Basic Matchers

> ==

== is the most commonly-used matcher and asserts equality.

```
describe Array do
  subject { %w(thing1 thing2) }
  it "has two things" do
    subject.length.should == 2
  end
end
```

Alternatively, you can use eq.

```
describe Array do
  subject { %w(thing1 thing2) }
  it "has two things" do
    subject.length.should eq(2)
  end
end
```

> =~

=~ asserts with a regular expression match.

```
describe User, ".build_with_default_ssn" do
  subject { User.build_with_default_ssn }

  it "generates a valid SSN" do
    subject.ssn.should =~ /^\d{3}-\d{2}-\d{4}$/
  end
end
```

Alternatively, you can use match.

```
describe User, ".build_with_default_ssn" do
  subject { User.build_with_default_ssn }

  it "generates a valid SSN" do
    subject.ssn.should match(/^\d{3}-\d{2}-\d{4}$/)
  end
end
```

Other Common Matchers

> include

include asserts that a value is present in the subject.

```
describe "dynamics of include" do
  it "works with arrays" do
    [1, 2, 3].should include(1)
  end

  it "works with strings" do
    "string".should include("str")
  end

  it "works with hashes" do
    { :foo => :bar }.should include(:foo)
  end
end
```

> be_true

be_true asserts truthiness.

```
describe User, "with admin privileges" do
  subject { User.new(:privileges => [:admin]) }

  it "knows he is an admin" do
    subject.admin?.should be_true
  end
end
```

> be_false

be_false asserts falsiness.

```
describe User, "without admin privileges" do
  subject { User.new(:privileges => []) }

  it "knows he is not an admin" do
    subject.admin?.should be_false
  end
end
```

Exceptions

> raise_error

In exceptional situations, you'll want to raise in Ruby. Testing can be done with `lambda` or `expect`, passing a block, and then calling `raise_error`. The class and error message are optional.

```
describe SocialNetworkNotifier, "when the network is down" do
  before        { fake_network_outage }
  let(:message) { MessageBuilder.generate }

  it "raises when contacting Twitter fails" do
    lambda {
      SocialNetworkNotifier.notify_twitter_with(message)
    }.should raise_error(SocialNetworkNotifier::NetworkUnreachableError,
                         "Unable to contact Twitter.")
  end

  it "raises when contacting Facebook fails" do
    expect {
      SocialNetworkNotifier.notify_facebook_with(message)
    }.to raise_error(SocialNetworkNotifier::NetworkUnreachableError, /Facebook/)
  end
end
```

`raise_error` also works with `should_not`.

```
describe SocialNetworkNotifier, "when the network is working" do
  before        { fake_network_success }
  let(:message) { MessageBuilder.generate }

  it "does not raise when contacting Twitter" do
    lambda {
      SocialNetworkNotifier.notify_twitter_with(message)
    }.should_not raise_error
  end

  it "does not raise when contacting Facebook" do
    expect {
      SocialNetworkNotifier.notify_facebook_with(message)
    }.to_not raise_error(SocialNetworkNotifier::NetworkUnreachableError)
  end
end
```

Predicates

RSpec is kind enough to match 'huh' methods (methods that end with a ?, which act as propositions) with appropriate matchers that read more easily.

```
describe "predicates" do
  subject { %w(thing1 thing2) }

  it "works with all Ruby objects" do
    subject.empty?.should be_false
    subject.should be_empty
  end
end
```

Predicates work on any Ruby object and any 'huh' method.

```
describe "more predicates" do
  subject { User.new(:privileges => [:admin]) }

  it "works with all Ruby objects" do
    subject.admin?.should be_true
    subject.should be_admin
  end
end
```