Power Cucumber and Capybara

The most powerful user stories exploit a domain language so you can specify high-level scenario steps like "Given I am logged in as 'Nancy'" or "When I purchase the item 'Cucumber'" that may require several steps. Getting used to Cucumber and Capybara can be tricky, though, so here's some power tools you can use to improve the expressiveness and power of your own Cucumber scenarios.

Use instance variables to carry state across steps

Cuke scenarios execute in the context of an object called a `World`. When you reference instance variables in a Cuke step definition, they are instance variables of that object, so they survive across steps (but not across scenarios). This lets you carry state across Cucumber steps.

For example, in an extended RottenPotatoes site, a user has a collection of favorite movies. The URI for a logged-in user's landing page is `/users/:id` (where :id is a primary key) and the URI for viewing that user's favorite collection is `/users/:id/favorites`. How would you use instance variables in Cuke steps to create the following step defs?

```
Given I am logged in as user "An Ju"
And my favorite collection contains the item "The Bourne Identity".
```

One possibility is to use an instance variable to remember the logged-in user:

```
Given /^I am logged in as user "(.*)"$/ do |username|
  login_as!(username) # simulate logging in
  @current_user = User.find_by_name!(username) # remember logged-in
user
end

Given /^my cart contains the item "(.*)"$/ do |item|
  @cart = @current_user.cart
  # ...etc
end
```

Scope your steps to specific divs or other elements, especially when using "Then I should see..."

Suppose we redesign the RottenPotatoes home page to always show "top rated movies" on the top part of the page, and show user-specific content on the bottom part. Here's a happy path scenario for "search for a movie by year" (line numbers added for clarity only):

```
 1 Given the following movies exist:
 2   | title        | year | rating |
 3   | Bridesmaids  | 2012 | PG-13  |
 4   | Cloud Atlas  | 2011 | R      |
 5   | The Help     | 2011 | R      |
 6 When I search for movies released in "2011"
 7 Then I should see the following: Cloud Atlas, The Help
 8 But I should not see the following: Bridesmaids
```

Exercise: As a warmup, fill in the step def below by arranging for movies to contain a list of movie titles so the step def will run:

```
Then /^I should see the following: (.*)$/ do |list|
  movies = _______________
  movies.each do |movie|
    steps %Q{Then I should see "#{movie}"}
  end
end
```

Hint/Reminder 1: here's the documentation for using nested steps in Cucumber.
Hint/Reminder 2: `%Q{...}` can be used to specify a string literal with double-quote semantics, that is, you can interpolate variables but you can include double-quote characters without escaping them.

The blank can be filled with `movies = list.split(/\s*,\s*/)`

The pitfall with this step def is as follows. Suppose the "top rated movies" pane happens to be showing "Bridesmaids". Then line 8 will fail even if our search code is correct. Conversely, if "top rated movies" happens to be showing Cloud Atlas or The Help, line 7 may pass even if our search code doesn't work! This kind of pitfall is particularly dangerous when you're checking for something that appears frequently, like the logged-in user's name (in many apps, it's shown on every page view).

The solution is to structurally separate different parts of the page and make each chunk easily and uniquely identifiable; for example, the search results might be displayed in a `div` whose id is search_results. This is because Cucumber supports steps of the form:

```
Then I should see "Cloud Atlas" within "div#search_results"
```

(You can find this step definition in `web_steps.rb`. As you can see, it's a thin wrapper around Capybara's `contain`, and the argument to `within` can be any CSS selector. The Within helper is provided by Capybara and is a good example of using closures in Ruby.)

Exercise: Use this scoped version as a building block to fix the step def for step 7, *without* changing the phrasing of the step itself in the scenario. (Hint: consider reusing the existing step def in `web_steps.rb`)

```
Then /^I should see the following: (.*)$/ do |list|
  movies = _____________
  movies.each do |movie|
    step %Q{Then I should see "#{movie}" within "#search_results"}
  end
end
```

Exercise: by changing just two lines in the step def, make it work for step 8 as well as step 7.

```
Then /^I should (not )?see the following: (.*)$/ do |neg,list|
  movies = _____________
  movies.each do |movie|
    step %Q{Then I should #{neg}see "#{movie}" within
"#search_results"}
  end
end
```

Writing your own steps using Capybara; the "big guns" of XPath

Besides `contain`, which looks at element text, Capybara provides a large number of "matchers" for examining the page. For example, if you just want to check for the presence of a page element (say, an error message), you can write

```
expect(page).to have_selector("p.error", :text => 'An error
occurred')
```

(The actual method defined in Capybara is `has_selector?`, because RSpec is smart enough to interpret `have_selector` as a call to `has_selector?`.)

`page` is a Capybara method that returns a representation of the page object, and `page.body` is a string containing the raw HTML of the full page text.

Here's an overall quick reference (README) and a full list of the matchers Capybara adds to RSpec. The spreewald gem contains a collection of highly useful Cucumber steps, many based on Capybara's advanced XPath capabilities, for testing various other aspects of the page, such as whether a set of strings appears in a certain order.

The most general and most powerful Capybara method is `find()`, which lets you use an arbitrary XPath expression, and its wrapper `has_xpath?`, which turns the result of `find()` into a Boolean true if the result matches any elements and nil otherwise. XPath lets you not only look for elements nested in other elements, but also match on elements having specific attributes or specific text:

```
page.find(:xpath, "//table/tr[@class='adult']")
   # return all table rows with css class "adult"

page.should have_xpath("//title[contains(text(),'Rotten')]")
   # ensure page has <title> element whose text includes "Rotten"
```

The online tool XPathTester lets you copy-and-paste a chunk of HTML or XML and then test various XPath expressions against it.

Exercise. Use XPath to fill in the following step def that checks whether a given option is selected in a dropdown menu.

```
Then "January" should be selected in the "Month" menu

Then /^"(.*)" should be selected in the "(.*)" menu$/ do
|choice,menu|
   if page.find(:xpath, "//select[@id='#{menu}'").blank?
     menu_id = ____    # handle <label for="month">Month</label> …
<select id="month">...</select>
     menu_id = page.find(:xpath,
"//label[contains(text(),'#{menu}'").first['for']
   else
     menu_id = menu
   end
  page.should
have_path("//select[@id='#{menu_id}']/option[@selected='selected'][co
ntains(text(),'#{choice}')]")
```

Exercise. Use either your own methods or the Spreewald gem to write a step def for the following type of step:

```
Then the row containing "Cloud Atlas" in table "search_results"
should appear before the row containing "The Help"
```

Hint: use `page.find(:xpath, ...)` to locate the group of rows, then match a regular expression against the raw text of those rows to check the order.

[no solution given]