RUBY vers. 2

Updated on December 15 2006
Thanks to Giulio Ardoino for his contribute.

The strings are written between ‘ ’ or between “ ” (but in this case they will become interpreted) for ex.:

a = 2 (numbers must be written without apexes singles or doubles)
puts 'a' ---> it will print a because between parenthesis the variable a is interpreted as string
puts "a" ---> it will print a because between parenthesis the variable a is interpreted as string
puts a ---> it will print 2 because the variable a is a number

a = ‘casa’ (the strings must be written with apexes otherwise we will have an error)
puts 'a' ---> it will print a
puts "a" ---> it will print a
puts a ---> it will print casa

A date is printed on video with instruction puts (the s means string)
puts 4 + 4 --> it will print -- 8
puts 5**2 --> it will print -- 25 (5 at square)
puts ' ' --> it will be empty
puts 'You'are back' --> it will give error
puts 'You\'re back' --> it will print -- You're back
+ joins the strings e/o numbers

The variables

The variables are not objects
Declare a Variable:
variabileuno = 'casa'
variabiledue = 2 (numbers must be written without apexes, less then we don’t want that 2 is a string)
puts variabileuno *2 --> it will print -- casacasa
variabiletre = ' casa'
puts variabiletre * 2 --> it will print -- casa casa
var1 = 20
var2 = var1
puts var1 --> it will print -- 20
puts var 2 --> it will print -- 20
var1 = 'seven'
puts var1 --> it will print -- seven

puts var2 --> it will print -- 20

Methods

The methods convert the dates of variables
variable.converter
.to_s --> it converts the variable in string
.to_i --> it converts the variable in number
var1 = 5
var2 = '12'
puts var1.to_s + var2 --> it will print -- 512 (as string and not as number)
var1 = 5
var2 = '12'
puts var1 + var2.to_i --> it will print -- 17 (as number and not as string)

Obtain dates from input of keyboard

puts 'What\'s your name?'
nome = gets (the function gets waits the inserting of dates from keyboard, but it leaves a space, i)
puts 'Ciao ' + nome + 'è bello conoscerti.' --> it will print --- Ciao Marco è
--> bello conoscerti.
nome = gets.chomp (writing chomp it's not considered the final space)

Methods or functions

If the objects (such as strings, entires and comma mobile) are names in Ruby, then the methods are like the verbs. Like every verb it needs of a name, every method needs of an object.

Now many methods:
.reverse (the variable is written on the contrary)
.length (writes in entire numbers "and not in string" the length of variable)
.to_s (it converts the variable in string)
.to_i (it convert the variable in entire)
.upcase (it writes the variable in upper case)
.downcase (it writes the variable small)
.swapcase ( )
.capitalize (it writes the first letter in upper case)
.center (it insert the variable at center)
.ljust (it insert the variable on the left)
.rjust (it insert the variable on the right)
.times (repeat the variable more times -->see do)
.join (join ---> see the Array)
.slice (get the character for character of a string)

Random

puts rand (give me random numbers with the comma)
puts rand(1000) (give me entire numbers from 0 to 999)
puts

Control of flow

> greater
< minor
== equal at
!= different equal
>= minor equal
<= greater equal
|| or
&& and

IF ELSIF ELSE END (end it always goes inserted to the end of every if and while)

```
a = 10
b = 20
if a > b
   puts 'a is greater then b'
elsif a == b
   puts 'a is equal at b'
else
   puts 'a is minor than b'
end
```

WHILE END (end it always goes inserted to the end of every if e while)

```
puts 'Indovina il numero'
a = gets.chomp //the inserted number will be considered as string and not as number
while a != '100'
   puts 'Hai inserito il numero sbagliato'
   a = gets.chomp
end
puts 'Hai indovinato il numero'
```

Array and Iterators

```
                0          1          2         3
citta = ['Lucca', 'Firenze', 'Pistoia', 'Siena']
```

puts citta ---> it will print -- tutte le città dall'alto verso il basso

puts ' ' ---> prints an empty row
puts citta[0] ---> it will print -- Lucca
puts citta[2] ---> it will print -- Pistoia
puts citta[4] ---> it will print -- nil that is nothing because the array 4 doesn't exist

each is a method used for array, it goes always used together to do and end do it not a method.

```
              0         1            2         3
citta = ['Lucca', 'Firenze', 'Pistoia', 'Siena']
citta.each do |italia|
puts 'Io ho abitato in queste ' + italia + '.!'    ---> it will print  --  all the
                                                   ---> towns toward down
end
```

do, we see another use of do (do is not a method)

```
3.times do
puts 'ciao'   ---> it will print --  3 times ciao from up toward down
end
```

```
              0         1            2         3
citta = ['Lucca', 'Firenze', 'Pistoia', 'Siena']
50.times do
puts citta  ---> it will print --  50 times Lucca Firenze Pistoia Siena from up to down
puts [ ]    ---> it will print --  50 times nothing, that is it won't do anything.
end
```

