



Formation Ruby on Rails - 0

Starting

Résumé: Quittons un peu le domaine du web pour se concentrer sur le [ruby](#) <3 , et les bases syntaxiques et sémantiques de ce langage.

Version: 1

Table des matières

I	Préambule	2
II	Règles communes	3
III	Règles spécifiques de la journée	4
IV	Exercice 00 : Classe Pas Classe	5
V	Exercice 01 : Petit Déjeuner	6
VI	Exercice 02 : Hashment bien	7
VII	Exercice 03 : Where am I ?	9
VIII	Exercice 04 : Backward	10
IX	Exercice 05 : Hal	11
X	Exercice 06 : Wait a minute	12
XI	Exercice 07 : elm	13
XII	Rendu et peer-évaluation	15

Chapitre I

Préambule

En plus d'être génial, Ruby est drôle !

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

Chapitre II

Règles communes

- Votre projet doit être réalisé dans une machine virtuelle.
- Votre machine virtuelle doit avoir tout les logiciels necessaire pour réaliser votre projet. Ces logiciels doivent être configurés et installés.
- Vous êtes libre sur le choix du systmème d'exploitation à utiliser pour votre machine virtuelle.
- Vous devez pouvoir utiliser votre machine virtuelle depuis un ordinateur en cluster.
- Vous devez utiliser un dossier partagé entre votre machine virtuelle et votre machine hôte.
- Lors de vos évaluations vous allez utiliser ce dossier partager avec votre dépôt de rendu.
- Vos fonctions de doivent pas s'arrêter de manière inattendue (segmentation fault, bus error, double free, etc) mis à part dans le cas d'un comportement indéfini. Si cela arrive, votre projet sera considéré non fonctionnel et vous aurez 0 au projet.
- Nous vous recommandons de créer des programmes de test pour votre projet, bien que ce travail **ne sera pas rendu ni noté**. Cela vous donnera une chance de tester facilement votre travail ainsi que celui de vos pairs.
- Vous devez rendre votre travail sur le git qui vous est assigné. Seul le travail déposé sur git sera évalué. Si Deepthought doit corriger votre travail, cela sera fait à la fin des peer-evaluations. Si une erreur se produit pendant l'évaluation Deepthought, celle-ci s'arrête.


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 !
- Chaque fichier rendu doit être terminé par un appel de fonction.
- Aucun import autorisé, à l'exception de ceux explicitement mentionnés dans la section "Fonctions Autorisées" du cartouche de chaque exercice.

Chapitre IV

Exercice 00 : Classe Pas Classe

	Exercice : 00
Exercice 00 : Classe Pas Classe	
Dossier de rendu : <i>ex00/</i>	
Fichiers à rendre : var.rb	
Fonctions Autorisées : n/a	


Créez un script nommé `var.rb` dans lequel vous devez définir une fonction `my_var`. Dans celle-ci, déclarez et initialisez 4 variables de types différents et imprimez les sur la sortie standard. Vous devez reproduire exactement la sortie suivante :

```
$> ./var.rb
mes variables :
  a contient : 10 et est de type: Fixnum
  b contient : 10 et est de type: String
  c contient : nil et est de type: NilClass
  d contient : 10.0 et est de type: Float
$>
```

Il est bien entendu **interdit** d'écrire explicitement les types de vos variables dans les prints votre code. N'oubliez pas d'appeler votre fonction à la fin de votre script comme expliqué dans les consignes.

Chapitre V

Exercice 01 : Petit Déjeuner

	Exercice : 01
Exercice 01 : Petit Déjeuner	
Dossier de rendu : <code>ex01/</code>	
Fichiers à rendre : <code>croissant.rb</code>	
Fonctions Autorisées : <code>n/a</code>	

Pour cet exercice, vous êtes libre de définir autant de fonctions que vous le souhaitez et de les nommer comme bon vous semble.

La tarball `d01.tar.gz` en annexe de ce sujet contient un sous-dossier `ex01/` dans lequel se trouve un fichier `numbers.txt` contenant des nombres aléatoires allant de 1 à 100 séparés par une virgule.


Concevez un script Ruby nommé `croissant.rb` dont le rôle est d'ouvrir le fichier `numbers.txt`, de lire les nombres qu'il contient, et de les afficher sur la sortie standard, un par ligne et sans virgule, dans l'ordre... croissant.



Commande pour extraire une tarball: `tar xzf d01.tar.gz`

Chapitre VI

Exercice 02 : Hashment bien

	Exercice : 02
Exercice 02 : Hashment bien	
Dossier de rendu : <i>ex02/</i>	
Fichiers à rendre : H2o.rb	
Fonctions Autorisées : n/a	

Vous êtes à nouveau libres de définir autant de fonctions que vous le souhaitez et de les nommer comme bon vous semble. Cette consigne ne sera plus mentionnée, sauf en cas de contradiction explicite.

Créez un script nommé **H2o.rb** dans lequel vous devez copier le tableau de couples **data** suivant tel quel dans l'une ou l'autre de vos fonctions :


```
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]]
```


Ecrivez le code qui, lorsqu'il est exécuté, le déclare et le transforme en hash avec pour clé le FixNum et comme valeur la/les String(s), affichant sur la console un message comme suit :

```
$> ./H2o.rb  
24 : Caleb  
84 : Calixte  
65 : Calliste  
12 : Calvin  
[...]  
$>
```

Chapitre VII

Exercice 03 : Where am I ?

