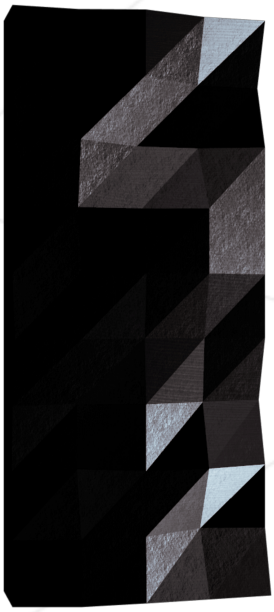

Initiation à la programmation

Ruby - Jour 1

Staff 42 pedago@42.fr

Résumé:
Ce document est le sujet du jour 1 du programme d'initiation à Ruby.

Table des matières

Chapitre I

Consignes

- Seule cette page servira de référence : ne vous fiez pas aux bruits de couloir.
- Attention aux droits de vos fichiers et de vos répertoires.
- Vous devez suivre la procédure de rendu pour tous vos exercices.
- Vous serez corrigés par les autres participants du programme d'initiation. C'est le "peer-correcting" de la pédagogie 42 !
- 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 / ...`.
- Lisez attentivement les exemples. Les exercices pourraient bien requérir des choses qui y sont précisées, et non dans le sujet...

Chapitre II

Procédure de rendu : Git

- Votre rendu est collecté via un serveur distant. Cela signifie que vous devrez envoyer sur ce serveur votre repertoire de rendu avec vos exercices suivant une arborescence précise.
- Le logiciel utilisé pour ce faire s'appelle `git`.
- Git permet de faire beaucoup de choses, vous trouverez ci-dessous les principales commandes :
- `Git clone` vous permet de `cloner` un répertoire présent sur un serveur directement dans votre répertoire courant. Cela signifie que tous les fichiers présents que vous y aviez mis depuis un autre poste se retrouveront sur votre ordinateur et pourront être modifiés et réenvoyés avec les commandes git suivantes. Cette commande est aussi importante quand vous souhaitez vérifier ce que vous avez "pushé" sur le serveur, n'hésitez pas à en abuser.

```
?> git clone vogsphere@vogsphere.42.fr:piscine/truc/machin/creme
Cloning into 'piscine'[...]
?>
```

- `Git add` vous permet d'ajouter à une liste de fichiers surveillés un ou plusieurs fichiers/repertoires. Ils seront ajoutés en l'état ce qui veut dire que si vous les modifiez par la suite, il faudra les ajouter à nouveau pour mettre à jour la liste.

```
?> git add file directory
?>
```

- `Git commit -m` vous permet de "fixer" votre liste de fichiers surveillés afin de préparer leur envoi sur le serveur. Le message est obligatoire et permet d'avoir un historique des créations et modifications que vous avez effectué au fur et à mesure de votre travail.

```
?> git commit -m "Ajout des exercices X Y Z"
[master (root-commit) 4e8b2aa] Ajout des exercices X Y Z
3 file changed, 500 insertions(+)
create mode 100755 X Y Z
?>
```

- `Git push origin master`, la commande ultime, celle qui vous permet d'envoyer sur le serveur vos exercices qui y seront collectés par la Moulinette. Vérifiez bien son retour, le moindre message d'erreur signifie que vous n'avez rien pu envoyer et que par conséquent votre répertoire sur le serveur est vide (ce qui serait dommage pour votre note). Si vous avez le moindre doute, contactez un membre du staff 42.

```
?> git push origin master
[...]
X files written.
?>
```

L'adresse git de votre dépôt se trouve sur l'intranet et dans le fichier config du répertoire .git de votre dossier de rendu, c'est avec elle que vous pourrez "git clone".

Chapitre III

Préambule

Voici ce que `Wikipédia` a à dire à propos de la loutre :

`La Loutre d'Europe ou Loutre européenne (Lutra lutra), souvent qualifiée de loutre commune dans les pays d'Europe où elle est présente, est un mammifère carnivore semi-aquatique et principalement nocturne, de la famille des Mustélidés (sous-famille Lutrinés). Elle est l'une des trois espèces de loutres se rattachant au genre Lutra. En France, on ne trouve que cette seule espèce de loutre.`

`Sa hauteur est d'environ 30 cm au garrot. Son pelage, brun foncé, est composé de deux couches : le poil de bourre, court, très fin, dense et laineux ; et le poil de jarre, long, lisse, brillant et imperméable.`

`Excellente nageuse, elle dispose de pattes palmées, d'un corps allongé (60 à 80 cm en moyenne, auquel il faut ajouter une queue épaisse à la base et s'effilant vers l'extrémité de 30 à 40 cm de longueur), pour un poids allant de 5 à 15 kg.`