Use methods to enlivend the array
.push (insert)
.pop (remove the last object from array)
.length (it says from how many elements in number is composed the array)

```
citta=[ ]
citta.push 'Lucca'    --->  insert Lucca on array
puts citta.length     --->  the array is composed from 1 object
citta.push 'Firenze'  --->  insert Firenze in array
citta.push 'Pistoia'  --->  insert Pistoia in array
puts citta.length     --->  the array is composed from 3 objects
citta.pop             --->  delete Pistoia, that is the last object, from array
puts citta.length     --->  the array is composed from 2 objects
```

We learn to write our methods or functions.

A function starts with def and ends with end. If the objects in ruby are like the names in English, the methods are like the verbs.
The methods have NOT to start with the letter upper case
This below is an example of a method

```
def prova
  puts 'ciao'
end
```
prova ---> it will print -- ciao

The local variables live only inside the method, if you use them out of method, you will have an error.
The local variables (inside the method) and outside from this, can have the same name, but they will never interfere one with the other.
We use 1 local variable and 1 parameter (thanks to local variable called number):

```
def prova numero
  puts 'ciao' * numero
end
```
prova 3 ---> it will print -- ciao 3 times
prova   ---> will give an error because it misses a parameter (wrong number of arguments (0 for 1) (ArgumentError)

We use 2 local variables “number and double” and we pass 3 parameters, redouble is always the name of method.

```
def raddoppia numero
  doppio = numero*2
  puts numero.to_s+' moltiplicato fa ' +doppio.to_s
end
```
raddoppia 7 ---> it will print -- 6 moltiplicato fa 12

The last value returned, gives 1 method, it’s the last expression valued and not the last string on method. Infact (sono una stringa) is between apexes, that means, that it is an expression.

We use only 1 local variable and 1 expression at the end of method.
```
def raddoppia numero
  puts 'ciao'*numero
  'sono una stringa'
end
```
x = raddoppia 3 ---> it will print -- ciaocioacioa
puts x   ---> it will print -- sono una stringa

Now we pass 3 dates to a method we have created, before we declare how many variables we use declaring the method (we can put them between parenthesis), after we recall the method passing to it the other variables, enough to consider the number (in this case are 3):

```
def prova a, b, c
```

```
        puts 'buon giorno ' + a + b
end
c = 'Cristian '
d = 'Massimiliano '
e = ‘Roberto’
prova c, d, e    ---> it will print  --  buon giorno Cristian Massimiliano Roberto
```

Other example with method for ftp:

```
require 'net/ftp'
def f (ip, ut, pa)
    ftp = Net::FTP.new(ip)
       ftp.login(ut, pa)
       files = ftp.chdir('/directoryinventata/')
       ftp.putbinaryfile('/Aggiornamento/file.tgz', 'file.tgz', 1024)
       ftp.putbinaryfile('/Aggiornamento/file1.tgz', 'file1.tgz', 1024)
       ftp.putbinaryfile('/Aggiornamento/file.sh', 'file.sh', 1024)
    ftp.close
end

puts 'Inserisci l\' ip'
ip = gets.chomp
puts 'Inserisci utente'
utente = gets.chomp
puts 'Inserisci password'
password = gets.chomp

f ip, utente, password
```

Classes and Objects

A category of objects such as dogs is called class, and same specified object belonging to a class is called instance of the class.
The class starts always withe the letter upper case.
Dog is a class
The object belonging to the class Cane is called instance.

First we define the characteristics of the class, later we create an instance.

```
class Dog
  def speak
    puts "Bau"
  end
end
```

The class Dog starts with class and ends with end
speak is a method of the class

```
lassie = Dog.new
```

lassie is the instance of the class Dog
The method new create a new object, that is a new instance of the class.
Now we decide to give to lassie a property (speak)
lassie.speak

Or we can exec temporarily the class like that:
(Dog.new).speak
or
Dog.new.speak

pochi is an instance of the class (it's a species of variable), instead Dog.new.speak will disappear just executed.

