



# Ruby on Rails Training - 0

## Starting

*Summary: Let's leave the web domain behind and focus on the [ruby](#) <3 and this language's syntactic and semantic basics.*

*Version: 1*

# Contents

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

# 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.


# Chapter III

## Today's specific instructions

- Every turned-in files will feature a fitting shebang AND the warning flag.
- No code in the global scope. Make functions!
- Each turned-in file must end with a function call.
- Imports are prohibited except for the ones specified in the "Authorized functions" section in each exercise cart.

# Chapter IV

## Exercise 00: Classy not classy

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


Create a script named `var.rb` in which you will define a `my_var` function. In this function, declare and set 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: Fixnum
  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**. 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 : <i>ex01/</i>	
Files to turn in : <b>croissant.rb</b>	
Allowed functions : <b>n/a</b>	

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

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


Design a Ruby script named `croissant.rb` that will open the `numbers.txt` file, read its numbers and display them on the standard output, one per line, without coma, in ascending order.



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 : <i>H2o.rb</i>	
Allowed functions : <i>n/a</i>	

Once again, you're free to define as many functions and name them as you see fit. This instruction won't be mentioned anymore, except if it has to be contradicted.

Create a script named *H2o.rb* in which you will copy the following **data** couples table, as is, in either 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 it and turns it into a hash with FixNum as a key and the String(s) as value, displaying on the console a message 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 : <b>Where.rb</b>	
Allowed functions : <b>n/a</b>	

Using the following hashes (you will copy them in a function, I will not specify again either):

```
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 (ex: Oregon) as an argument and displays its capital city in the standard output (ex: Salem). If the argument doesn't give any result, your script must display: **Unknown state**. If there is no or too many arguments, your script must not react and it must quit.

```
$> ./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 : <b>erehW.rb</b>	
Allowed functions : <b>n/a</b>	

You have the same hashes as in the previous exercise. Create a program that takes a capital city as the argument and displays the matching State. Your program must behave identically as 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 : <b>wheretor.rb</b>	
Allowed functions : <b>n/a</b>	


Always with the same hashes as the **ex03**'s, write a program similar to previous exercises except:

- The program must take a string containing as many words as you like, separated by a coma, as the argument.
- For each word in this string, the program must detect whether this word is a capital city or a State.
- The program must not take the case or the spaces.
- If there are none or too many parameters, the program does not display anything.
- When there are two consecutive comas in the string, the program doesn't display anything.
- The program must display results separated by a carriage return and use the following specific format:

```
$> ./wheretor.rb "Salem , ,Alabama, Toto , ,MontGOMery"
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 : <b>CoffeeCroissant.rb</b>	
Allowed functions : <b>n/a</b>	

Using the following table:

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

Write the code that only displays the names sorted out by ascending 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 check the work has been correctly completed.

# Chapter XI

## Exercise 07: elm

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

The `d01.tar.gz` tarball in this subject's appendix contains a subfolder named `ex07/` in which you'll find a file named `periodic_table.txt`, that describes the periodical table of the elements in a useful format for programmers.

Create a program that uses this file to write an **HTML** page representing the periodical table of elements in a correct form.

- Each element must appear in a 'box' of the **HTML** table.
- The name of an element must be a level 4 title hash.
- The attributes of an element must appear as a list. This list must at least feature an atomic number, a symbol and an atomic mass.
- You will have to remotely observe the layout of a Mendeleiev table as you'll find it on Google. There will have to be empty boxes where there should be as well as carriage returns where necessary.

Your program must create the `periodic_table.html` result file. This **HTML** file must obviously be readable by any browser and must be **W3C** compliant.

You're free to design you program as you see fit. Don't hesitate breaking your code into specific functionalities you may potentially reuse. You can customize your hashes with an "incline" **CSS** style to make your repo prettier (just for the table's borders, for instance). You can even generate a `periodic_table.css` table if you like.

Here is an output excerpt that will give you a slight idea of what's 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.