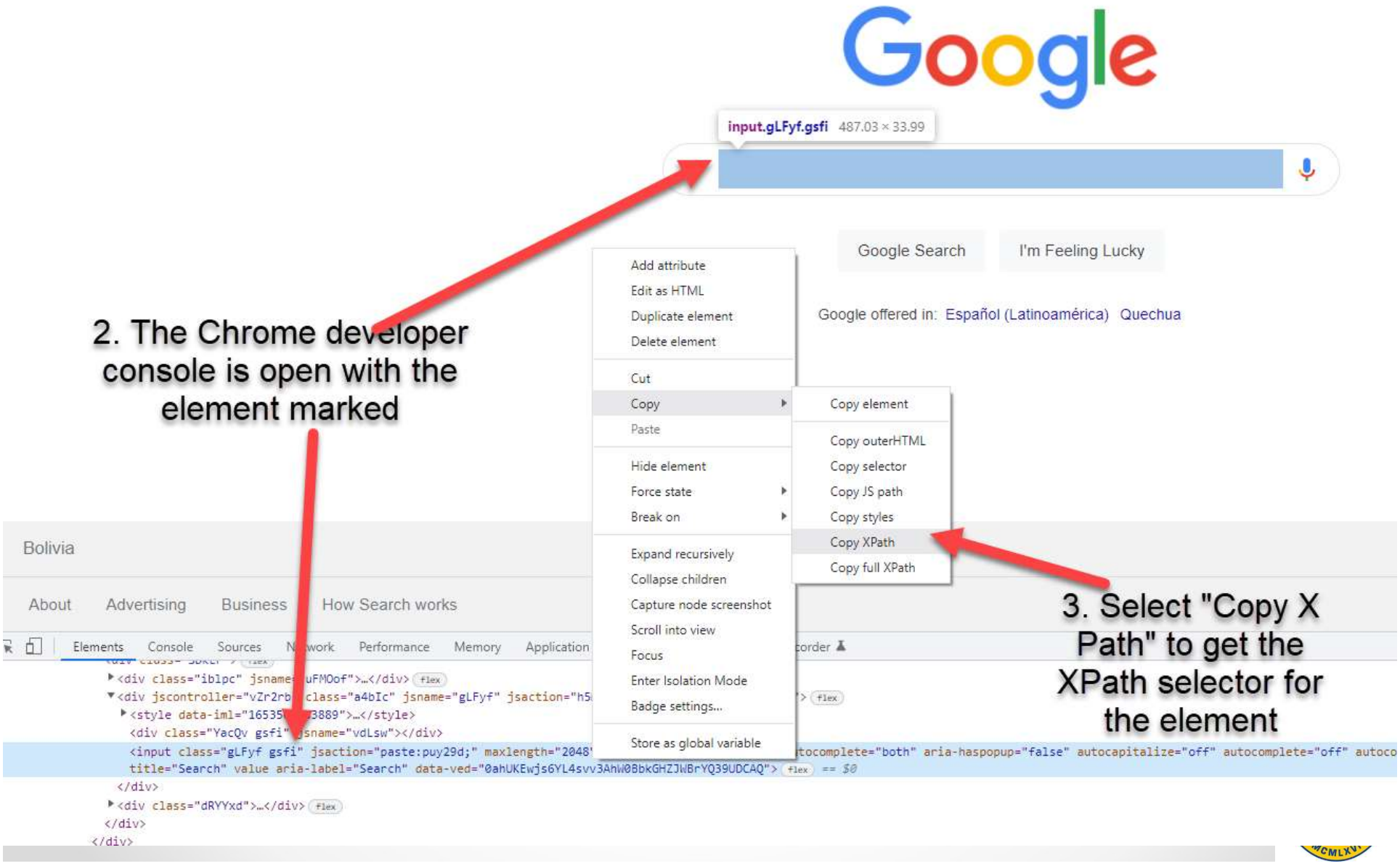
Query elements using Dev Console - CSS



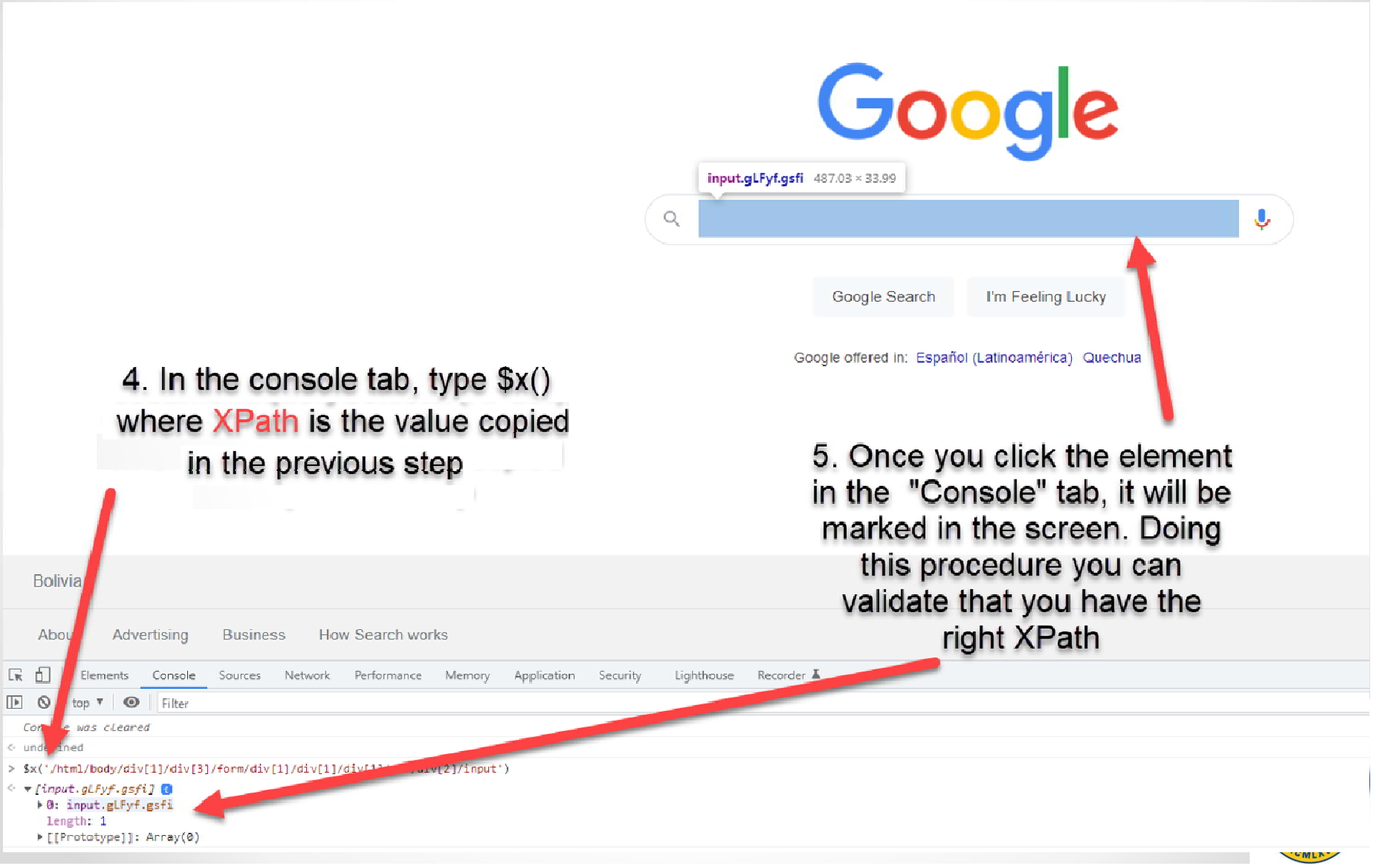
Query elements using Dev Console - **CSS**



