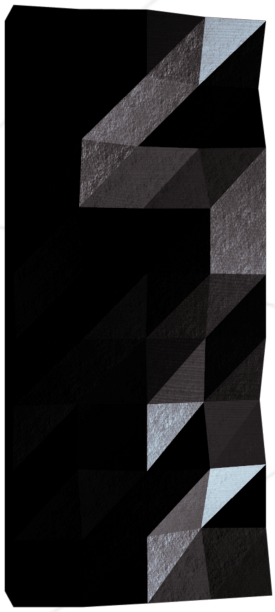

Ruby on Rails Training - 1

Gems

Summary: You can always rely on a Gem.

Version: 1

Contents

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

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

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 : $ex00/$	
Files to turn in : `deepthought`	
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 : $ex01/$	
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% of cases.

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 : $ex02/$	
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 : $ex03/$	
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 virtual machine, 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>`

Chapter VIII

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.`