



Ruby on Rails Training - 0

Oob

Summary: On this second day, you will learn how to make classes and tackle the core of the P.O.O.. If you feel like your code is too wordy, this just means your code is not D.R.Y. enough!

Version: 1.1

Contents

I	Preamble	2
II	General rules	3
III	Specific instructions of the day	4
IV	Exercise 00: HTML	5
V	Exercise 01: Raise HTML	7
VI	Exercise 02: Rescue HTML	8
VII	Exercise 03: Elem	10
VIII	Exercise 04: Dejavu	12
IX	Exercise 05: Validation	15
X	Submission and peer-evaluation	17

Chapter I

Preamble

Besides being great, Ruby is beautiful!

This guide (written by Bozhidar Batsov) was made to be a Ruby programming convention to be used internally in his company. After some time, though, it felt like it could also be interesting to the Ruby community and users, and that the world did not really need any additional internal programming convention. But the world could indeed use a collection of habits, idioms, and style advice for Ruby programming.

As soon as he published this guide, he received a lot of feedback from the exceptional Ruby community all around the world. Thanks for your suggestion and your support! Together, we can create a resource that will benefit all the Ruby developers world wide.

- [Style is what differentiate the good from the excellent.](#)
- The broadly used [Rubocop](#)

Chapter II

General rules

- Your project must be realized in a virtual machine.
- Your virtual machine must have all the necessary software to complete your project. These softwares must be configured and installed.
- You can choose the operating system to use for your virtual machine.
- You must be able to use your virtual machine from a cluster computer.
- You must use a shared folder between your virtual machine and your host machine.
- During your evaluations you will use this folder to share with your repository.
- Your functions should not quit unexpectedly (segmentation fault, bus error, double free, etc) apart from undefined behaviors. If this happens, your project will be considered non functional and will receive a 0 during the evaluation.
- We encourage you to create test programs for your project even though this work **won't have to be submitted and won't be graded**. It will give you a chance to easily test your work and your peers' work. You will find those tests especially useful during your defence. Indeed, during defence, you are free to use your tests and/or the tests of the peer you are evaluating.
- Submit your work to your assigned git repository. Only the work in the git repository will be graded. If Deepthought is assigned to grade your work, it will be done after your peer-evaluations. If an error happens in any section of your work during Deepthought's grading, the evaluation will stop.

Chapter III

Specific instructions of the day

- Every turned-in files will feature a fitting shebang AND the warning flag.
- No code in the global scope. Make functions (or classes)!
- Each turned-in file must feature a series of tests proving it's fully working:

```
$> cat <FILE>.rb
[...]
```

```
if $PROGRAM_NAME == __FILE__
  ## your code here
end
```


- Each file must be tested in an interactive console (irb or pry you choose) as follows:

```
require_relative "file\_name.rb"
>
```

- Imports are prohibited except for the ones specified in the "Authorized functions" section in each exercise cart.

Chapter IV

Exercise 00: HTML

	Exercise 00
Exercise 00: HTML	
Turn-in directory : <i>ex00/</i>	
Files to turn in : ex00.rb	
Allowed functions : n/a	

Make an `Html` class that can create and fill an HTML file. To do so, you will implement:

- A builder that takes in parameter a file name (without extension) that:
 - calls a `head` method
 - gives the file's name to an instance variable `@page_name`
- A `attr_reader` for `@page_name`
- A `Head` method must set a valid `html` followed by a `body` opening hash ahead of the file.
- A "dump" method that takes a string in parameter and sets a `<body>` hash after our string, surrounded by `<p>` hashes.
- A "finish" method that ends the file with `</body>`



Every insertion must be lines.

In a **ruby** console, here is what you must get:

```
> require_relative "ex00.rb"
=> true
> a = Html.new("test")
=> #< Html:0x00000001d71580 @page_name="test" >
> 10.times{|x| a.dump("titi_number#{x}")}
=> 12
> a.finish
=> 7
```


And in our shell:

```
$> cat -e test.html
<!DOCTYPE html>$
<html>$
<head>$
<title>test</title>$
</head>$
<body>$
  <p>titi_number0</p>$
  <p>titi_number1</p>$
  <p>titi_number2</p>$
  <p>titi_number3</p>$
  <p>titi_number4</p>$
  <p>titi_number5</p>$
  <p>titi_number6</p>$
  <p>titi_number7</p>$
  <p>titi_number8</p>$
  <p>titi_number9</p>$
</body>$
```

Like in the example, using a loop to fill one's **MUST** work. The return values in the `irb` or `pry` can change.