Query elements using Dev Console – **XPath**



Query elements using Dev Console - XPath



Capyabara

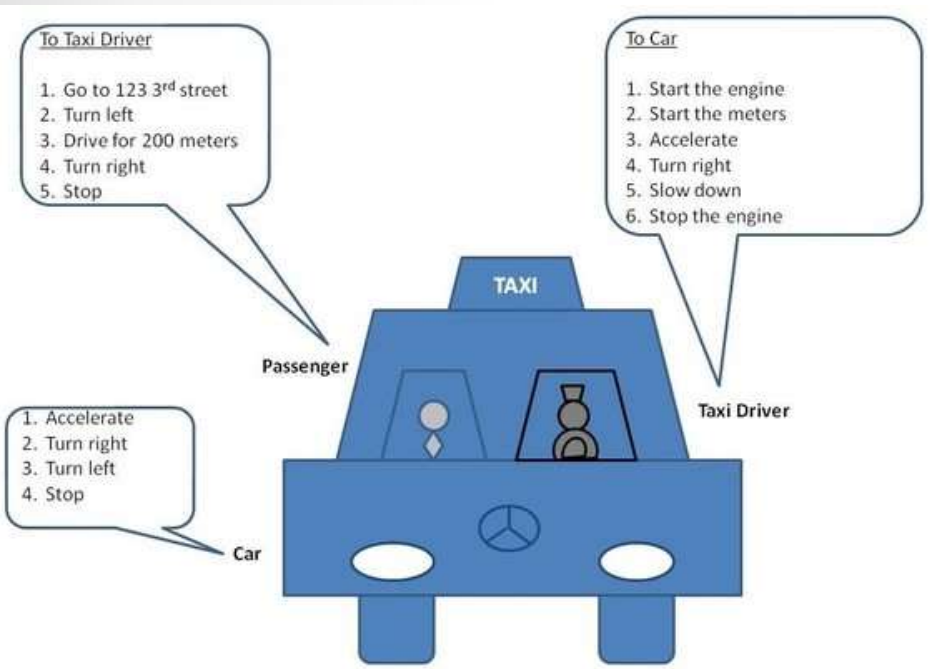
...