Other example

```
class Greeter
        def initialize name= "World"
          @name = name
        end

        def say_hi
           puts "Hi #{@name}!" # "Hi " + @name + "!"
        end

        def say_bye
            puts "Bye #{@name}, come back soon." # "Bye " + @name + ",come back soon."
        end
end
```

Greeter is the class
@name is a variable of instance, available for all methods of a class

We create an object:

```
g = Greeter.new("Pat") #has been created the object g
g.say_hi #we use the object using the methods
g.say_bye #we use the object using the methods


#g.@name #It's not possible use it


class Greeter
        attr_accessor :name
end
```

#How modify a class, doesn't change the objects that already exist, but it has effect on new objects that will be created. It has effect on variables of object.
Greeter.new("Andy")

```
g.respond_to?("name")
g.respond_to?("name=")
g.say_hi #Stamperà Hi Pat !
g.name="Betty"
g
g.name
g.say_hi #Stamperà Hi Betty!
```

Example complete:

```
class MegaGreeter
  attr_accessor :names

  # Create the object
  def initialize(names = "World")
    @names = names
  end

  # Say hi to everybody
  def say_hi
    if @names.nil?
      puts "..."
    elsif @names.respond_to?("each")

      # @names is a list of some kind, iterate!

 @names.each do
        |name|
        puts "Hello #{name}!"
      end
    else
      puts "Hello #{@names}!"
    end
  end

  # Say bye to everybody
  def say_bye
    if @names.nil?
      puts "..."
    elsif @names.respond_to?("join")
      # Join the list elements with commas
      puts "Goodbye #{@names.join(", ")}.  Come back soon!"
    else
      puts "Goodbye #{@names}.  Come back soon!"
    end
  end

end

if __FILE__ == $0
```

```
  mg = MegaGreeter.new      #I create the object mg from class MegaGreeter

  mg.say_hi                         #It prints Hello World !
  mg.say_bye                  #Goodbye World.  Like back soon!

  # Change name to be "Zeke"
  mg.names = "Zeke"     #I change the value to variable of the object

  mg.say_hi             #It prints Hello Zeke!
  mg.say_bye             #It prints Hello Zeke. Like back soon!

  # Change the name to an array of names
  mg.names = ["Albert", "Brenda", "Charles", "Dave", "Englebert"]
  mg.say_hi                         #It prints Hello Albert!
                                #It prints Hello Brenda!
                                #It prints Hello Charles!
                                #It prints Hello Dave!
                                #It prints Hello Englebert!
  mg.say_bye   #stampa Goodbye Albert, Brenda, Charles, Dave, Englebert.
Come back soon!

  # Change to nil
  mg.names = nil       #I change name at variable leaving empty
  mg.say_hi                   #It prints ...
  mg.say_bye                   #It prints  ...
end
```

Stamperà a video:
Hello World!
Goodbye World. Come back soon!
Hello Zeke!
Goodbye Zeke. Come back soon!
Hello Albert!
Hello Brenda!
Hello Charles!
Hello Dave!
Hello Englebert!
Goodbye Albert, Brenda, Charles, Dave, Englebert. Come
back soon!
...
...

Write and read a file

Write in a file:

file = ‘prova.txt’ ---> I insert the name of file in a variable
testo = 'Prova di scrittura in un file' ---> I insert the text
File.open file, 'w' **do** |f| ---> I use the method File.open I read the

```
                                ---> file, with the cycle do I read byte for byte
  f.write testo                 ----> I write the string of variable testo
end                             ---> I close the cycle do
```

Read a file (2 ways):

1)

```
file = 'prova.txt'          --->I insert the name of file in a variable
lettura = File.read file    ---> I use the method File.read to read the variable file
puts lettura                --> Using puts I read the variable lettura
```

or

2)

```
IO.foreach("prova.txt") {|line| puts line } ---> For each line read from prova.txt,
                                             ---> it prints on video 1 line
```

Read a row of 1 file and get the characters:

```
arr = IO.readlines("rimmer.txt")          ---> I read of file rimmer.txt
a = arr[6]                      ---> I insert the row 5 of array into variable a
puts a.slice(9..14)       ---> I print the letters from 9 to 14 of the row 5 of array
```

Write and read of file using the method yaml and using an array

```
require 'yaml'                          --- > I use the method yaml
array = ['ciao', 'io', 'sono', 'Luca']  --- > I insert the dates in a array
test = array.to_yaml                --- > Using the method yaml, I insert the array
                                    ---> into variable testo
puts test                           --- > We print on video the variable test

file = 'rimmer.txt'                 ---> We create the file rimmer.txt
File.open file, 'w' do |f|          ---> I use the method File.open in writing
                                    ---> to write into variable file with do

  f.write test                      ---> I write the variable text (the array)
into file
end                                 ---> I finish the cycle end

lettura = File.read file            ---> Into variable lettura I insert the read
made of file rimmer.txt
leggiarray = YAML::load lettura     ---> Using YAML::load I read the file
                                    ---> lettura that I insert in a variable
puts leggiarray                     ---> I read leggiarray
```