



Ruby on Rails Training - 0

Starting

Summary: Let's leave the web domain behind and focus on the [ruby](#) and this language's syntactic and semantic basics.

This module is achievable with Ruby version 3.2

Version: 1.2

Contents

I	Preamble	2
II	General rules	3
III	Today's specific instructions	4
IV	Exercise 00: Classy not classy	5
V	Exercise 01: Breakfast	6
VI	Exercise 02: With Hash browns	7
VII	Exercise 03: Where am I?	9
VIII	Exercise 04: Backward	10
IX	Exercise 05: Hal	11
X	Exercise 06: Wait a minute	12
XI	Exercise 07: elm	13
XII	Submission and peer-evaluation	15

Chapter I

Preamble

Besides being great, Ruby is fun!

- [Poignant Guide To Ruby](#)
- [TryRuby](#)
- [RubyMonk](#)
- [Rubyquizz](#)
- [RubyWarrior](#)

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

Today's specific instructions

- Every submitted file must have an appropriate shebang line and include the warning flag
- Avoid having code in the global scope. Use functions instead!
- Each submitted file should conclude with a function call.
- Imports are not allowed, except for those specified in the "Authorized functions" section of each exercise description.

Chapter IV

Exercise 00: Classy not classy

	Exercise 00
Exercise 00: Classy not classy	
Turn-in directory : <i>ex00/</i>	
Files to turn in : var.rb	
Allowed functions : n/a	


Create a script named `var.rb` in which you will define a `my_var` function. In this function, declare and assign 4 different types variables and print them on the standard output. You must precisely recreate the following output:

```
$> ./var.rb
my variables :
  a contains: 10 and is a type: Integer
  b contains: 10 and is a type: String
  c contains: nil and is a type: NilClass
  d contains: 10.0 and is a type: Float
$>
```

Of course, explicitly stating the variable types in your code prints is **prohibited**. Also, don't forget to call your function at the end of your script as mentioned in the instructions.

Chapter V

Exercise 01: Breakfast

	Exercise 01
Exercise 01: Breakfast	
Turn-in directory : <code>ex01/</code>	
Files to turn in : <code>croissant.rb</code>	
Allowed functions : <code>n/a</code>	

For this exercise, you are free to define as many functions as you like and name them as you see fit.

The tarball named `d01.tar.gz` in the appendix of this subject contains a subfolder named `ex01/`. Inside that folder, you'll find the file `numbers.txt` which contains random numbers from 1 to 100 separated by commas.


Your task is to design a Ruby script named `croissant.rb` that opens the `numbers.txt` file, reads the numbers and displays them on the standard output, one per line, in ascending order without commas.



The command to extract a tarball is: `tar xzf d01.tar.gz`

Chapter VI

Exercise 02: With Hash browns

	Exercise 02
Exercise 02: With Hash browns	
Turn-in directory : <i>ex02/</i>	
Files to turn in : H2o.rb	
Allowed functions : n/a	

Once again, you are free to define as many functions as you like and name them as you see fit. This instruction will not be mentioned again unless it needs to be contradicted.

Create a script named **H2o.rb** and copy the following **data** couples table as it in into one of your functions:


```
data = [['Caleb' , 24],
        ['Calixte' , 84],
        ['Calliste', 65],
        ['Calvin' , 12],
        ['Cameron' , 54],
        ['Camil' , 32],
        ['Camille' , 5],
        ['Can' , 52],
        ['Caner' , 56],
        ['Cantin' , 4],
        ['Carl' , 1],
        ['Carlito' , 23],
        ['Carlo' , 19],
        ['Carlos' , 26],
        ['Carter' , 54],
        ['Casey' , 2]]
```


Write the code that, when executed, declares and converts it into a hash with Integer as the key and the String(s) as the value. Display a message on the console as follows:

```
$> ./H2o.rb  
24 : Caleb  
84 : Calixte  
65 : Calliste  
12 : Calvin  
[...]  
$>
```

Chapter VII

Exercise 03: Where am I?

