Ruby on Rails Training - 0

Oob

Summary: On this second day, you will learn how to create classes and delve into the core of O.O.P. If you find your code is too verbose,it is simply imply that your code is not D.R.Y. enough!

Version: 1.2

Contents

Chapter I

Preamble

Besides being great, `Ruby` is beautiful!

This guide (authored by Bozhidar Batsov) was initially developed as an internal Ruby programming convention withing his company. However, over time, it became apparent that it could also be valuable to the wider Ruby community. The world didn't necessarily require another internal programming convention, but it could certainly benefit from a collection of best practices, idioms, and style advice for Ruby programming

Once the guide was published, it received an overwhelming amount of feedback from the exceptional Ruby community worldwide. Thanks for your suggestions and your support! Together, we can create a valuable resource that will benefit Ruby developers worldwide.

- 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.

`For obvious security reasons, any credentials, API keys, env variables etc... must be saved locally in a .env file and ignored by git. Publicly stored credentials will lead you directly to a failure of the project.`

Chapter III

Specific instructions of the day

- All submitted files should include a suitable shebang and the warning flag.
- Avoid placing code in global scope.Instead, use functions (or classes)!
- Each submitted file must include a set of tests to demonstrate its full functionality:

```
$> cat <FILE>.rb
[...]
if $PROGRAM_NAME == __FILE__
  ## your code here
end
```

- Each file should be tested in an interactive console (such as irb or pry) using the following format :

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

- Imports are prohibited unless explicitly allowed in the “Authorized function” section of each exercise description.
- The use of the keyword "for", "while" or "until" is strictly prohibited. Their usage is considered cheating and will result in module failure with a score of -42.

Chapter IV

Exercise 00: HTML

	Exercise 00
Exercise 00: HTML	
Turn-in directory : *ex*00/	
Files to turn in : `ex00.rb`	
Allowed functions : `n/a`	

Make an `Html` class that can create and populate an `HTML` file. To accomplish this, you need to implement the following:

- A builder that takes a file name (without extension) as a parameter, and performs the following tasks:
    - calls a `head` method.
    - Assigns the file's name to an instance variable @`page_name`
- Provides an attr_reader for @`page_name`
- The `Head` method should set a valid `html` tag followed by an opening `body` tag at the beginning of the file.
- A "dump" method that takes a string as a parameter and appends the string, surrounded by `<p>`tag after the <body> tag.
- A "finish" method concludes the file with a closing `</body>` tag.

`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>$
```

Using a loop, as shown in the example, to populate the file MUST work. The return values may vary in irb or pry.

Chapter V

Exercise 01: Raise HTML

	Exercise 01
Exercise 01: Raise HTML	
Turn-in directory : *ex*01/	
Files to turn in : `ex01.rb`	
Allowed functions : `n/a`	

For this exercise, you will reuse the code from the ex00 and incorporate error handling by raising exceptions when necessary to avoid generating invalid or absurd HTML pages. All the following exceptions must be precisely replicated, replacing `<filename>` with the name of the file that caused the error:

- Creating a file with an identical name that already exist should raise an exception: `"A file named <filename> already exists!"`.
- Writing text with the `dump` methode in a file that does not contain an opening `body` tag should raise an exception: `"There is no body tag in <filename>"`.
- Writing text with the `dump` methode after a closing `<body>` tag must raise the exception: `"The body has already been closed in <filename>"`.
- Closing the `<body>` tag with the `finish` methode when the closing `<body>` tag is already present should 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 : *ex*02/	
Files to turn in : `ex02.rb`	
Allowed functions : `n/a`	

Now that you have problematic behaviors in your hands, we are going to take advantage of them.

This exercise demands that you save the execution of the processes by fixing the issues with specific error messages and providing solution to ensure their proper functioning.

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

These two classes are your new Exceptions to be properly implemented in your HTML class. They should contain the methods `"show_state"`, `"correct"` and `"explain"`. Each of these methods has a specific role:

- `show_state` display the state before correction.
- `correct` fixes the error that caused the `raise` statement.
- `explain` displays the state after correction.

Of course, each of these methods has its own behavior, which you need to implement. So, when you try to create a file that already exists, your code raises the `Dup_file` exception, which:

- Displays the list of similar files (with the full PWD).
- Creates a new file by adding '.new' before the extension, multiple times if necessary: test.html , test.new.html , test.new.new.html, 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
```

