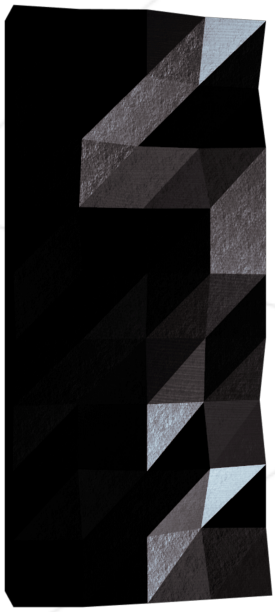

D03 - Formation Ruby on Rails

English version coming soon !

Résumé: Il y a toujours une Gem pour ça.

Table des matières

Chapitre I

Préambule

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 !

Chapitre II

Consignes

- Seule cette page servira de référence : ne vous fiez pas aux bruits de couloir.
- Le sujet peut changer jusqu'à une heure avant le rendu.
- Si aucune information contraire n'est explicitement présente, vous devez assumer les versions de langages suivantes :
  - `Ruby >= 2.3.0`
  - pour `d09 Rails > 5`
  - mais pour tous les autres jours `Rails 4.2.7`
  - `HTML 5`
  - `CSS 3`
- Nous vous interdisons FORMELLEMENT d'utiliser les mots clés `while, for, redo, break, retry, loop et until` dans les codes sources Ruby que vous rendrez. Toute utilisation de ces mots clés est considérée comme triche (et/ou impropre), vous donnant la note de -42.
- Les exercices sont très précisément ordonnés du plus simple au plus complexe. En aucun cas, nous ne porterons attention ni ne prendrons en compte un exercice complexe si un exercice plus simple n'est pas parfaitement réussi.
- Attention aux droits de vos fichiers et de vos répertoires.
- Vous devez suivre la procédure de rendu pour tous vos exercices : seul le travail présent sur votre dépot GIT sera évalué en soutenance.
- Vos exercices seront évalués par vos camarades de piscine.
- Vous ne devez laisser dans votre répertoire aucun autre fichier que ceux explicitement specifiés par les énoncés des exercices.
- Vous avez une question ? Demandez à votre voisin de droite. Sinon, essayez avec votre voisin de gauche.
- Votre manuel de référence s'appelle `Google / man / Internet / ...`.
- Pensez à discuter sur le forum Piscine de votre Intra, ou Slack, ou IRC...

- Lisez attentivement les exemples. Ils pourraient bien requérir des choses qui ne sont pas autrement précisées dans le sujet...
- Par pitié, par Thor et par Odin ! Réfléchissez nom d'une pipe !

Chapitre III

Règles spécifiques de la journée

- Tous les fichiers rendus seront dotés d'un shebang approprié ET du flag de warning.
- Aucun code dans le scope global. Faites des fonctions ou des classes !
- Chaque rendu doit etre une Gem (sauf pour l'exercice 03 !)
- Chaque Gem doit utiliser `minitest`, la license MIT et ne doit proposer aucun code de conduite.
- Ces Gems sont à vocation pédagogiques : ne vous occupez pas des repos d'upload ! **Commentez les lignes correspondantes dans le .gemspec**.
- Chaque Gem doit inclure les tests demandés dans l'exercice et ceux-ci doivent être executés par un `bundle exec rake` dans le dossier racine de la gem
- Aucun import autorisé, à l'exception de ceux explicitement mentionnés dans la section "Fonctions Autorisées" du cartouche de chaque exercice.

Chapitre IV

Exercice 00 : J'aime

	Exercice : 00
Exercice 00 : J'aime	
Dossier de rendu : $ex00/$	
Fichiers à rendre : `deepthought`	
Fonctions Autorisées : `colorize`	

Créez votre première gem !
Vous allez ainsi mettre au monde une portion de code réutilisable par la planète entière...
Procurant de maniere portable le code suivant :

```
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
```

Excitant, non ? !

