Object Oriented WEB Programming Using Ruby

Day 10: 15/June/2017

Test Script writing and
Test running / First Coding

Share ToDo/Done with others

Use Markdown notation.

How to write? See attached Document in the lecture slides folder.

How to view the file in the LINUX environment? There is a Firefox add-on.

What is Test Driven Development

TDD: Test Driven Development Definition

"Test-driven development" refers to a style of programming in which three activities are tightly interwoven: coding, testing (in the form of writing unit tests) and design (in the form of refactoring).

https://www.agilealliance.org/glossary/tdd/

TDD Procedure

It can be succinctly described by the following set of rules:

- write a "single" unit test describing an aspect of the program
- run the test, which should fail because the program lacks that feature
- write "just enough" code, the simplest possible, to make the test pass
- "refactor" the code until it conforms to the simplicity criteria
- repeat, "accumulating" unit tests over time

RSpec : BDD

Now Behavior Driven Development

RSpec: Behaviour Driven Development for Ruby. Making TDD Productive and Fun.

http://rspec.info/

RSpec

RSpec is testing tool for the Ruby programming language. Born under the banner of Behaviour-Driven Development, it is designed to make Test-Driven Development a productive and enjoyable experience with features listed in the next page.

Ref: http://rspec.info/

See details in the above site.

RSpec features:

- a rich command line program (the rspec command)
- textual descriptions of examples and groups (rspec-core)
- flexible and customizable reporting
- extensible expectation language (rspec-expectations)
- built-in mocking/stubbing framework (rspec-mocks)

Why we learn RSpec now?

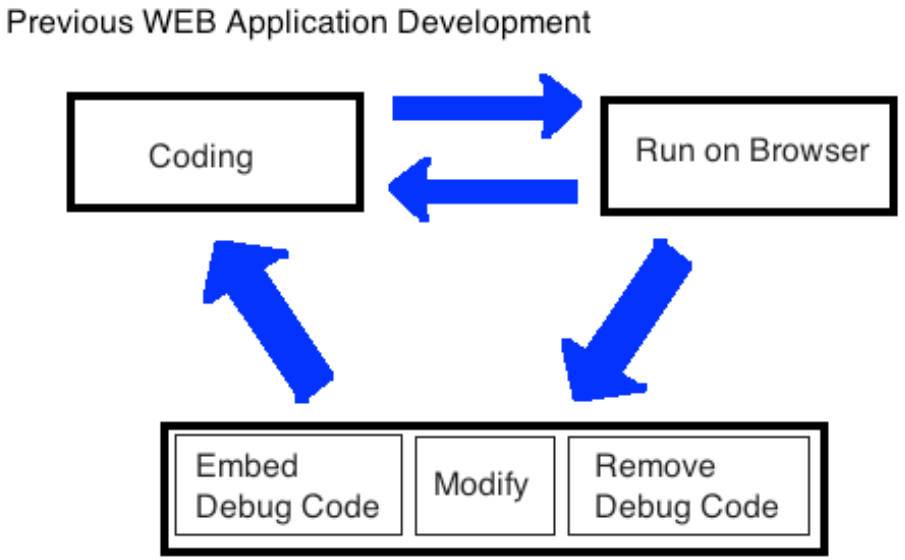
Because, there is a principle for Ruby on Rails programmer, as

"Until we fails the test, we should not write any product code!"

What does it mean?

