



D01 - Ruby on Rails Training

Syntactic and semantic basics

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

Contents

I	Preamble	2
II	Ocaml piscine, general rules	3
III	Today's specific instructions	5
IV	Exercise 00: Classy not classy	6
V	Exercise 01: Breakfast	7
VI	Exercise 02: With Hash browns	8
VII	Exercise 03: Where am I?	10
VIII	Exercise 04: Backward	11
IX	Exercise 05: Hal	12
X	Exercise 06: Wait a minute	13
XI	Exercise 07: elm	14

Chapter I

Preamble

Besides being great, Ruby is fun!

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

Chapter II

Ocaml piscine, general rules

- Every output goes to the standard output, and will be ended by a newline, unless specified otherwise.
- The imposed filenames must be followed to the letter, as well as class names, function names and method names, etc.
- Unless otherwise explicitly stated, the keywords `open`, `for` and `while` are forbidden. Their use will be flagged as cheating, no questions asked.
- Turn-in directories are `ex00/`, `ex01/`, ..., `exn/`.
- You must read the examples thoroughly. They can contain requirements that are not obvious in the exercise's description.
- Since you are allowed to use the OCaml syntaxes you learned about since the beginning of the piscine, you are not allowed to use any additional syntaxes, modules and libraries unless explicitly stated otherwise.
- The exercises must be done in order. The graduation will stop at the first failed exercise. Yes, the old school way.
- Read each exercise FULLY before starting it! Really, do it.
- The compiler to use is `ocamlopt`. When you are required to turn in a function, you must also include anything necessary to compile a full executable. That executable should display some tests that prove that you've done the exercise correctly.
- Remember that the special token `";;"` is only used to end an expression in the interpreter. Thus, it must never appear in any file you turn in. Regardless, the interpreter is a powerful ally, learn to use it at its best as soon as possible!
- The subject can be modified up to 4 hours before the final turn-in time.
- In case you're wondering, no coding style is enforced during the OCaml piscine. You can use any style you like, no restrictions. But remember that a code your peer-evaluator can't read is a code he or she can't grade. As usual, big functions are a weak style.
- You will NOT be graded by a program, unless explicitly stated in the subject. Therefore, you are given a certain amount of freedom in how you choose to do the

exercises. However, some piscine day might explicitly cancel this rule, and you will have to respect directions and outputs perfectly.

- Only the requested files must be turned in and thus present on the repository during the peer-evaluation.
- Even if the subject of an exercise is short, it's worth spending some time on it to be absolutely sure you understand what's expected of you, and that you did it in the best possible way.
- By Odin, by Thor! Use your brain!!!


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 : 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 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 : croissant.rb	
Allowed functions : n/a	

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 : Where.rb	
Allowed functions : n/a	

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 : 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 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 : wheretor.rb	
Allowed functions : n/a	


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 : 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 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 : <code>ex07/</code>	
Files to turn in : <code>elm.rb</code>	
Allowed functions : <code>n/a</code>	

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 your 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>
[...]
```