RSpec

Anatomy of an RSpec project

To use RSpec, we'll need to structure our project files in a certain way. We separate our implementation code files from the testing files using a `/lib` and `/spec` folder respectively. Another word for a "test" is a "spec" (short for specification, since the tests specify how our code should behave). Let's say we had two methods that we wanted to have tests for, `add` and `prime?`, then we can structure our project like so:

```
/example_project
  ├── lib
  │   ├── add.rb
  │   └── prime.rb
  └── spec
      ├── add_spec.rb
      └── prime_spec.rb
```

To use RSpec, we **must** follow this structure. We need folders with the literal names `lib` and `spec` as direct children of the `example_project` folder. The test files inside of the `/spec` folder must end with `_spec` in their names.

How to Read Specs

Let's take a look at the contents of `/spec/add_spec.rb` to see how we test the `add` method. The behavior outlined in the specs will dictate how we ought to design the method in `/lib/add.rb`.

```
# /spec/add_spec.rb

require "add" # this line will include code from "/lib/add.rb"

describe "add method" do
  it "should accept two numbers as arguments" do
    expect { add(2, 3) }.to_not raise_error
  end

  it "should return the sum of the two numbers" do
```

```
    expect(add(2, 3)).to eq(5)
    expect(add(10, 12)).to eq(22)
  end
end
```

Reading this code, you should get the feel of how the `add` method will be tested. Here's the semantic interpretation of the code:

- The description of the add method outlines 2 criteria:
  - it should accept two numbers as arguments
  - it should return the sum of the two numbers

Don't worry, all of these terms are pretty self explanatory. For example, try to interpret the spec we would use for the `prime?` method:

```
# /spec/prime_spec.rb
require "prime"

describe "prime? method" do
  it "should accept a number as an argument" do
    expect { prime?(7) }.to_not raise_error
  end

  context "when the number is prime" do
    it "should return true" do
      expect(prime?(7)).to eq(true)
      expect(prime?(11)).to eq(true)
      expect(prime?(13)).to eq(true)
    end
  end

  context "when the number is not prime" do
    it "should return false" do
      expect(prime?(4)).to eq(false)
      expect(prime?(9)).to eq(false)
      expect(prime?(20)).to eq(false)
      expect(prime?(1)).to eq(false)
    end
  end
end
```

Above we use `context` additional blocks to outline different scenarios that our code is expected to satisfy. A straight forward interpretation to the

first `context` is: *When the number is prime, it should return true*.

Wrapping Up

Here are the core RSpec terms you'll see in every spec file:

- `describe` names the method being tested
- `it` expresses the expected behavior of the method being tested
- `expect` shows how that behavior is tested