



D03 - Ruby on Rails Training

Gems

Summary: You can always rely on a Gem.

Contents

I	Preamble	2
II	Ocaml piscine, general rules	3
III	Specific instructions for today	5
IV	Exercise 00: I like	6
V	Exercise 01: ft_wikipedia	8
VI	Exercise 02: TDD	10
VII	Exercise 03: Rails	11

Chapter I

Preamble

The RS7000 is a major groove production workstation! It's sort of like Akai's MPC-series (but for bosses), combining sampling and sequencing, but with an added internal synth engine. The RS7000 is particularly suited for dance, techno, Hip Hop, R&B, and ambient genres.

The sampler section consists of a 4MB (expandable to 64MB) sampler (5kHz to 44.1kHz or 32kHz to 48kHz via digital option board). You can use it to sample external sounds, re-sample the RS7000's sounds itself, or load samples from a variety of common formats! Auto-beat slicing lets you easily sample any loops or sounds and sync them to your sequence tempo! All the professional sampling and editing features you'd expect are here, and more!

The tone generator offers 62-voice polyphonic AWM2 synthesis, with over 1,000 synth sounds and 63 drum kit sounds (all via ROM). Here you'll find the resonant filters (6 types), advanced LFO modulation, BPM-synchronized LFO waveforms, and more! Edits made to the internal sounds, as well as to any samples are all stored within your sequence patterns.

The Sequencer is the real meat of the RS7000, where you make music out of the sounds it's got and that you've put into it! It offers pattern-based recording with 16 tracks each, and a 200,000 note-per-song capacity. Linear sequencer sequencing, like you would do using a software sequencer like Cubase, is also supported by the RS7000. Pattern-based sequences can be converted to the linear format as well. Realtime, grid and step recording methods are also available. Linking patterns into songs can be done in real time and meticulously tweaked.

Total MIDI control, real-time hands on control, 18 assignable knobs and two pads, a Master effect section (with a multi-band compressor, slicer, isolater, other DJ-style master effects), and more make the RS7000 the most professional quality groove/loop/dance machine out there!

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

Specific instructions for today

- 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 be a Gem. (except for exercise 03!)
- Each Gem must use `minitest`, the MIT license and must not propose any conduct code.
- The Gems have an educational aim: don't bother with the upload rests! **Just comment the corresponding lines in the .gemspec.**
- Each Gem must include the tests required in the exercise and they must be executed by a `bundle exec rake` in the Gem's root folder.
- Imports are prohibited except for the ones specified in the "Authorized functions" section in each exercise cart.

Chapter IV

Exercise 00: I like

	Exercise 00
Exercise 00: I like	
Turn-in directory : <i>ex00/</i>	
Files to turn in : deephought	
Allowed functions : colorize	

Create your first Gem!

You're about to deliver a code portion that the whole world will be able to use... Providing the following code a portable way (de manière portable en français, je n'ai même pas trop compris la phrase française haha, je me demande s'il n'y a pas une coquille dans l'énoncé):

```
require 'colorize'

class Deepthought
  def initialize
    end
  def respond(question)
    if question == "The Ultimate Question of Life, the Universe and Everything"
      puts "42".green
      return "42"
    else
      puts "Mmmm i'm bored".red
      return "Mmmm i'm bored"
    end
  end
end
```


Exciting, right?

Your Gem must observe the following specifications:

- Gem name: `deepthought`
- Tests? Of course! Use `minitest`
- A conduct code? No
- License: MIT
- Version: `'0.0.1'`
- The `"grep -Hrn 'TODO' -color=always ."` command, executed at the root of your Gem, must not return ANYTHING
- You must write a test that checks that `Deepthought.new` does return the expected object
- You must write a test that checks the return value of both `'respond'` methods.

Chapter V

Exercise 01: ft_wikipedia

	Exercise 01
Exercise 01: ft_wikipedia	
Turn-in directory : <i>ex01/</i>	
Files to turn in : ft_wikipedia	
Allowed functions : nokogiri/open-uri	

There's an interesting fact about Wikipedia. It's called "the path to philosophy". If you click on the first link of a page that not in italic or between brackets, you will get end on the philosophy page in 94

From my own experience, it never took more than 35 clicks...