https://github.com/jnicklas/capybara

Capybara

- Capybara is a web-based automation framework used for **creating functional tests that simulate how users would interact with the application**
- A library/gem built to be used on top of an underlying web-based driver, in our case Selenium
- **Offers user-friendly DSL** (Domain Specific Language), which mimics the language an actual user would use.
  - click_button('button_name')
  - fill_in('First Name', :with => 'John')
  - choose('A Radio Button')
  - check('A Checkbox')
  - attach_file('Image', '/path/to/image.jpg')
- **Powerful synchronization**

Browser driver



Capybara - DSL

Cheatsheets

- https://kapeli.com/cheat_sheets/Capybara.docset/Contents/Resources/Documents/index
- https://gist.github.com/elfassy/11399304

Navigating

- visit('/projects')

Clicking links and buttons

- click_link('id-of-link')
- click_link('Link Text')
- click_button('Save')
- click('Link Text') # Click either a link or a button
- click_on('Button Value')
- find('form.foo .btn').click

Capybara - DSL

Interacting with forms

- fill_in('First Name', :with => 'John')
- fill_in('Password', :with => 'Seekrit')
- fill_in('Description', :with => 'Really Long Text…')
- choose('A Radio Button')
- choose("radio_group_selector"), option: "Option 5"
- check('A Checkbox')
- uncheck('A Checkbox')
- attach_file('Image', '/path/to/image.jpg')
- select('Option', :from => 'Select Box')
- unselect('Option', from: select_box)
- find("#select_id").select("value")

Capybara - DSL

Scoping

- within(:xpath, "//li[@id='employee']") do
- fill_in 'Name', :with => 'Jimmy'
- end
- within("li#employee") do
- fill_in 'Name', :with => 'Jimmy'
- end
- within_fieldset('Employee') do
- fill_in 'Name', :with => 'Jimmy'
- end
- within_table('Employee') do
- fill_in 'Name', :with => 'Jimmy'
- end

Capybara - DSL

Querying

- page.has_xpath?('//table/tr')
- page.has_css?('table tr.foo')
- page.has_content?('foo')
- page.should have_xpath('//table/tr')
- page.should have_css('table tr.foo')
- page.should have_content('foo')
- page.should have_no_content('foo')
- expect(page).to have_selector 'foobar'
- find_field('First Name').value
- find_link('Hello').visible? #false, finds only visible
- find_button('Send').click
- find('//table/tr').click
- all('a').each { |a| a[:href] }

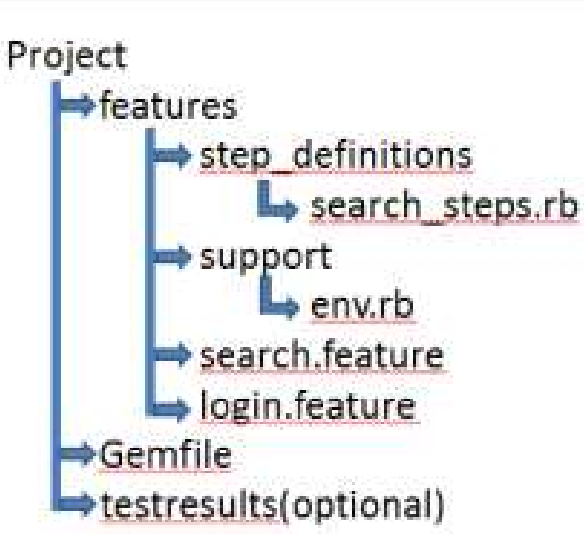
Capybara - DSL

Find actions

- find("input.file").attach_file
- find("input.checkbox").check
- find("input.select").choose
- find(".button").click_button
- find(".link").click_link
- find(".link").click_link_or_button
- find(".link").click_on
- find(".link").click
- find("input.text").fill_in(:with => 'Jimmy')
- find("input.select").select
- find("input.checkbox").uncheck
- find("input.select").unselect
- find("input.select").unselect_option(option: "Option 5")
- find("input.checkbox").checked?
- find(".button").disabled?
- find(".link").hover
- find("input.select").selected?
- find("input.text").value
- find(".text").text
- find(".link").visible?