Furthermore, if you try to write AFTER a `</body>`, your code should raise the `Body_closed` exception, which should:

- Display the line and its line number in the file.
- Remove the mentioned closing tag, insert the text, and add closing tag again.

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 : *ex*03/	
Files to turn in : `ex03.rb`	
Allowed functions : `n/a`	

It is now time to switch methods. Your initial attempt at an `HTML` file engine is successful and promising, but now we are going to push the object paradigm a little further.

You are now creating a class to represent your `HTML` in such a way that a `to_s` method on its instances displays the generated `HTML` code.

Thus, with its `add_content` method, the `Elem` class is capable of having another instance of`Elem` as its content.

This architecture allows for usage like this:

```
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 ensure proper implementation, we provide a test file in the `ex03/` directory withing the `module02.1.1.tar.gz` tarball included with this subject.

Let's recap:

- A `Elem` class with the following construction parameters: a tag type,an array of contents, a tag type (orphan or not), and a Hash that allows us to implement 'in-tag' informations (such as src, style, data, etc.).
- A `Text` class that is constructed with a simple `String` as a parameter.
- An **overloading** of the `to_s` method.
- The execution of the test script must pass **COMPLETELY**.

Chapter VIII

Exercise 04: Dejavu

	Exercise 04
Exercise 04: Dejavu	
Turn-in directory : *ex*04/	
Files to turn in : `ex04.rb`	
Allowed functions : `n/a`	

Congratulations! You are now capable of generating any `HTML` element and its content. However, it's a bit cumbersome to generate each element by specifying every attribute at each instantiation. This is an opportunity to use the inheritance to create other smaller classes that are easier to use.

Create the following classes by deriving 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 should execute these commands without errors:

```
> 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 that the exercise is tedious, there may be another solution.

Chapter IX

Exercise 05: Validation

	Exercise 05
Exercise 05: Validation	
Turn-in directory : *ex*05/	
Files to turn in : `whereto.rb`	
Allowed functions : `n/a`	

Despite making real progress in your work, you would like everything to be a little cleaner, a little more structured. That's just who you are, you enjoy constraints and challenges. So why not impose a standard on the structure of your `HTML` documents? Start by copying the classes from the previous two exercises into the folder for this exercise.

Create a class`Page` whose constructor should take an instance of a class inheriting from `Elem` as a parameter. Your `Page` class should implement a method `isvalid()` that should return `True` if all the following rules are respected, and `False` otherwise:

- If during the tree path, a node is not of 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 contain exactly one `Head`, followed by one `Body`.
- `Head` should contain only one `Title`.
- `Body` and `Div` should only contain elements of the following types: `H1`, `H2`, `Div`,`Table`, `Ul`, `Ol`, `Span` or `Text`.
- `Title`, `H1`, `H2`, `Li`, `Th`, `Td` should only contain one `Text`.
- `P` should only contain `Text` elements.
- `Span` should only contain `Text` or `P` elements.
- `Ul` and `Ol` must contain at least one `Li` and only `Li` elements.

- `Tr` must contain at least one `Th` or `Td` and only `Th` or `Td` elements. `Th` and `Td` should be mutually exclusive.
- `Table` should only contain `Tr` elements.
- `Img`: should have a `src` field and its value should be of type `Text`.

Demonstrate the functionality of your `Page` class with a sufficient number of tests to cover all the features. For example, executing the following:

```
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

Submit your work to your `Git` repository as usual. Only the work present in your repository will be evaluated during the defense. Make sure to check the names of your folders and files to ensure they comply with the requirements of the subject.

`The evaluation will take place on the evaluated group computer.`