If you want to try it yourself, create a "ft_wikipedia" Gem you can use and that displays the following:

```
Ft_wikipedia.search("Kiss")
First search @ :https://en.wikipedia.org/wiki/Kiss
https://en.wikipedia.org/wiki/Love
https://en.wikipedia.org/wiki/Affection
https://en.wikipedia.org/wiki/Disposition
https://en.wikipedia.org/wiki/Habit_(psychology)
https://en.wikipedia.org/wiki/Behavior
https://en.wikipedia.org/wiki/American_and_British_English_spelling_differences
https://en.wikipedia.org/wiki/English_orthography
https://en.wikipedia.org/wiki/Orthography
https://en.wikipedia.org/wiki/Convention_(norm)
https://en.wikipedia.org/wiki/Norm_(philosophy)
https://en.wikipedia.org/wiki/Sentence_(linguistics)
https://en.wikipedia.org/wiki/Word
https://en.wikipedia.org/wiki/Linguistics
https://en.wikipedia.org/wiki/Science
https://en.wikipedia.org/wiki/Knowledge
https://en.wikipedia.org/wiki/Awareness
https://en.wikipedia.org/wiki/Conscious
https://en.wikipedia.org/wiki/Quality_(philosophy)
https://en.wikipedia.org/wiki/Philosophy
=> 19
```

The programs lists every visited url, and returns the number of links it needed to reach the "https://en.wikipedia.org/wiki/Philosophy" page.

Sometimes, searches like "matter" end in a loop!! You **must** manage these occurrences raising a "StandardError" exception that displays as follows:

```
Ft_wikipedia.search("matter")
First search @ :https://en.wikipedia.org/wiki/matter
https://en.wikipedia.org/wiki/Atom
https://en.wikipedia.org/wiki/Matter
https://en.wikipedia.org/wiki/Atom
Loop detected there is no way to philosophy here
=> nil
```

Sometimes, a search such as "Effects_of_blue_lights_technology" will end in a stalemate. You **must** manage these occurrences raising a "StandardError" exception that displays as follows:

```
Ft_wikipedia.search("Effects_of_blue_lights_technology")
First search @ :https://en.wikipedia.org/wiki/Effects_of_blue_lights_technology
Dead end page reached
=> nil
```


Keep in mind that by "first link", we mean a link featured in the article that is:

- an article on Wikipedia
- an article in the same language
- a real article (not a file or a support)

You also **must** write every test that will prove your program is running properly with accurate searches: "directory", "problem", "Einstein", "kiss", "matter"... These examples are absolutely not contractual.

Chapter VI

Exercise 02: TDD

	Exercise 02
Exercise 02: TDD	
Turn-in directory : <i>ex02/</i>	
Files to turn in : Taillste	
Allowed functions : n/a	

The TDD, or **Test-Driven Development**, is a practice that runs way beyond the field of web development.

We provide an empty Gem you will have to implement. They already come with tests. I won't talk too much about the features in this Gem since the goal of this exercise is to make you find them out by yourself running tests.


To validate the exercise, the following requirements **WILL HAVE** to be observed:

- The "gem build" command at the root of your Gem's folder is running properly, without any error (besides help warning, description lacking and homepage missing)
- The last displayed line of the "rake" command will be:

██

Chapter VII

Exercise 03: Rails

	Exercise 03
Exercise 03: Rails	
Turn-in directory : <i>ex03/</i>	
Files to turn in : HelloWorld	
Allowed functions : n/a	

Aaaah! The infamous Hello World! Here you are, facing the wall...

You must install **rails** on your machines, make a page containing a title, "Hello World!", and when you run the server included in rails, this page should be the first to show.

Google is filled with good advice, still, still, here is a good one: get documentation before you run your install...

Your install really has to run flawlessly, given that you will have to run it several times. You better start on the right foot!

To validate the exercise, the following requirements **WILL HAVE** to be observed:

- Install the rails Gem, including its necessary dependancies
- Make things so that the included server starts (the command is "rails server" or "rails s")
- Make sure the first page of your website (accessible via "http://localhost:3000/") is an HTML page displaying "Hello World!" in a tag named **<h1>**