Project Structure

I. Base Folder



Base Folder Structure

where:

features – folder to host all your feature files

step_definitions – folder to host all your step definition Ruby files

support – folder to host your configuration files (env.rb)

Gemfile – defines the top-level gems to be used in your project

Run cucumber

Before starting

- Copy 999RubySamples.zip file to a target directory

Command to run the script :

- cucumber features/<name of the feature file>.feature

```
F:\CodeRepo\999RubySamples>cucumber features\2TableExample.feature
Feature: Show how to work with tables

  Scenario: Calculate summation of a list of integer numbers # features/2TableEx
ample.feature:3
    Given a list of integer numbers                         # features/step_def
initions/tableSteps.rb:5
      | A | 25   |
      | B | 1500 |
      | C | 580  |
      | D | 600  |
    When I calculate the sum of them                        # features/step_def
```

Command to generate the report:

- cucumber features --format html --out reports

```
F:\CodeRepo\999RubySamples>cucumber features\2TableExample.feature --format html
 --out reports\table.html
```

Cucumber, Ruby and Capybara Demo

Cucumber ...

Ruby
A PROGRAMMER'S BEST FRIEND

Travel Sample

```
Feature: Mercury Tours Verify Registration

Scenario: Register a user on site
    Given I am on the Nercury Tours homepage
    And I click the "Register" link
    When I enter the required fields as show below
    |First Name:          | Pepito              |
    |Last Name:           | Perez               |
    |Phone:               | 1234-567-12         |
    |Email:               | pepe@pepazo.com     |
    |Address:             | Av. America #123    |
    |City:                | Cochabamba          |
    |State/Province:      | Cochabamba          |
    |Postal Code:         | 9897                |
    |Country:             | BOLIVIA             |
    |User Name:           | Pepazo              |
    |Password:            | ILoveQA             |
    |Confirm Password:    | ILoveQA             |
    And send my registration form
    Then the confirmation screen is show
    And my user name is "Pepazo"
```

```ruby
Given(/^I am on the Nercury Tours homepage$/) do
     page.driver.browser.manage.window.maximize
     visit ('http://newtours.demoaut.com/')
end

Given(/^I click the "([^"]*)" link$/) do |linkText|
  click_link(linkText)
end

When(/^I enter the required fields as show below$/) do |table|
  data = table.rows_hash
  data.each_pair do |key, value|
    case key
    when "First Name:"
        fill_in 'firstName', :with => value
        @name = value
    when "Last Name:"
        fill_in 'lastName', :with => value
        @lastName = value
    when "Phone:"
        fill_in 'phone', :with => value
    when "Email:"
        fill_in 'userName', :with => value
    when "Address:"
        fill_in 'address1', :with => value
    when "City:"
        fill_in 'city', :with => value
    when "State/Province:"
        fill_in 'state', :with => value
    when "Postal Code:"
        fill_in 'postalCode', :with => value
    when "Country:"
        select(value, :from => 'country')
    when "User Name:"
        fill_in 'email', :with => value
        @userName = value
    when "Password:"
        fill_in 'password', :with => value
    when "Confirm Password:"
        fill_in 'confirmPassword', :with => value
    end
  end
end

When(/^send my registration form$/) do
  xpath_base = '/html/body/div/table/tbody/tr/td[2]/table/tbody
  find(:xpath, xpath_base).click
end

Then(/^the confirmation screen is show$/) do
  greeting = "Dear"+" "+@name+" "+@lastName
  #page.should have_content()
  expect(page).to have_content(greeting)

end

Then(/^my user name is "([^"]*)"$/) do |userName|
  text= " Note: Your user name is "+userName+"."
  expect(page).to have_content(text)
end
```

Let's put it on practice!

