Basic RSpec Syntax

Grouping

RSpec tests are written using the `describe` and `it` methods.

- `describe` is used to group a set of tests together and define a subject.
- `it` is used to define a specific test case.

```
describe MyClass do
  describe "instantiates" do
    it "to an instance of MyClass" do
      expect(MyClass.new).to be_an_instance_of(MyClass)
    end

    it "without errors" do
      expect {MyClass.new}.to_not raise_error
    end
  end
end
```

Expectations

Expectations are used to specify the expected behavior of the code being tested.

```
expect(value).to matcher(expected_value)
expect(value).to_not matcher(expected_value)
```

Some commonly used matchers include:

- `eq` - check if two values are equal
- `be` - check if two objects are the same object
- `be_within` - check if a value is within a certain range
- `be_truthy` / `be_falsy` - check if a value is truthy/falsy through duck typing
- `be_empty` - check if a collection (or string) has no elements
- `raise_error` - check if a certain error is raised
- `receive` - check if an object received a method call

```
expect(4).to eq(4)
expect(obj1).to be(obj2)
expect(3.14).to be_within(0.15).of(3.0)
expect(4).to be_truthy
expect("").to be_empty
expect { raise "Error!" }.to raise_error("Error!")
expect(obj1).to receive(:method).with(arg1, arg2).and_return(value)
```

Stubs

Stubs facilitate testing without depending on the implementation of methods that we aren't testing. They are used to simulate behavior or values during testing.

```
@obj = MyClass.new
allow(@obj).to receive(:method).and_return(value)
```

Doubles

Doubles allow you to test code without depending on the implementation of another class.

```
@my_double = double('myclass')
allow(@my_double).to receive(:method).and_return(return_value)
```

`before` and `after` blocks

`before` and `after` blocks are executed before and after (respectively) running tests.

- `before` blocks are often used to set up state that is required for each test
- `after` blocks are often used to clean up state that was created during each test.

```
describe "Calculator" do
  before(:each) do
    @calculator = Calculator.new
  end

  after(:each) do
    @calculator.clear_history
  end

  it "adds two numbers" do
    expect(@calculator.add(2, 3)).to eq(5)
  end

  it "multiplies two numbers" do
    expect(@calculator.multiply(2, 3)).to eq(6)
  end
end
```