```
var = boolean? ? foo : bar
```

Finally, it's often convenient to use the ternary operator in a function's return value:

```
def foo
  do_stuff
  boolean? ? "bar" : "baz"
end
```

Since Ruby implicitly returns the value of the last expression in a function, here the `foo` method returns `"bar"` or `"baz"` depending on whether `boolean?` is `true` or `false`.

Exercises

Solutions to the exercises are available to all Rails Tutorial purchasers here.

To see other people's answers and to record your own, subscribe to the Rails Tutorial course or to the Learn Enough All Access Bundle.

1. By inspecting your browser's cookies directly, verify that the "remember me" checkbox is having its intended effect.

2. At the console, invent examples showing both possible behaviors of the ternary operator (Box 9.2).

9.3 Remember tests

Although our "remember me" functionality is now working, it's important to write some tests to verify its behavior. One reason is to catch implementation errors, as discussed in a moment. Even more important, though, is that the core user persistence code is in fact completely untested at present. Fixing these

issues will require some trickery, but the result will be a far more powerful test suite.

9.3.1 Testing the "remember me" box

When I originally implemented the checkbox handling in Listing 9.23, instead of the correct

```
params[:session][:remember_me] == '1' ? remember(user) : forget(user)
```

I actually used

```
params[:session][:remember_me] ? remember(user) : forget(user)
```

In this context, **`params[:session][:remember_me]`** is either **`'0'`** or **`'1'`**, both of which are **`true`** in a boolean context, so the resulting expression is *always true*, and the application acts as if the checkbox is always checked. This is exactly the kind of error a test should catch.

Because remembering users requires that they be logged in, our first step is to define a helper to log users in inside tests. In Listing 8.24, we logged a user in using the **`post`** method and a valid **`session`** hash, but it's cumbersome to do this every time. To avoid needless repetition, we'll write a helper method called **`log_in_as`** to log in for us.

Our method for logging a user in depends on the type of test. Inside controller tests, we can manipulate the **`session`** method directly, assigning **`user.id`** to the **`:user_id`** key (as first seen in Listing 8.14):

```
def log_in_as(user)
  session[:user_id] = user.id
end
```

We call the method **`log_in_as`** to avoid any confusion with the application code's **`log_in`** method as defined in Listing 8.14. Its location is in the **`ActiveSupport::TestCase`** class inside the **`test_helper`** file, the same location as the **`is_logged_in?`** helper from Listing 8.30:

```
class ActiveSupport::TestCase
  fixtures :all

  # Returns true if a test user is logged in.
  def is_logged_in?
    !session[:user_id].nil?
  end

  # Log in as a particular user.
  def log_in_as(user)
    session[:user_id] = user.id
  end
end
```

We won't actually need this version of the method in this chapter, but we'll put it to use in Chapter 10.

Inside integration tests, we can't manipulate **`session`** directly, but we can **`post`** to the sessions path as in Listing 8.24, which leads to the **`log_in_as`** method shown here:

```
class ActionDispatch::IntegrationTest

  # Log in as a particular user.
  def log_in_as(user, password: 'password', remember_me: '1')
    post login_path, params: { session: { email: user.email,
                                          password: password,
                                          remember_me: remember_me } }
  end
end
```

Because it's located inside the **`ActionDispatch::IntegrationTest`** class, this is the version of **`log_in_as`** that will be called inside integration tests. We use the same method name in both cases because it lets us do things like use code from a controller test in an integration without making any changes to the login method.

Putting these two methods together yields the parallel **`log_in_as`** helpers shown in Listing 9.24.

Listing 9.24: Adding a `log_in_as` helper.
test/test_helper.rb

```
ENV['RAILS_ENV'] ||= 'test'
.
.
.
class ActiveSupport::TestCase
  fixtures :all

  # Returns true if a test user is logged in.
  def is_logged_in?
    !session[:user_id].nil?
  end

  # Log in as a particular user.
  def log_in_as(user)
    session[:user_id] = user.id
  end
end

class ActionDispatch::IntegrationTest

  # Log in as a particular user.
  def log_in_as(user, password: 'password', remember_me: '1')
    post login_path, params: { session: { email: user.email,
                                          password: password,
                                          remember_me: remember_me } }
  end
end
```

Note that, for maximum flexibility, the second `log_in_as` method in Listing 9.24 accepts keyword arguments (as in Listing 7.13), with default values for the password and for the "remember me" checkbox set to `'password'` and `'1'`, respectively.