Conventional WEB application Development

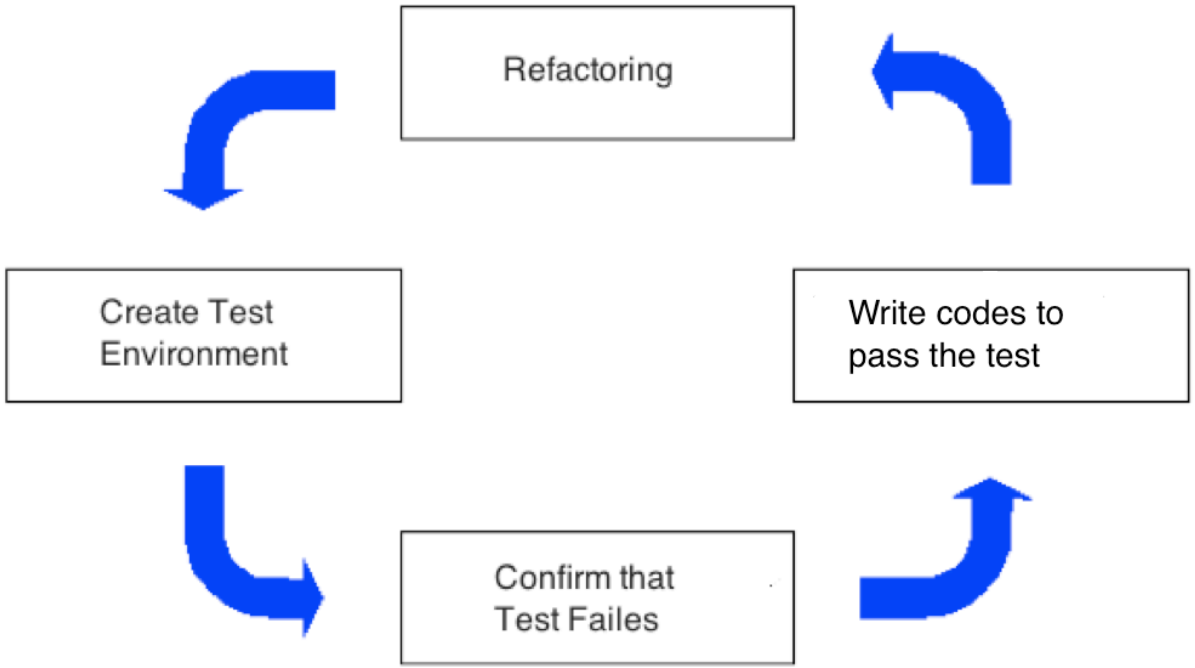
- We had written the code, run, and checked with its actual runtime environment.



P304, Kiso Ruby on Rails, Impress Japan, 2007

Procedure of TDD

- Create Test Environment First.



P304, Kiso Ruby on Rails, Impress Japan, 2007

In Test, “Failure” has meaning

- We prepare “Test” before we start writing programs. This will become ‘specification.’
- We Test first before we write program, the test should “Fail” and that proves the “Test works properly.”
- Here the “Failure” is not an “Error”

4 steps to introduce Test

- Step 1: Write "Test"
  - Make Specification Clear, and write "Test" according to how it should work.
- Step 2: Confirm it "fails" before writing program.
  - Prepare "Test Script" and execute test to prove it works before writing programs. (Debug the test script.)
- Step 3: Coding
  - So that the program passes the test
- Step 4: Refactoring
  - Keep it passes the test, and clean the source code.

Let us introduce RSpec

Now type the following command in the GNOME terminal;

gem install rspec-rails

If you get version display by typing the following command, RSpec is successfully installed.

rspec –v

```
[root@cisnote myshop]# rspec -v
2.13.1
[root@cisnote myshop]#
```

Add RSpec to Gemfile

Open (projects)/Gemfile, then add the following scripts;

```
group :test, :development do
    gem "rspec-rails"
    gem "rails-controller-testing"
    gem "factory_girl_rails"
    gem "capybara"
    gem "database_cleaner"
end
```

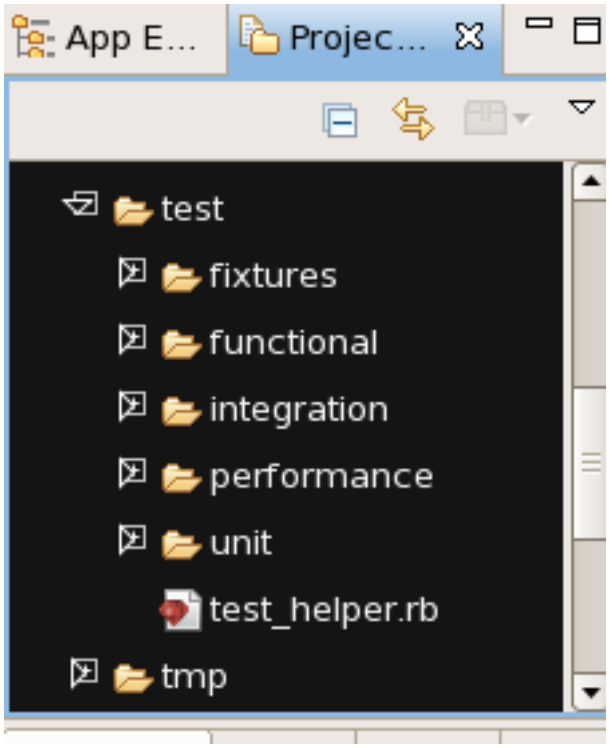
bundle install

Type the following as usual, after you edit the Gemfile

bundle install

Test folder and the files

You would see ‘test’ folder in the project directory, however they are all for Test::Unit and we will not use those files.