Votre Gem doit répondre aux specificités suivantes :

- Nom de Gem : deepthought
- Des tests ? oui bien sur ! : utilisez `minitest`
- Un code de conduite ? Non
- Licence : MIT
- Version : '0.0.1'
- La commande `"grep -Hrn 'TODO' -color=always ."`, exécutée à la racine de votre Gem, ne doit RIEN retourner
- Vous devez écrire le test qui vérifie que Deepthought.new retourne bien l'objet attendu
- Vous devez écrire le test qui vérifie la valeur de retour des deux cas de la méthode 'respond'

Chapitre V

Exercice 01 : ft_wikipedia

	Exercice : 01
Exercice 01 : ft_wikipedia	
Dossier de rendu : $ex01/$	
Fichiers à rendre : `ft_wikipedia`	
Fonctions Autorisées : `nokogiri/open-uri`	

Un fait intéressant de Wikipedia, est la "route vers la philosophie".
Si on clique sur le premier lien en minuscule de la première description de n'importe quelle page, dans 94% des cas, on atteint la page philosophie.

De mon experience, cela n'a jamais depassé 35 liens...

Pour en être sur, vous devez creer une gem "ft_wikipedia" qui s' utilise et s'affiche comme ceci :

```
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
```

Le programme liste toutes les urls visitées, et retourne le nombre de liens qu'il a du

traverser avant la page "https ://en.wikipedia.org/wiki/Philosophy".

Il arrive que certaines recherches comme "matter" tournent en rond!! vous **devez** gérer ce cas avec une levée d'exception "StandardError", et afficher comme suit :

```
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
```

Il arrive que certaines recherches comme "Effects_of_blue_lights_technology" soient des impasses!! vous **devez** gérer cela avec une levée d'exception "StandardError", et afficher comme suit :

```
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
```

Pensez bien aussi que par "premier lien", on entends un lien disponible sur l'article qui soit :

- un article disponible sur Wikipedia
- un article de la même langue
- un vrai article (et non un fichier ou une aide)

Vous **devez** également écrire tous les tests qui prouvent le bon fonctionement de votre programme avec des recherches appropriées : "directory", "problem", "einstein", "kiss", "matter"...
Ce sont des exemples vraiment non contractuels pour le coup.

Chapitre VI

Exercice 02 : TDD

	Exercice : 02
Exercice 02 : TDD	
Dossier de rendu : $ex02/$	
Fichiers à rendre : `Taillste`	
Fonctions Autorisées : `n/a`	

Le TDD, ou `Test-Driven Development`, est une pratique très courrue par delà le monde du developpement web.

Nous vous fournissons une Gem vide qu'il vous faut implémenter. Des tests y ont été deja écrits. Je ne m'étendrai pas sur les features de cette Gem, étant donné que le but de l'exercice est que vous devez les déduire avec ces tests.

Pour valider l'exercice, il faut `IMPERATIVEMENT` que :

- La commande "gem build" à la racine du dossier de votre Gem s'exécute sans erreur (outre les warnings d'aide, de manque de description et de homepage manquante)
- la dernière ligne de l'affichage de la commande "rake" soit :

```
"16 runs, 16 assertions, 0 failures, 0 errors, 0 skips"
```

Chapitre VII

Exercice 03 : Rails

	Exercice : 03
Exercice 03 : Rails	
Dossier de rendu : $ex03/$	
Fichiers à rendre : `HelloWorld`	
Fonctions Autorisées : `n/a`	

Aaaah ! Hello World ! le célèbre. Vous voici donc maintenant au pied du mur ...

Vous devez installer `rails` sur vos machines, faire une page contenant un titre "Hello World !", et lorsque vous lancez le serveur inclus dans rails, cette page doit être la première affichée.

Google est rempli de bons conseils, cependant j'en ai un autre pour vous : documentez vous avant de lancer votre installation...

Il est primordial que votre installation se déroule correctement, étant donné que vous aurez à le refaire plusieurs fois. Autant prendre un bon départ !

Pour valider l'exercice, il faudra :

- Installer la Gem rails, dépendances nécessaires incluses
- Faire en sorte que le serveur inclus dans rails démarre (la commande est : "rails server" ou "rails s")
- Faire en sorte que la première page de votre site (accessible via "http ://localhost :3000/") soit une page HTML affichant "Hello World !" dans une balise de titre `<h1>`