Chapter V

Exercise 01: Raise HTML

	Exercise 01
Exercise 01: Raise HTML	
Turn-in directory : <i>ex01/</i>	
Files to turn in : ex01.rb	
Allowed functions : n/a	


For this exercise, you will reuse the code from the ex00 and incorporate error management, raise exceptions when this is required by the behavior, thus avoiding the production of ill or weird html pages. All the following exceptions must be thoroughly replicated, replacing `<filename>` by the name of the file that created the error:

- Creating a file that exists already (identical name) must raise an exception: "A file named `<filename>` already exist!".
- Writing text with `dump` in a file without any body opening tag must raise an exception: "There is no body tag in `<filename>`".
- Writing text with `dump` after a closing body tag must raise the exception: "Body has already been closed in `<filename>`".
- Closing the body with `finish` when the body's closing tag must raise the exception: "`<filename>` has already been closed."

```
> require_relative "ex01.rb"
=> true
> a = Html.new("test")
=> #<Html:0x0000000332b9c0 @page_name='test'>
> a = Html.new("test")
RuntimeError: test.html already exist!
from /ex01.rb:15:in "head"
> a.dump("Lorem_ipsum")
=> nil
> a.finish
=> nil
> a.finish
RuntimeError: test.html has already been closed
from/ex01.rb:39:in "finish"
```


Chapter VI

Exercise 02: Rescue HTML

	Exercise 02
Exercise 02: Rescue HTML	
Turn-in directory : <i>ex02/</i>	
Files to turn in : ex02.rb	
Allowed functions : n/a	

Now that you have problematic behaviors in your hands, let's have a little fun.

In this exercise, you will have to save the processes' execution correcting it with a display specific to errors and solutions to secure the operations.

Take the code you've created in the previous exercise and create two new classes: **Dup_file** and **Body_closed**. They will Inherit the **StandardError** class.

Both these classes will be the new Exceptions you will have to implement in your HTML class and they will have to contain the "**show_state**", "**correct**" and "**explain**" methods. Each will have a very specific role:

- **show_state** will show the state before correction.
- **correct** will correct the error that provoked the **raise**.
- **explain** shows the state after the correction.

Of course, each method will have its own behavior, that you will have to implement. Thus, when you try to create a file that already exists, your code will have to raise the `Dup_file` exception, that will:

- display the list of similar files (with the full PWD).
- create a new file adding '.new' before the extension, as many times as it takes:
test.html , test.new.html , test.new.new.html etc etc...

Exemple :

```
A file named <filename> was already there: /home/desktop/folder_2/<filename>.html  
Appended .new in order to create requested file: /home/desktop/folder_2/<filename>.new.html
```

Besides, if you try to write AFTER a `</body>`, your code will have to raise the `Body_closed` exception that must:


- display the line and its number in the file.
- delete said tag, insert the text and set a closing tag in the end.

Example:

```
In <filename> body was closed :  
> ln :25 </body> : text has been inserted and tag moved at the end of it.
```

Chapter VII

Exercise 03: Elem

	Exercise 03
Exercise 03: Elem	
Turn-in directory : <i>ex03/</i>	
Files to turn in : ex03.rb	
Allowed functions : n/a	

Now, it's time to change method. Your first HTML file engine test is satisfying and promising, but it's time to push the boundaries of the paradigm.

You will create a class to represent your HTML so that a `to_s` method on its instances displays the generated HTML code.

Thus, with its `add_content` method, the `Elem` class is going to be able to contain another `Elem` instance.

This architecture will allow such usage:

```
html = Elem.new(.....)
head = Elem.new(.....)
body = Elem.new(...)
title = Elem.new(Text.new("blah blah"))
head.add_content(title)
html.add_content([head, title, body])
puts html
```


To secure a good implementation, we provide a test file in the `ex03/` folder in the `module02.1.1.tar.gz` tarball included with this subject.

Let's sum it up:

- An **Elem** class with a construction parameter, a tag type, an array of contents, a tag type (orphan or not), and a Hash allowing to implement 'in-tag' infos (src, style, data...).
- A **Text** class that builds itself with a simple **String** as a parameter.
- An **overload** of the **to_s** method.
- The test script execution must **COMPLETELY** pass.