What have we installed?

RSpec – test automation suite

FactoryGirl – Generation of Test Data

Capybara – Web screen input emulator

DatabaseCleaner – Test Data remover

https://github.com/thoughtbot/factory_girl

https://github.com/teamcapybara/capybara

Prepare RSpec environment

Type the following command to prepare RSpec environment

rails g rspec:install

```
Running via Spring preloader in process 1550
      create  .rspec
      create  spec
      create  spec/spec_helper.rb
      create  spec/rails_helper.rb
          )—MacBook:ScopsOwl
```

Now we catch up!

We catch up with the principle;

> "We should not write any code until we had failed the test."

Prepare rspec for commodity model! Type

rails g rspec:model ik_memo

```
$ rails g rspec:model ik_memo
Running via Spring preloader in process 1571
      create  spec/models/ik_memo_spec.rb
      invoke  factory_girl
      create    spec/factories/ik_memos.rb
```

Rspec file for Views

Now then we generate test for views;

rails g rspec:view ik_memos index

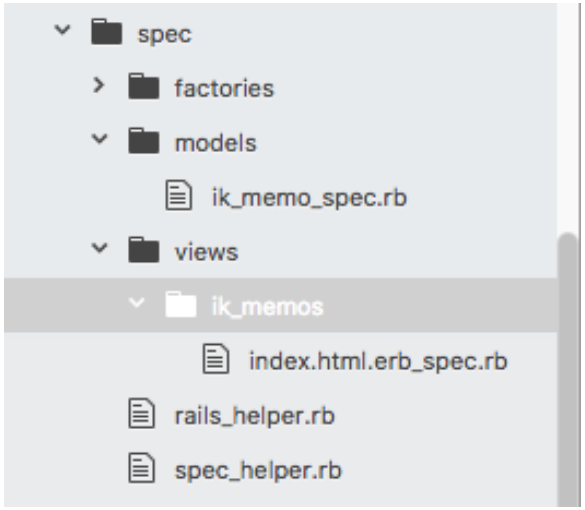
Now we get

ik_memo_spec.rb

and

index.html.erb_spec.rb

rspec files.

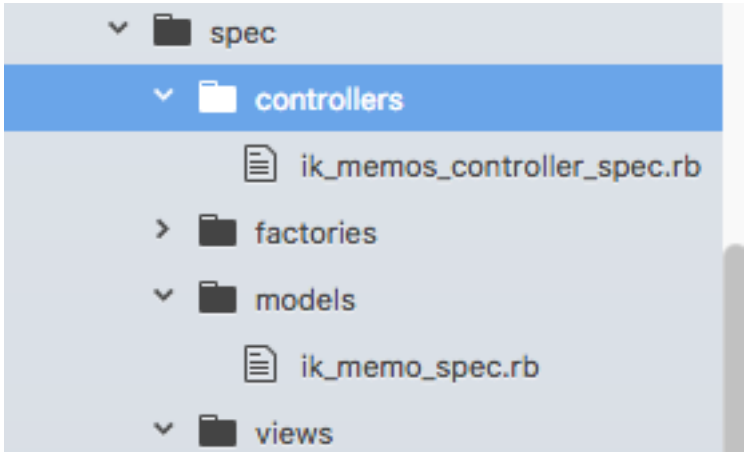



```
$ rails g rspec:view ik_memos index
Running via Spring preloader in process 1586
      create  spec/views/ik_memos
      create  spec/views/ik_memos/index.html.erb_spec.rb
```

Rspec file for controller

For controller, type

rails g rspec:controller ik_memos




```
Running via Spring preloader in process 1604
      create  spec/controllers/ik_memos_controller_spec.rb
```

RSpec's way of thinking…

Before you write a program code, we should write a specification first in principle.

First, we describe the program's behaviour as an executable sample.

Here, the group of logical test cases is 'behaviour,' and the verification of the code is called as 'expectation.'

