University of Sheffield

COM1001 SPRING SEMESTER

Professor Phil McMinn

p.mcminn@sheffield.ac.uk

Writing Automated Tests with RSpec

RSpec

RSpec is a tool that we can use to write automated test cases.

It has some similarities but important differences with **JUnit**.

First of all, **it can be used for testing at all different scopes** – unit, integration, system, not just unit testing, as with JUnit.

Secondly, it takes a **behaviour-driven approach**.

> **Tests *describe* how some aspect of the system should work**. These descriptions map easily to a **Given-When-Then** structure, making it easy to translate **acceptance criteria from user stories into tests**.

The Bare Bones

We need the `rspec` gem

```ruby
require "rspec"

RSpec.describe "A sandwich" do
  # ...
  it "tastes delicious" do
    # ...
  end
end
```

We're describing how a **sandwich** should behave

The sandwich should taste delicious (else what's the point?!)

The Bare Bones

Actual code would go here if this were a real test – to set up the sandwich and to check that it does indeed taste delicious!

```
require "rspec"

RSpec.describe "A sandwich" do
  # ...
  it "tastes delicious" do
    # ...
  end
end
```

We're describing how a **sandwich** should behave

The sandwich should taste delicious (else what's the point?!)

The Bare Bones

Actual code would go here if this were a real test – to set up the sandwich and to check that it does indeed taste delicious!

```
require "rspec"

RSpec.describe "A sandwich" do
  # ...
  it "tastes delicious" do
    # ...
  end
end
```

We're describing how a **sandwich** should behave

The sandwich should taste delicious (else what's the point?!)

A Slightly More Complex Example

```
require "rspec"

RSpec.describe "A sandwich" do
  context "when it contains salad" do
    # ...
    it "is healthy" do
      # ...
    end
  end

  context "when it contains cheese" do
    # ...
    it "is delicious" do
      # ...
    end
  end
end
```

We can check the behaviour of our sandwich in different scenarios (contexts)

Note how this maps to the **Given-When-Then** style of acceptance criteria:

Given a sandwich
When it contains cheese
Then it is delicious

(Apologies to any vegans…)

RSpec in Practice – testing `strip`

Suppose we need to write tests for the `strip` Ruby method, which can be invoked on strings.

The strip method removes leading and trailing whitespace from a string, and is often used to sanitise inputs that users enter into forms.

```ruby
require "rspec"

RSpec.describe String do
  describe ".strip" do
    context "when invoked on a string with leading spaces" do
      it "returns the string with the leading spaces removed" do
        expect("  hello".strip).to eq("hello")
      end
    end

    context "when invoked on a string with trailing spaces" do
      it "returns the string with the trailing spaces removed" do
        expect("hello   ".strip).to eq("hello")
      end
    end

    context "when invoked on a string with no leading or trailing spaces" do
      it "returns the same string" do
        expect("hello".strip).to eq("hello")
      end
    end
  end
end
```

week[X]/rspec_example/string_strip_spec.rb

Since the strip method is part of the `String` class, this is part of the description of how a string should work in Ruby

The second `describe` block is for the strip method itself (written `.strip` by convention)

Finally, some test code! Here we're asserting the actual output is equal to that expected (`"hello"`). These types of statements are called **expectations**.

Matchers

Expectations are expressed in the form:

```
expect(actual_result).to/not_to matcher(expected_result)
```

e.g., `expect(result).to eq(0)`

Other examples of **matchers** include:

Matcher	Passes if…
`be true`	`actual_result == true`
`be false`	`actual_result == false`
`be_nil`	`actual_result.nil?`
`be < expected_result`	`actual_result < expected_result`
`be > expected_result`	`actual_result > expected_result`
`be <= expected_result`	`actual_result <= expected_result`
`be >= expected_result`	`actual_result >= expected_result`
`be_between(x, y).inclusive`	`actual_result >= x && actual_result <= y`
`start_with(x)`	`actual_result.start_with?(x)`
`include(x)`	`actual_result.include?(x)`

For more information, see the RSpec documentation at https://rspec.info/documentation

Live Demonstration:

the `rspec_example` example

(from the com1001-examples GitHub repository)

Featuring:

- Running `rspec` on a file or directory
- Failing tests