	Exercice : 03
Exercice 03 : Where am I ?	
Dossier de rendu : <i>ex03/</i>	
Fichiers à rendre : Where.rb	
Fonctions Autorisées : n/a	

En utilisant les hashes suivants (vous les recopierez dans une fonction, ça non plus je ne l'écrirai plus) :

```
states = {
  "Oregon"    => "OR",
  "Alabama"   => "AL",
  "New Jersey" => "NJ",
  "Colorado"  => "CO"
}


capitals_cities = {
  "OR" => "Salem",
  "AL" => "Montgomery",
  "NJ" => "Trenton",
  "CO" => "Denver"
}
```

Ecrivez le programme qui prend en argument un État (ex : Oregon) et qui affiche sur la sortie standard sa capitale (ex : Salem). Si l'argument ne donne aucun résultat, votre script doit afficher : **Unknown state**. S'il n'y a pas ou bien s'il y a trop d'arguments, votre script ne doit rien faire et quitter.

```
$> ./Where.rb Oregon
Salem
$> ./Where.rb toto
Unknown state
$> ./Where.rb
$> ./Where.rb Oregon Alabama
$> ./Where.rb Oregon Alabama Ile-De-France
$>
```

Chapitre VIII

Exercice 04 : Backward


	Exercice : 04
Exercice 04 : Backward	
Dossier de rendu : <i>ex04/</i>	
Fichiers à rendre : erehW.rb	
Fonctions Autorisées : n/a	

Vous disposez à nouveau des deux mêmes hashes qu'à l'exercice précédent. Créez un programme qui prend une capitale en argument et qui affiche cette fois-ci l'État correspondant. Le reste des comportements de votre programme doivent être identiques à ceux de l'exercice précédent.

```
$> ./erehW.rb Salem
Oregon
$> ./erehW.rb toto
Unknown capital city
$> ./erehW.rb
$>
```

Chapitre IX

Exercice 05 : Hal

	Exercice : 05
Exercice 05 : Hal	
Dossier de rendu : <i>ex05/</i>	
Fichiers à rendre : wheretor.rb	
Fonctions Autorisées : n/a	


Toujours en réutilisant les hashes de l'ex03, écrivez un programme similaire aux précédents, à la différence que :

- Le programme doit prendre en argument une string contenant autant de mots à chercher qu'on veut, séparés par une virgule.
- Pour chaque mot de cette string, le programme doit détecter si ce mot est une capitale ou un État.
- Le programme ne doit pas être sensible à la casse, ni aux espaces blancs.
- S'il n'y a pas de paramètre ou bien trop de paramètres, le programme n'affiche rien.
- Lorsqu'il y a deux virgules d'affilée dans la string, le programme n'affiche rien.
- Le programme doit afficher les résultats séparés par un retour à la ligne. et utiliser le formatage précis suivant :

```
$> ./wheretor.rb "Salem , ,Alabama, Toto , ,MontG0mery"
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)
$>
```

Chapitre X

Exercice 06 : Wait a minute

	Exercice : 06
Exercice 06 : Wait a minute	
Dossier de rendu : <i>ex06/</i>	
Fichiers à rendre : CoffeeCroissant.rb	
Fonctions Autorisées : n/a	

En utilisant le tableau suivant :

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

Ecrire le code qui affiche seulement les noms triés par ordre d'âge croissant puis alphabétique lorsque les âges sont identiques, ligne par ligne.


```
$> ./CoffeeCroissant.rb
Stacy
Jill
Juan
Steve
Dom
Frank
$>
```



Les données du hash seront changées pendant la soutenance pour vérifier que le travail a été fait correctement.

Chapitre XI

Exercice 07 : elm

	Exercice : 07
Exercice 07 : elm	
Dossier de rendu : <code>ex07/</code>	
Fichiers à rendre : <code>elm.rb</code>	
Fonctions Autorisées : <code>n/a</code>	

La tarball `d01.tar.gz` en annexe de ce sujet contient un sous-dossier `ex07/` dans lequel se trouve un fichier `periodic_table.txt` qui décrit le tableau périodique des éléments dans un format pratique pour les informaticiens.

Créez un programme qui utilise ce fichier pour écrire une page HTML représentant le tableau périodique des éléments correctement formaté.

- Chaque élément doit être dans une 'case' d'un tableau HTML.
- Le nom d'un élément doit être dans une balise titre de niveau 4.
- Les attributs d'un élément doivent être sous forme de liste. Doivent être listés au minimum le numéro atomique, le symbole, et la masse atomique.
- Vous devez respecter à minima le layout d'une table de Mendeleiev telle qu'on peut la trouver sur Google, à savoir qu'il doit y avoir des cases vides là où il doit y en avoir, ainsi que des retours à la ligne là où il en faut.

Votre programme doit créer le fichier résultat `periodic_table.html`. Ce fichier HTML doit bien entendu être immédiatement lisible par n'importe quel navigateur et doit être valide W3C.

Vous êtes libre de concevoir votre programme comme vous l'entendez. N'hésitez pas à fragmenter votre code en fonctionnalités distinctes et éventuellement réutilisables. Vous pouvez customiser les balises avec un style CSS "inline" pour rendre votre rendu plus joli (ne serait-ce que le contour des cases du tableau), voire même générer un fichier `periodic_table.css` si vous préférez.

Voici un extrait d'un exemple de sortie pour vous donner une idée :

```
[...]
<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>
[...]
```

Chapitre XII

Rendu et peer-évaluation

Rendez votre travail dans votre dépôt Git comme d'habitude. Seul le travail présent dans votre dépôt sera évalué en soutenance. Vérifiez bien les noms de vos dossiers et de vos fichiers afin que ces derniers soient conformes aux demandes du sujet.



L'évaluation se déroulera sur l'ordinateur du groupe évalué.