Preparation to use FactoryGirl

in spec directory, there is a file
rails_helper.rb

Uncomment the following line (near line23):

```
Dir[Rails.root.join('spec/support/**/
  *.rb')].each { |f| require f }
```

Then, add the following:

```
config.include FactoryGirl::Syntax::Methods
```

and

```
require 'capybara/rails'
```

Spec/rails_helper.rb

```ruby
# This file is copied to spec/ when you run 'rails generate rspec:install'
require 'spec_helper'
ENV['RAILS_ENV'] ||= 'test'
require File.expand_path('../../config/environment', __FILE__)
# Prevent database truncation if the environment is production
abort("The Rails environment is running in production mode!") if Rails.env.pro
require 'rspec/rails'
require 'capybara/rails'
# Add additional requires below this line. Rails is not loaded until this poin

# Requires supporting ruby files with custom matchers and macros, etc, in
# spec/support/ and its subdirectories. Files matching `spec/**/*_spec.rb` are
# run as spec files by default. This means that files in spec/support that end
# in _spec.rb will both be required and run as specs, causing the specs to be
# run twice. It is recommended that you do not name files matching this glob t
# end with _spec.rb. You can configure this pattern with the --pattern
# option on the command line or in ~/.rspec, .rspec or `.rspec-local`.
#
# The following line is provided for convenience purposes. It has the downside
# of increasing the boot-up time by auto-requiring all files in the support
# directory. Alternatively, in the individual `*_spec.rb` files, manually
# require only the support files necessary.
#
Dir[Rails.root.join('spec/support/**/*.rb')].each { |f| require f }

# Checks for pending migration and applies them before tests are run.
# If you are not using ActiveRecord, you can remove this line.
ActiveRecord::Migration.maintain_test_schema!

RSpec.configure do |config|
  config.include FactoryGirl::Syntax::Methods
  # Remove this line if you're not using ActiveRecord or ActiveRecord fixtures
  config.fixture_path = "#{::Rails.root}/spec/fixtures"
```

Prepare Database Cleaner

In

spec/spec_helper.rb

Add the following in the configure block;

```
config.before(:suite) do
  DatabaseCleaner.strategy = :truncation
end
config.before(:each) do
  DatabaseCleaner.start
end
config.after(:each) do
  DatabaseCleaner.clean
end
```

Spec/spec_helper.rb

```
15  # See http://rubydoc.info/gems/rspec-core/RSpec/Core/Configuration
16  RSpec.configure do |config|
17
18    config.before(:suite) do
19      DatabaseCleaner.strategy = :truncation
20    end
21    config.before(:each) do
22      DatabaseCleaner.start
23    end
24    config.after(:each) do
25      DatabaseCleaner.clean
26    end
27
28    # rspec-expectations config goes here. You can use an alternate
29    # assertion/expectation library such as wrong or the stdlib/minitest
30    # assertions if you prefer.
31    config.expect_with :rspec do |expectations|
32      # This option will default to `true` in RSpec 4. It makes the `desc
```

Prepare to use test table

Rails migration generate database table for "Development" purpose, not for "Test" purpose. So type the following command:

```
rake db:migrate RAILS_ENV=test
```

When we complete with the test, we should set it back to the development mode;

```
rake db:migrate RAILS_ENV=development
```

rake db:migrate RAILS_ENV=test

```
$ rake db:migrate RAILS_ENV=test
== 20170526050804 CreateUsers: migrating ======================================
-- create_table(:users)
   -> 0.0023s
== 20170526050804 CreateUsers: migrated (0.0024s) =============================

== 20170526053634 AddDeviseToUsers: migrating =================================
-- change_table(:users)
   -> 0.0044s
-- add_index(:users, :email, {:unique=>true})
   -> 0.0012s
-- add_index(:users, :reset_password_token, {:unique=>true})
   -> 0.0009s
== 20170526053634 AddDeviseToUsers: migrated (0.0067s) ========================

== 20170526105959 CreateSmplChats: migrating ==================================
-- create_table(:smpl_chats)
   -> 0.0009s
== 20170526105959 CreateSmplChats: migrated (0.0010s) =========================

== 20170607081542 CreateIkMemos: migrating ====================================
-- create_table(:ik_memos)
   -> 0.0012s
== 20170607081542 CreateIkMemos: migrated (0.0012s) ===========================

== 20170607094136 CreateIkCategories: migrating ===============================
-- create_table(:ik_categories)
   -> 0.0013s
== 20170607094136 CreateIkCategories: migrated (0.0014s) ======================

== 20170607094228 AddIkCategoryIdToIkMemos: migrating =========================
-- add_column(:ik_memos, :ik_category_id, :integer)
   -> 0.0010s
== 20170607094228 AddIkCategoryIdToIkMemos: migrated (0.0116s) ================
```