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

Using the following hashes (you should copy them into a function (we will no longer specify the need to do so):

```
states = {
  "Oregon"   => "OR",
  "Alabama"  => "AL",
  "New Jersey" => "NJ",
  "Colorado" => "CO"
}


capitals_cities = {
  "OR" => "Salem",
  "AL" => "Montgomery",
  "NJ" => "Trenton",
  "CO" => "Denver"
}
```

Write the program that takes “State” (e.g., Oregon) as an argument and displays its capital city in the standard output (e.g., Salem). If the argument does not yield any result, your script should display: **Unknown state**. If there are no arguments or too many arguments, your script should exit without reacting.

```
$> ./Where.rb Oregon
Salem
$> ./Where.rb toto
Unknown state
$> ./Where.rb
$> ./Where.rb Oregon Alabama
$> ./Where.rb Oregon Alabama Ile-De-France
$>
```

Chapter VIII

Exercise 04: Backward


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

You have the same hashes as in the previous exercise. Create a program that takes a capital city as an argument and displays the matching State. Your program must behave identically to the previous exercise.

```
$> ./erehW.rb Salem
Oregon
$> ./erehW.rb toto
Unknown capital city
$> ./erehW.rb
$>
```

Chapter IX

Exercise 05: Hal

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


Using the same hashes as in the **ex03**, write a program similar to previous exercises, with the following modifications:

- The program should take a string containing as an argument, which can contain multiple words separated by commas.
- For each word in the string, the program should detect whether it is a capital city or a State.
- Case sensitivity and spaces should not be considered.
- If there are no parameters or too many parameters, the program should not display anything.
- If there are two consecutive commas in the string, the program should not display anything.
- The program should display the results with each results on a new line using the following specific format:

```
$> ./wheretor.rb "Salem , ,Alabama, Toto , ,MontG0mery"
Salem is the capital of Oregon (akr: OR)
Montgomery is the capital of Alabama (akr: AL)
Toto is neither a capital city nor a state
Montgomery is the capital of Alabama (akr: AL)
$>
```

Chapter X

Exercise 06: Wait a minute

	Exercise 06
Exercise 06: Wait a minute	
Turn-in directory : <i>ex06/</i>	
Files to turn in : CoffeeCroissant.rb	
Allowed functions : n/a	

Using the following table:

```
data = [
  ['Frank', 33],
  ['Stacy', 15],
  ['Juan' , 24],
  ['Dom'  , 32],
  ['Steve', 24],
  ['Jill' , 24]
]
```

Write the code that displays only the names, sorted in ascending order by age and alphabetically when ages are the same, line by line.


```
$> ./CoffeeCroissant.rb
Stacy
Jill
Juan
Steve
Dom
Frank
$>
```



The hash's data change during evaluation to verify that the task has been completed correctly.

Chapter XI

Exercise 07: elm

	Exercise 07
Exercise 07: elm	
Turn-in directory : <code>ex07/</code>	
Files to turn in : <code>elm.rb</code>	
Allowed functions : <code>n/a</code>	

The `d01.tar.gz` tarball in the appendix of this contains a subfolder named `ex07/`. Inside this folder, you'll find a file named `periodic_table.txt`, which describes the periodic table of elements in a programmer-friendly format.

Create a program that uses this file to generate an **HTML** page representing the periodic table of elements in a correct format.

Here are the requirements:

- Each element should be displayed in a 'box' withing the **HTML** table.
- The name of an element should be displayed as level 4 heading.
- The attributes of an element should be presented as a list. This list should include at least the atomic number, symbol and atomic mass.
- You will need to refer to the layout of a Mendeleïev table on platforms like Google for proper alignment. Ensure there are empty boxes where necessary and include appropriate line breaks when needed.

Your program should generate the `periodic_table.html` result file. This **HTML** file must be readable by any browser and should comply with **W3C** standards

Feel free to design you program according to your preferences. It is recommended to break your code into specific functionalities that can be potentially reuse. You can customize your hashes with an "inline" **CSS** style to enhance the appearance of your table, such as styling the table's borders. You can even generate a separate `aperiodic_table.css`

file to style the table if you prefer.

Here is an output excerpt that will give you a better idea of what is expected:

```
[...]
<table>
  <tr>
    <td style="border: 1px solid black; padding:10px">
      <h4>Hydrogen</h4>
      <ul>
        <li>No 1</li>
        <li>H</li>
        <li>1.00794</li>
        <li>1 electron</li>
      </ul>
    </td>
  </tr>
</table>
[...]
```

Chapter XII

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.