`Elle vit au bord des cours d'eau (ruisseaux, rivières et même fleuves), jusqu'à une altitude de 1300 m, dans les marais et parfois sur les côtes marines. Elle est habituellement solitaire, occupant un territoire de 5 à 15 km de rives le long d'un cours d'eau (parfois davantage) ou de 20 à 30 km2 en zone de marais. Elle emprunte régulièrement les mêmes passages sur la berge pour se mettre à l'eau : les "coulées". Lorsqu'elle sort de l'eau, elle se roule dans l'herbe pour essuyer sa fourrure, sur des zones reconnaissables à l'herbe couchée et appelées "places de ressui".`

`Elle fait sa tanière (qu'on appelle une "catiche", de l'ancien français "se catir" = se blottir, se cacher) entre les racines des arbres des berges des cours d'eau ou dans d'autres cavités (cavité rocheuse, tronc creux, terrier d'une autre espèce). La catiche contient souvent une entrée plus ou moins dissimulée au-dessous du niveau d'eau et un conduit d'aération.`

C'est mignon, une loutre.

Chapitre IV

Exercice 00 : 42

	Exercice : 00
Les vraies savent afficher 42	
Dossier de rendu : $ex00/$	
Fichiers à rendre : `42.rb`	
Fonctions Autorisées : `Toutes`	
Remarques : `n/a`	

- Créez un script `42.rb` qui, lorsqu'on l'exécute, affiche "42" suivi d'un retour à la ligne.

```
?> ./42.rb | cat -e
42$
?>
```

Ca vous parait simple ? C'est normal.

Le ruby est un langage très proche de l'anglais, cela rend les recherches plus faciles.

Chapitre V

Exercice 02 : XXX !

	Exercice : 02
XXX !	
Dossier de rendu : *ex*02/	
Fichiers à rendre : `xxx.rb`	
Fonctions Autorisées : `Toutes`	
Remarques : `n/a`	

- Créez un script `xxx.rb` qui affiche 1000 fois la lettre X et un retour à la ligne.
- Vous utiliserez une boucle while pour réaliser cet exercice.

```
$> ./xxx.rb | cat -e
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

[TL ;DP]

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX$
$>
```

Chapitre VI

Exercice 03 : what's your name

	Exercice : 03
Récupérer un input dans des variables	
Dossier de rendu : $ex03/$	
Fichiers à rendre : `whatsyourname.rb`	
Fonctions Autorisées : `Toutes`	
Remarques : `n/a`	

- Créez le script `whatsyourname.rb` qui demande d'abord à l'utilisateur d'entrer son nom, puis son prénom, et enfin affiche les deux.

```
?> ./whatsyourname.rb
Hey, what's your first name ? : Laurie
And your last name ? : Mezard
Well, pleased to meet you Laurie Mezard.
?>
```

Chapitre VII

Exercice 04 : i_got_that

	Exercice : 04
Have you got it ?	
Dossier de rendu : $ex04/$	
Fichiers à rendre : `i_got_that.rb`	
Fonctions Autorisées : `Toutes`	
Remarques : `n/a`	

- Créez un script `i_got_that.rb`. Ce script doit contenir une boucle while qui accepte un input de l'utilisateur, écrit une phrase en retour, et s'arrête uniquement lorsque l'utilisateur a entré "STOP". Chaque tour de boucle doit accepter un input de l'utilisateur.

```
?> ./i_got_that.rb
What you gotta say ? : Hello
I got that ! Anything else ? : I like ponies
I got that ! Anything else ? : stop...
I got that ! Anything else ? : STOP
?>
```

Chapitre VIII

Exercice 05 : do_op

	Exercice : 05
Mini-calculatrice	
Dossier de rendu : $ex05/$	
Fichiers à rendre : `do_op.rb`	
Fonctions Autorisées : `Toutes`	
Remarques : `n/a`	

- Vous allez réaliser une mini-calculette sommaire.

