Fiche Ruby

December 11, 2018

1 Généralités

1.1 Les variables

Il n'y a pas de typage des variables. Pour déclarer une variable il suffit d'écrire le nom de la variable suivi de sa valeur, par exemple:

```
name="Yoda"
nombre=5
```

- Variables **locales**: leurs noms commencent par une lettre minuscule ou un underscore "_"
- Variables **globales**: Ce sont des variables qui sont accessibles n'importe où dans le programme, leurs noms commencent par un "$"

  ```
  $globale=5
  ```

  Il existe quelques variables globales dont le nom est réservé:

  - $_ désigne la dernière chaîne lue par gets ;
  - $@ désigne le lieu de l'erreur (le cas échéant) ;
  - $. désigne la dernière ligne lue par l'interpréteur ;
  - $* désigne les arguments de la ligne de commande.
- Constantes: Leurs identifiants commencent par une lettre majuscule ou peuvent être écrits entièrement en majuscule.

  ```
  TAILLE = 1000
  ```

1.2 Les conditions

```
if nombre < 18
    puts "la personne est mineure"
elsif score > 100
    puts "l esperance de vie est compromise"
end
```

1.3 Les boucles

- Boucle **while**

```
while(condition)
    #CODE
end
```

- Boucle infinie **loop**

```
loop {#CODE}
```

  On utilise l'instruction break pour sortir de ce type de boucle.

- Boucle **until**

```
until(condition)
    #CODE
end
```

- Boucle **for**

```
for n in (0..5)
    puts n
end
```

1.4 Les saisies

Pour saisir le contenu d'une variable il suffit d'utiliser la méthode gets, e.g: name=gets

Exemple:

```
puts "votre nom ?"
    nom = gets
puts "votre nom est #{nom}" # "nom" sera remplacé par la valeur saisie
```

1.5 Affichage d'un élement

- puts "Hello World" : Affiche Hello World suivi d'un retour à la ligne
- print "Hello World" : Affiche Hello World suivi d'un espace

Pour afficher le contenu d'une variable dans un puts ou un print : , e.g : puts "Hello nom"

2 Fonction

Exemple de méthode permettant l'affichage d'un texte:

```
def h
puts "Hello World !"
end
```

3 Classes et Objets

Le mot `class` définit une classe. Chaque classe doit contenir une méthode `initialize`. Il s'agit du constructeur. Pour définir une variable de classe il suffit d'écrire : @variableName. Le @ permet de différentier une variable d'une classe d'une variable d'un objet.

On appelle attribut, la propriété d'un objet qui est visible de l'extérieur. Lorsqu'un attribut permet de lire la valeur d'une des propriété d'une classe il s'agit d'un attribut en lecture et se déclare `attr_reader :rue`

Lorsqu'il s'agit d'un attribut en écriture il est possible d'écrire `attr_writer :rue`.

La plupart du temps il sera nécessaire d'avoir des attributs qui fonctionnent en lecture et en écriture. Ainsi, il sera possible d'utiliser Exemple de structure d'une Classe:

```
class Adresse
    attr_accessor :rue, :code_postal, :ville, pays
    def initialize
        @rue = @code_postal = @ville = @pays = ""
    end
    def rue=(une_rue)
        @rue= une_rue
    end
end
```

Dans la classe Main:

```
adresse = Adresse.new
adresse.rue="Rue du Lion, 42"
```

4 Héritage

Il est possible d'établir des relations d'héritage entre classes.

- Définition de l'héritage

```
class Personne
# CODE
end

class Student < Personne # Student hérite de Personne
# CODE
end
```

- Redéfinition des méthodes
  On peut redéfinir la méthode d'une super-classe pour modifier son comportement.

```
class Personne
    def presentation
        puts "Je suis une personne"
    end
end

class Student < Personne
    def presentation
    # Redéfinition de la méthode presentation héritée de la classe Personne
        print "Je suis un(e) étudiant(e)"
    end
end

personne_1 = Personne.new
personne_2 = Student.new

personne_1.presentation # Méthode de la super-classe
personne_2.presentation # Méthode redéfinie
```

- Appel des méthodes de la super-classe en utilisant `super`

```
class Personne
    def presentation
        puts "une personne"
    end
end

class Student < Personne # Student hérite de Personne
    def presentation
        print "Je suis un(e) étudiant(e) donc"
```

```
            super
        end
    end
```

5 Exercice : Chiffres Romains

Les Romains écrivaient les nombres en utilisant les lettres: I, V, X, L, C, D et M.

L'objectif de cet exercice est d'écrire une fonction permettant de convertir des nombres arabes en nombres romains, comme par exemple:

```
 1 -> I
 10 -> X
8 -> VIII
```

Vous trouverez une description complète de comment cela fonctionne sur le lien suivant:
http://www.novaroma.org/via_romana/numbers.html (site en anglais)
Ce lien contient aussi une implémentation de cet exercice en langage JS.

Astuces

- Pouvez vous rendre le code lisible et compréhensible par tous ?
- Est ce que c'est plus facile de découper en petites fonctions ou de regrouper tout dans une seule fonction ?
- Quels sont les meilleures structures de données permettant de stocker tout les nombres romains ? (`I, V, D, M` etc)

Source: http://codingdojo.org/