Execution summary
passed 3 scenarios

Implementation cucumber-ruby - 7.1.0
Runtime ruby - 3.0.2
OS mingw32 - 10.0.19045
CPU x64

Some text or @tags

You can use either plain text for the search or cucumber tag expressions to filter the output.

features/2travel.feature

Feature: Mercury Tours Verify Registration

In order to book a flight in Mercury site As a registered customer I want to test the shoppping online options

Scenario: Register a user on site

- **Given** I am on the Mercury Tours homepage
- **And** I click the "Register" link
- **When** I enter the required fields as show below

First Name:	Pepito
Last Name:	Perez
Phone:	1234-567-12
Email:	pepe@pepazo.com
Address:	Av. America #123
City:	Cochabamba

Scenario: Find a flight with a register user

- **Given** I am on the Mercury Tours homepage
- **And** I enter my user and password
- **When** I press the "Sign-In" button
- **Then** the login successfully message is displayed

@cc
Scenario: Register a user on site

- **Given** I am on the Mercury Tours homepage
- **And** I click the "SIGN-ON" link
- **And** I enter my user and password
- **When** I press the Submit button
- **Then** the login successfully message is displayed

features/0google.feature

Feature: As a internet user

I want to use the google search engine
so I test that works correctly

@cc
Scenario: Search for the Houston Dynamo Website

- **Given** I am on the Google homepage
- **When** I search for "Tickets - Dynamo Houston"
- **Then** I will click the "Tickets - Dynamo FC" link

S– Ing. Carlos Canedo A.

Hooks Overview

- Hooks **allow us to perform actions at various points** in the cucumber test cycle
- You can define **hooks at different levels**: feature, scenario or step
- **Before** hooks will run before the first step of each scenario.
- **After** hooks will run after the last step of each scenario even when the test was failed or skipped.
- In the same way you have **Step hooks**, that are invoked before and after a step
- **Placing common functionality** in these **reduces the number of test steps** in each scenario

https://cucumber.io/docs/cucumber/api/#hooks

Background

- Often you find that several scenarios in the same feature start with a common context.
- Cucumber provides a mechanism for this, by providing a **Background** keyword where you **can specify steps that should be run before each scenario in the feature**. **Typically these will be Given steps**, but you can use any steps that you need to.
- A **Background** is **placed before the first Scenario/Example**, at the same level of indentation.
- **Hint**: if you find that some of the scenarios don't fit the background, consider splitting them into a separate feature.

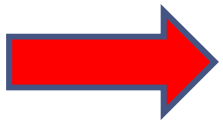
https://cucumber.io/docs/gherkin/reference/#background

Scenario Outline

- The Scenario Outline keyword can be used to **repeat the same steps** with **different values or arguments** being passed to the step definitions. This is helpful if you want to test multiple arguments in the same scenario.
- **The scenario will run for each row of the Example table**. In the example show below, when Cucumber starts to run this program, first, it will use the word "Refer" to check for palindrome and the output should be "true". Next, it will run the same scenario, but using the word "Coin" and output "false". The scenario will run for all the rows of the table.

```
Scenario: Valid Palindrome
    Given I entered string "Refer"
    When I test it for Palindrome
    Then the result should be "true"

Scenario: Invalid Palindrome
    Given I entered string "Coin"
    When I test it for Palindrome
    Then the result should be "false"
```



```
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"  |
```

Important

In creating scenarios, a key design goal is that they must be stateless

- Each scenario must make sense and be able to be executed independently of any other scenario

You can't have the success condition of one scenario depend on the fact that some other scenario executed before it

- Each scenario creates its particular context, executes one thing, and tests the result

Having stateless scenarios provides multiple benefits

- Tests are simpler and easier to understand
- You can run just a subset of your scenarios and you don't have to worry about your test set breaking
- Depending on your system, you might be able to run tests in parallel reducing the amount of time it takes to execute all of your tests

Useful links

Gherkin

- https://cucumber.io/docs/gherkin/reference/
- http://itsadeliverything.com/declarative-vs-imperative-gherkin-scenarios-for-cucumber

Cucumber

- https://cucumber.io/
- https://www.tutorialspoint.com/cucumber/index.htm
- https://cucumber.io/docs/gherkin/step-organization/
- https://cucumber.io/docs/gherkin/reference/#scenario-outline

Capybara

- https://github.com/jnicklas/capybara
- https://www.rubydoc.info/github/teamcapybara/capybara

Ruby

- https://www.tutorialspoint.com/ruby/

Useful links

BDD

- https://blog.kloia.com/behaviour-driven-development-bdd-cucumber-b656206beadb
- https://dannorth.net/introducing-bdd/
- http://dannorth.net/whats-in-a-story/