

nil, boolean, equality, type, regex matchers

```
expect(user).to be_nil
expect (foo.bar?).to be_truthy
expect (foo.bar?).to be_falsey
expect (user.id).to eq("foo")
expect (user).to be_a(FOOAP: :User)
expect (array).to be_an(Array)
expect (user.id).to match(/foo/)
```

💡 Using *not* instead of *to* reverses the condition.

Collection matchers

```
expect([1, 2, 3]).to include(2)
expect (request.headers).to include("X-foo")
=> "bar", "X-Bar" => "qux"
expect (request.headers).to have_key("X-foo")
expect([1, 2, 3]).to all(be_a(Fixnum))
```

Message expectations

```
expect(FooClass).to receive(:bar).and_return(:baz)
expect (foo).to receive(:bar).and_return(:baz)
expect (foo).to receive(:bar).with(:qux) {
  Baz.new }
# equivalent to previous example
allow (foo).to receive(:bar) { Baz.new }
expect (foo).to have_received(:bar).with(:qux)
# use sparingly :)
expect _any_instance_of(FooClass).to
receive(:bar) { true }
```

💡 *expect* will fail if the message is not received. *allow* is a relaxed version and will not fail.

Exceptions

```
expect { Foo.find(-1) }.to
raise_error(ActiveResource::ResourceNotFound)
expect { Foo.find(1) }.not_to raise_error
```

Rails models

```
expect(foo).to be_valid
expect (bar).not_to be_valid
expect (bar.errors.messages).to have_key(:baz)
```

Rails controllers and routing

```
get :index
expect (response).to render_template("index")
# if routes.rb contains
get "/foo" => "foo #method"
# then test it with
expect (get("/foo?bar=baz")).to route_to(controller: "foo", bar: "baz")
```

Rails views

```
assign(:foo, stub_model(Foo, bar: "baz"))
render
expect (rendered).to match(/baz/)
```

Methods

```
expect(foo).to respond_to(:bar)
```

Other

Define a memoized function to run at the start of each example

```
let(:baz) { double(Baz, qux: "quux") }
```

Now, you can say

```
allow (foo).to receive(:bar).and_return(baz)
```

and if you reuse *baz* in your test, you can be sure it's the same thing.