how to write spec file

```
describe Array, "when empty" do
   before do
        @empty_array = []
   end
   it "should be empty" do
        @empty_array.should be_empty
   end
   after do
        @empty_array = nil
   end
end
```

We write RED part and BLUE part in the left.

RED parts are variables, methods, and conditional statements.

Blue parts are mainly for English, which comes between it and and, so that we can easily READ the specs.

Then, the question is …

How to write those spec files!

See ‘good rspec file samples,’ in here

http://betterspecs.org/

You are strongly recommended to read through the above site!

Before we go forward…

Please read the above site carefully,

as the principles for an ‘advanced’ Ruby on Rails programmer.

Here we have one thing more…

That is ‘Factory Girl.’

What is Factory Girl?

factory_girl is a fixtures replacement with a straightforward definition syntax, support for multiple build strategies (saved instances, unsaved instances, attribute hashes, and stubbed objects), and support for multiple factories for the same class (user, admin_user, and so on), including factory inheritance.

from

https://github.com/thoughtbot/factory_girl

In short, with factory_girl

we can easily build the test data(fixtures) and strategies for the test specification description.

This will replace rails fixtures.

How to use Factory Girl

Let us see this site;

https://github.com/thoughtbot/factory_girl/blob/master/GETTING_STARTED.md

Spec/factories/ik_memos.rb

Each factory has a name and a set of attributes.
The name is used to guess the class of the object by default, but it's possible to explicitly specify it:

```
FactoryGirl.define do
  factory :ik_memos, class: IkMemo do
    factory :memo1 do
      content "Rainy season started."
      ik_category_id 1
    end

    factory :memo2 do
      content "Meet with John, at Mitaka, 13:00 June 20."
      ik_category_id 2
    end
    factory :memo2_wrong, parent: :memo2 do
      ik_category_id 0
    end
    factory :memo2_new, parent: :memo2 do
      content "Meet with John, at Mitaka, 19:00 Jan 20."
    end
  end
end
```

```
FactoryGirl.define do
  factory :category_1 do
    id 1
    name "Idea"
  end
  factory :category_2 do
    id 2
    name "Meet with friend"
  end
end
```

Spec/factories/ik_memos.rb

```
FactoryGirl.define do
  factory :ik_memos, class: IkMemo do
    factory :memo1 do
      content "Rainy season started."
      ik_category_id 1
    end

    factory :memo2 do
      content "Meet with John, at Mitaka, 13:00 June 20."
      ik_category_id 2
    end
    factory :memo2_wrong, parent: :memo2 do
      ik_category_id 0
    end
    factory :memo2_new, parent: :memo2 do
      content "Meet with John, at Mitaka, 19:00 Jan 20."
    end
  end
end
```

Spec/factories/ik_memos.rb

```ruby
FactoryGirl.define do
  factory :ik_memos, class: IkMemo do
    factory :memo1 do
      content "Rainy season started."
      ik_category_id 1
    end

    factory :memo2 do
      content "Meet with John, at Mitaka, 13:00 June 20."
      ik_category_id 2
    end
    factory :memo2_wrong, parent: :memo2 do
      ik_category_id 0
    end
    factory :memo2_new, parent: :memo2 do
      content "Meet with John, at Mitaka, 19:00 Jan 20."
    end
  end
end