Chapter VIII

Exercise 04: Dejavu

	Exercise 04
Exercise 04: Dejavu	
Turn-in directory : <i>ex04/</i>	
Files to turn in : ex04.rb	
Allowed functions : n/a	

Congratulations! You are now able to generate any kind of **HTML** element and its content. However, it's kinda boring to generate each element specifying its attribute each time, for each instantiation. Here is the opportunity to use the inheritance to make other easier to use small classes.

Make the following classes derivating them from **Elem**:

- **Html**, **Head**, **Body**
- **Title**
- **Meta**
- **Img**
- **Table**
- **Th**, **Tr**, **Td**
- **Ul**, **Ol**, **Li**
- **H1**, **H2**
- **P**
- **Div**
- **Span**

- Hr
- Br

Your code must execute these commands without any error:

```
> puts Html.new([Head.new([Title.new("Hello ground!")]),  
> Body.new([H1.new("Oh no, not again!"),  
> Img.new([], {'src':'http://i.imgur.com/pfp3T.jpg'}) ])])
```

And display:


```
<Html>  
<Head>  
<Title>Hello ground!</Title>  
</Head>  
<Body>  
<H1>Oh no, not again!</H1>  
<Img src='http://i.imgur.com/pfp3T.jpg' />  
</Body>  
</Html>
```



If you feel like the exercise is off-putting, it means there might be another solution.

Chapter IX

Exercise 05: Validation

	Exercise 05
Exercise 05: Validation	
Turn-in directory : <i>ex05/</i>	
Files to turn in : whereto.rb	
Allowed functions : n/a	

Despite showing real progress, you'd like everything to look a little cleaner, a little square. That's who you are: you like constraints and challenges. So why not imposing a norm at the structure of your **HTML** documents? Start by copying the classes of both previous exercises in this exercise's folder.

Create a **Page** class which builder will have to take in parameter an instance of a class inheriting the **Elem**. Your **Page** class must implement a **isvalid()** method which must return **True** if all the following rules are observed and **False** otherwise:

- If, on during the tree path, a node is not a type **html**, **head**, **body**, **title**, **meta**, **img**, **table**, **th**, **tr**, **td**, **ul**, **ol**, **li**, **h1**, **h2**, **p**, **div**, **span**, **hr**, **br** or **Text**, the tree is invalid.
- **Html** must strictly contain a **Head**, **then** a **Body**.
- **Head** must contain a single **Title** and only this **Title**.
- **Body** and **Div** must only contain elements of the following types: **H1**, **H2**, **Div**, **Table**, **Ul**, **Ol**, **Span**, or **Text**.
- **Title**, **H1**, **H2**, **Li**, **Th**, **Td** must only contain a single **Text** and only this **Text**.
- **P** must only contain some **Text**.
- **Span** must only contain some **Text** or **P**.
- **Ul** and **Ol** must contain at least a **Li** and only some **Li**.

- **Tr** must contain at least one **Th** or **Td** and only **Th** or **Td**. The **Th** and the **Td** must be mutually exclusive.
- **Table** must contain **Tr**'s and only **Tr**'s.
- **Img**: must contain a **src** field and its value is a **Text**.

Demonstrate the operation of your **Page** class with tests of your choice. You will run enough tests to cover all the functionalities. For instance, the execution of:

```
if $PROGRAM_NAME == __FILE__
  toto = Html.new([Head.new([Title.new(Text.new("Hello ground!"))]),
    Body.new([H1.new(Text.new("Oh no, not again!")), Img.new([],
      {'src': Text.new('http://i.imgur.com/pfp3T.jpg')} )] )])
  test = Page.new(toto)
  test.is_valid?
  tata = Html.new([Head.new([Title.new(Text.new("Hello ground!"))]),
    Body.new([H1.new(Text.new("Oh no, not again!")), Img.new([],
      {'src': Text.new('http://i.imgur.com/pfp3T.jpg')} )] )])
  test2 = Page.new(tata)
  test2.is_valid?
end
```

MUST display:

```
Currently evaluating a Html :
- root element of type "html"
- Html -> Must contains a Head AND a Body after it
Head is OK
Evaluating a multiple node
Currently evaluating a Text :
-Text -> Must contains a simple string
Text content is OK
Evaluating a multiple node
Currently evaluating a Text :
-Text -> Must contains a simple string
Text content is OK
Currently evaluating a Img :
Img content is OK
      FILE IS OK
```

Chapter X

Submission and peer-evaluation

Turn in your assignment in your `Git` repository as usual. Only the work inside your repository will be evaluated during the defense. Don't hesitate to double check the names of your folders and files to ensure they are correct.



The evaluation process will happen on the computer of the evaluated group.