To verify the behavior of the "remember me" checkbox, we'll write two tests, one each for submitting with and without the checkbox checked. This is easy using the login helper defined in Listing 9.24, with the two cases appearing as

```
log_in_as(@user, remember_me: '1')
```

and

```
log_in_as(@user, remember_me: '0')
```

(Because `'1'` is the default value of `remember_me`, we could omit the corresponding option in the first case above, but I've included it to make the parallel structure more apparent.)

After logging in, we can check if the user has been remembered by looking for the `remember_token` key in the `cookies`. Ideally, we would check that the cookie's value is equal to the user's remember token, but as currently designed there's no way for the test to get access to it: the `user` variable in the controller has a remember token attribute, but (because `remember_token` is virtual) the `@user` variable in the test doesn't. Fixing this minor blemish is left as an exercise (Section 9.3.1), but for now we can just test to see if the relevant cookie is `nil` or not. The results appear in Listing 9.25. (Recall from Listing 8.24 that `users(:michael)` references the fixture user from Listing 8.23.)

Listing 9.25: A test of the "remember me" checkbox. GREEN
test/integration/users_login_test.rb

```
require 'test_helper'

class UsersLoginTest < ActionDispatch::IntegrationTest

  def setup
    @user = users(:michael)
  end
  .
  .
  .
  test "login with remembering" do
    log_in_as(@user, remember_me: '1')
    assert_not_empty cookies[:remember_token]
  end

  test "login without remembering" do
    # Log in to set the cookie.
    log_in_as(@user, remember_me: '1')
    # Log in again and verify that the cookie is deleted.
    log_in_as(@user, remember_me: '0')
    assert_empty cookies[:remember_token]
  end
end
```

Assuming you didn't make the same implementation mistake I did, the tests should be GREEN:

Listing 9.26: GREEN

```
$ rails test
```

Exercises

Solutions to the exercises are available to all Rails Tutorial purchasers here.

To see other people's answers and to record your own, subscribe to the Rails Tutorial course or to the Learn Enough All Access Bundle.

1. As mentioned above, the application currently doesn't have any way to access the virtual **`remember_token`** attribute in the integration test in Listing 9.25. It is possible, though, using a special test method called **`assigns`**. Inside a test, you can access *instance* variables defined in the controller by using **`assigns`** with the corresponding symbol. For example, if the **`create`** action defines an **`@user`** variable, we can access it in the test using **`assigns(:user)`**. Right now, the Sessions controller **`create`** action defines a normal (non-instance) variable called **`user`**, but if we change it to an instance variable we can test that **`cookies`** correctly contains the user's remember token. By filling in the missing elements in Listing 9.27 and Listing 9.28 (indicated with question marks **`?`** and **`FILL_IN`**), complete this improved test of the "remember me" checkbox.

Listing 9.27: A template for using an instance variable in the **`create`** action.
app/controllers/sessions_controller.rb

```
class SessionsController < ApplicationController

  def new
  end

  def create
    ?user = User.find_by(email: params[:session][:email].downcase)
```

```
    if ?user && ?user.authenticate(params[:session][:password])
      log_in ?user
      params[:session][:remember_me] == '1' ? remember(?user) : forget(?user)
      redirect_to ?user
    else
      flash.now[:danger] = 'Invalid email/password combination'
      render 'new'
    end
  end

  def destroy
    log_out if logged_in?
    redirect_to root_url
  end
end
```

Listing 9.28: A template for an improved "remember me" test. GREEN
test/integration/users_login_test.rb

```
require 'test_helper'

class UsersLoginTest < ActionDispatch::IntegrationTest

  def setup
    @user = users(:michael)
  end
  .
  .
  .
  test "login with remembering" do
    log_in_as(@user, remember_me: '1')
    assert_equal FILL_IN, assigns(:user).FILL_IN
  end

  test "login without remembering" do
    # Log in to set the cookie.
    log_in_as(@user, remember_me: '1')
    # Log in again and verify that the cookie is deleted.
    log_in_as(@user, remember_me: '0')
    assert_empty cookies[:remember_token]
  end
  .
  .
  .
end
```

9.3.2 Testing the remember branch

In Section 9.1.2, we verified by hand that the persistent session implemented in the preceding sections is working, but in fact the relevant branch in the **`current_user`** method is currently completely untested. My favorite way to handle this kind of situation is to raise an exception in the suspected untested block of code: if the code isn't covered, the tests will still pass; if it is covered, the resulting error will identify the relevant test. The result in the present case appears in Listing 9.29.