```

How to use factories

factory_girl supports several different build strategies: build, create, attributes_for and build_stubbed:

```ruby
# Returns a User instance that's not saved
user = FactoryGirl.build(:user)

# Returns a saved User instance
user = FactoryGirl.create(:user)

# Returns a hash of attributes that can be used to build a User instance
attrs = FactoryGirl.attributes_for(:user)

# Returns an object with all defined attributes stubbed out
stub = FactoryGirl.build_stubbed(:user)

# Passing a block to any of the methods above will yield the return object
FactoryGirl.create(:user) do |user|
  user.posts.create(attributes_for(:post))
end
```

spec/controllers/ memos_controller_spec.rb (1/2)

```
require 'rails_helper'

RSpec.describe IkMemosController, type: :controller do
  before(:all) do
    @c_idea = FactoryGirl.create(:c_idea)
    @c_meet = FactoryGirl.create(:c_meet)
    @memo1 = FactoryGirl.create(:memo1)
    @memo2 = FactoryGirl.create(:memo2)
  end
  after(:all) do
    DatabaseCleaner.clean
  end

  describe '#index' do
    it "displays using index template" do
      get :index
      expect(response).to render_template :index
    end
  end
```

spec/controllers/ memos_controller_spec.rb (2/2)

```
 describe "#update" do
    context( "when category_id is null, ") do
      it "will not be updated, and return to edit." do
        get :edit, params: { id: @memo2 }
        patch :update, params: { id: @memo2, ik_memo: attributes_for(:memo2_wrong) }
        expect(response).to render_template :edit
      end
    end
    context( "when category_id is not null, ") do
      it "will be updated" do
        get :edit, params: { id: @memo2 }
        patch :update, params: { id: @memo2, ik_memo: attributes_for(:memo2_new) }
        @memo2.reload
        expect(@memo2.ik_category_id).to eq(2)
        expect(@memo2.content).to eq("Meet with John, at Mitaka, 19:00 Jan 20.")
       end
      it "will redirect, after updated, " do
        get :edit, params: { id: @memo2 }
        patch :update, params: { id: @memo2, ik_memo: attributes_for(:memo2_new) }
        expect(response).to redirect_to action: :show, id: @memo2.id
      end
    end
  end

end
```

Spec/views/ik_memos/ index.html.erb_spec.rb

```
require 'rails_helper'

RSpec.describe "ik_memos/index", type: :view do
  feature 'in the ik_memos/index screen,' do
    scenario 'When index is displayed, database creation
    message should appear.' do
      visit ik_memos_path    # Ik Memos index page
      # click_on (t :click_here )     # click Click Here link
      expect(page).to have_content 'This is a sample page of
    database creation.'
    end
  end

end
```

Perform Test

Run automated test by typing the following command

```
rspec spec/*/*_spec.rb spec/views/*/*_spec.rb
```

Perform Test

Run automated test by typing the following command

rspec spec/*/*_spec.rb spec/views/*/*_spec.rb

```
***.

ing: (Failures listed here are expected and do not affect your suite's status)

 IkCategory add some examples to (or delete) /Users/kobayashi_ikuo/Documents/atom.worksp
opsOwl/spec/models/ik_category_spec.rb
 # Not yet implemented
 # ./spec/models/ik_category_spec.rb:4

 IkMemo add some examples to (or delete) /Users/kobayashi_ikuo/Documents/atom.workspace/
wl/spec/models/ik_memo_spec.rb
 # Not yet implemented
 # ./spec/models/ik_memo_spec.rb:4

 ik_categories/index add some examples to (or delete) /Users/kobayashi_ikuo/Documents/at
kspace/ScopsOwl/spec/views/ik_categories/index.html.erb_spec.rb
 # Not yet implemented
 # ./spec/views/ik_categories/index.html.erb_spec.rb:4


shed in 0.66656 seconds (files took 5.58 seconds to load)
amples, 0 failures, 3 pending
```

Find some examples…

Please try find the RSpec and factory_girl examples, such site as

http://blog.thefrontiergroup.com.au/2014/12/using-factorygirl-easily-create-complex-data-sets-rails/

Before we write code, we shall write tests,

Before we write tests, we shall define factories,

And, before we need to write rspec and factories, we shall learn much more about the rspec and factories better expressions.

So, please read many many samples as possible about rspec and factory_girl.

RSpec writings

are not ‘must’ in this class.

If you want to try, please write those test codes.

And this may be reflected to the grading.

Now, install your DB and screen.

(If you could, write the test first, then,)

install your Database according to your schema, and design the views.

Now all the technical introduction on Ruby on Rails are completed.

Next, we share codes each other using the tools on which other member had written, or create a link to other members pages.

We introduce sharing ToDo files, and Done.

Show your Test scripts, next week

If you try to write test scripts, demonstrate running rspec.