- Créez un script `do_op.rb`. Ce script va prompter l'utilisateur pour récupérer des valeurs et une ope2ratio arithmétoque parmi les suivantes : '+', '-', '*', '/' et '%'. Le programme va réaliser l'opération et annoncer le résultat. Il ne doit pas effectuer de gestion d'erreur. Une fois cela fait, il promptera l'utilisateur pour de nouvelles valeurs, et ce en boucle jusqu'à son arrêt par un CTRL-C.

```
?> ./do_op.rb
Donne-moi un premier nombre : 10
Donne-moi un operateur : +
Donne-moi un second nombre : 8
Le resultat est 18.
Donne-moi un premier nombre : ^C
?>
```

Chapitre IX

Exercice 06 : aff_first_param

	Exercice : 06
Afficher un paramètre	
Dossier de rendu : $ex06/$	
Fichiers à rendre : `aff_first_param.rb`	
Fonctions Autorisées : `Toutes`	
Remarques : `n/a`	

- Créez un script `aff_first_param.rb` qui, lorsqu'on l'exécute, affiche la première chaîne de caractères passée en paramètre suivie d'un retour à la ligne. S'il n'y a aucun paramètre, affichez `none` suivi d'un retour à la ligne.

```
?> ./aff_first_param.rb | cat -e
none$
?> ./aff_first_param.rb "Code Ninja" "Numerique" "42" | cat -e
Code Ninja$
?>
```

Chapitre X

Exercice 07 : UPCASE_IT

	Exercice : 07
Voir en grand	
Dossier de rendu : $ex07/$	
Fichiers à rendre : `upcase_it.rb`	
Fonctions Autorisées : `Toutes`	
Remarques : `n/a`	

- Créez un script `upcase_it.rb` qui prend une chaîne de caractères en paramètre. Lorsqu'on l'exécute, le script affiche la chaîne de caractères en majuscules suivie d'un retour à la ligne. Si le nombre de paramètres est différent de 1, affichez `none` suivi d'un retour à la ligne.

```
?> ./upcase_it.rb | cat -e
none$
?> ./upcase_it.rb "initiation" | cat -e
INITIATION$
?> ./upcase_it.rb 'CeT eXeRcIcE eSt AsSeZ fAcIlE !' | cat -e
CET EXERCICE EST ASSEZ FACILE !$
?>
```

Chapitre XI

Exercice 08 : downcase_it

	Exercice : 08
Voir plus petit	
Dossier de rendu : $ex08/$	
Fichiers à rendre : `downcase_it.rb`	
Fonctions Autorisées : `Toutes`	
Remarques : `n/a`	

- Créez un script `downcase_it.rb` qui prend une chaîne de caractères en paramètre. Lorsqu'on l'exécute, le script affiche la chaîne de caractères en minuscules suivie d'un retour à la ligne. Si le nombre de paramètres est différent de 1, affichez `none` suivi d'un retour à la ligne.

```
?> ./downcase_it.rb | cat -e
none$
?> ./downcase_it.rb "LUCIOLE" | cat -e
luciole$
?> ./downcase_it.rb 'CeT eXeRcIcE eSt AsSeZ fAcIlE !' | cat -e
cet exercice est assez facile !$
?>
```

`Cet exercice ne devrait pas vous prendre plus de 10 secondes.`

Chapitre XII

Exercice 09 : aff_rev_params

	Exercice : 09
Afficher un tableau à l'envers	
Dossier de rendu : $ex09/$	
Fichiers à rendre : `aff_rev_params.rb`	
Fonctions Autorisées : `Toutes`	
Remarques : `n/a`	

- Créez un script `aff_rev_params.rb` qui, lorsqu'on l'exécute, affiche toutes les chaînes de caractères passées en paramètre, suivies d'un retour à la ligne et dans l'ordre inverse. S'il y a moins de deux paramètres, affichez `none` suivi d'un retour à la ligne.

```
?> ./aff_rev_params.rb | cat -e
none$
?> ./aff_rev_params.rb "coucou" | cat -e
none$
?> ./aff_rev_params.rb "debutants" "les" "coucou" | cat -e
coucou$
les$
debutants$
?>
```

Chapitre XIII

Exercice 10 : scan_it

	Exercice : 10
Scanner un texte	
Dossier de rendu : $ex10/$	
Fichiers à rendre : `scan_it.rb`	
Fonctions Autorisées : `Toutes`	
Remarques : `n/a`	

- Créez un script `scan_it.rb` qui prend deux paramètres. Le premier est un mot clé a chercher dans une chaîne. Le deuxième est la chaîne à parcourir. Lorsqu'on l'exécute, le programme affiche le nombre d'occurences du mot clé dans la chaîne. Si le nombre de paramètres est différent de 2 ou que la première chaîne n'apparaît pas dans la deuxième, affichez `none` suivi d'un retour à la ligne.

```
?> ./scan_it.rb | cat -e
none$
?> ./scan_it.rb "les" | cat -e
none$
?> ./scan_it.rb "les" "les exercices du J01 ne sont pas les plus difficiles" | cat -e
3$
?>
```

Essayez "string.scan(string)".