Listing 9.29: Raising an exception in an untested branch. GREEN
app/helpers/sessions_helper.rb

```
module SessionsHelper
  .
  .
  .
  # Returns the user corresponding to the remember token cookie.
  def current_user
    if (user_id = session[:user_id])
      @current_user ||= User.find_by(id: user_id)
    elsif (user_id = cookies.signed[:user_id])
      raise       # The tests still pass, so this branch is currently untested.
      user = User.find_by(id: user_id)
      if user && user.authenticated?(cookies[:remember_token])
        log_in user
        @current_user = user
      end
    end
  end
  .
  .
  .
end
```

At this point, the tests are GREEN:

Listing 9.30: GREEN

```
$ rails test
```

This is a problem, of course, because the code in Listing 9.29 is broken. Moreover, persistent sessions are cumbersome to check by hand, so if we ever want

to refactor the `current_user` method (as we will in Chapter 11) it's important to test it.

Because both versions of the `log_in_as` helper method defined in Listing 9.24 automatically set `session[:user_id]` (either explicitly or by posting to the login path), testing the "remember" branch of the `current_user` method is difficult in an integration test. Luckily, we can bypass this restriction by testing the `current_user` method directly in a Sessions helper test, whose file we have to create:

```
$ touch test/helpers/sessions_helper_test.rb
```

The test sequence is simple:

1. Define a `user` variable using the fixtures.
2. Call the `remember` method to remember the given user.
3. Verify that `current_user` is equal to the given user.

Because the `remember` method doesn't set `session[:user_id]`, this procedure will test the desired "remember" branch. The result appears in Listing 9.31.

Listing 9.31: A test for persistent sessions. RED
test/helpers/sessions_helper_test.rb

```
require 'test_helper'

class SessionsHelperTest < ActionView::TestCase

  def setup
    @user = users(:michael)
    remember(@user)
  end

  test "current_user returns right user when session is nil" do
    assert_equal @user, current_user
    assert is_logged_in?
  end
```

```
  test "current_user returns nil when remember digest is wrong" do
    @user.update_attribute(:remember_digest, User.digest(User.new_token))
    assert_nil current_user
  end
end
```

Note that we've added a second test, which checks that the current user is **`nil`** if the user's remember digest doesn't correspond correctly to the remember token, thereby testing the **`authenticated?`** expression in the nested **`if`** statement:

```
if user && user.authenticated?(cookies[:remember_token])
```

Incidentally, in Listing 9.31 we could write

```
assert_equal current_user, @user
```

instead, and it would work just the same, but (as mentioned briefly in Section 5.3.4) the conventional order for the arguments to **`assert_equal`** is *expected*, *actual*:

```
assert_equal <expected>, <actual>
```

which in the case of Listing 9.31 gives

```
assert_equal @user, current_user
```

With the code as in Listing 9.31, the test is RED as required:

Listing 9.32: RED

```
$ rails test test/helpers/sessions_helper_test.rb
```

We can get the tests in Listing 9.31 to pass by removing the **`raise`** and restoring the original **`current_user`** method, as shown in Listing 9.33.

Listing 9.33: Removing the raised exception. GREEN
app/helpers/sessions_helper.rb

```
module SessionsHelper
  .
  .
  .
  # Returns the user corresponding to the remember token cookie.
  def current_user
    if (user_id = session[:user_id])
      @current_user ||= User.find_by(id: user_id)
    elsif (user_id = cookies.signed[:user_id])
      user = User.find_by(id: user_id)
      if user && user.authenticated?(cookies[:remember_token])
        log_in user
        @current_user = user
      end
    end
  end
  .
  .
  .
end
```

At this point, the test suite should be GREEN:

Listing 9.34: GREEN

```
$ rails test
```

Now that the "remember" branch of **`current_user`** is tested, we can be confident of catching regressions without having to check by hand.

Exercises

Solutions to the exercises are available to all Rails Tutorial purchasers here.

To see other people's answers and to record your own, subscribe to the Rails Tutorial course or to the Learn Enough All Access Bundle.

1. Verify by removing the **`authenticated?`** expression in Listing 9.33 that the second test in Listing 9.31 fails, thereby confirming that it tests the right thing.