Curso: Ruby Básico

Rodrigo di Lorenzo Lopes
rodrigo.lorenzo@abril.com.br

Celestino Ferreira Gomes
contato@tinogomes.com

3 de Maio de 2013

Conteúdo

1 Introducao: Ruby.new

1.1 Ideia do Curso

- Apresentar a linguagem
- Técnicas de metaprogramação
- Problemas e desafios

1.2 Sobre Ruby

Ruby é uma linguagem de programação *interpretada, de* tipagem dinâmica e *forte, com* gerenciamento de memória automático, originalmente planejada e desenvolvida no Japão em 1995, por Yukihiro "Matz"Matsumoto, para ser usada como linguagem de script.

Matz queria uma linguagem de script que fosse mais poderosa do que Perl, e mais orientada a objetos do que Python. Ruby é primariamente, uma linguagem *orientada a objetos, mas suporta outros paradigmas de programação, como* funcional, *imperativa e* reflexiva.

Foi inspirada principalmente por Python, Perl, Smalltalk, Eiffel, Ada e Lisp, sendo muito similar em vários aspectos a Python.

fonte: Wikipedia - http://bit.ly/wiki-ruby

1.3 Inslatação

Antes de prosseguir com o curso, vejamos como instalar o interpretador da linguagem. Ruby é hoje portado para os principais sistemas operacionais:

1.3.1 Windows

Baixar o executável de instalação em `http://rubyinstaller.org/downloads/`

1.3.2 Linux (Debian / Ubuntu)

```
$ [sudo] apt-get install ruby1.9.1
```

1.3.3 Mac (via Homebrew)[1]

```
$ brew install ruby
```

Podemos verificar a versão já instalada do Ruby que está no `PATH`.
Verificar a versão de ruby instalada, digite `ruby --version`.

```
$ ruby --version
```

Você receberá uma mensagem como essa:

```
ruby 1.9.3p374 (2013-01-15 revision 38858) [x86_64-darwin10.8.0]
```

`ruby -v` também imprime a versão do ruby, porém ativa o modo *verbose e permite executar scripts.*

O código-fonte ruby é um arquivo de texto como qualquer outro. Você pode utilizar seu editor de texto favorito, mas aqui a lista dos principais editores e IDEs disponíveis para a linguagem:

- *Sublime Text 2*
- *Vim*
- *Emacs*
- *TextMate 2 (Mac)*
- *Netbeans 6.9.1*
- *Aptana*

[1] Linux e Mac normalmente já vem com uma instalação de Ruby.

1.4 Ruby.new

Ao longo desse curso, veremos comos os paradigmas de programação são aplicados em Ruby. Por enquanto, para exemplificar o pardigma orienta-a-objetos, observe o exemplo:

```
5.times { print "Ola!" }
```

```
for (int i=0; i <10; i++) { printf("Ola!");}
```

A consistência de "tudo ser objeto"[1] parte dos príncipios de legibilidade e expressividade (acima por exemplo da eficiência em tempo de execução). É quase possível entender o código sem mesmo conhecer a linguagem. Veja o exemplo abaixo:

```
exit unless "restaurante".include? "aura"
['toasty', 'cheese', 'wine'].each
      { |food| print food.capitalize }
```

1.5 Sobre a sintaxe

Algumas itens sobre a sintaxe são apenas conveções. Embora, pode-se utilizar uma conveção arbitrária para nomes de classes e métodos, é importante manter a consistência com os programas da linguagem.

- *Espaços, tabulações e blocos de comentários serão ignorados pelo interpretador.*
- *Ponto-e-vírgula e nova linha é considerado um novo comando. Porém, se for encontrado um operador, como '+', '-', ou o sinal de '; indica que a próxima linha é continuação do comando atual.*
- *Os identificadores são nomes de variáveis, constantes e métodos e é* case-sensitive, ou seja, 'DNS' e 'Dns' são duas constantes diferentes.
- Deve-se usar letras_minúsculas para nomes de variáveis e métodos. 'nome_completo'
- Deve-se usar LETRAS_MAIÚSCULAS para nome de CONSTANTES. 'MAX_ATTENDEES = 100'
- Deve-se usar *CamelCase para nome de Classes. 'NotificationMailer'*

1.6 Running Ruby

Vamos criar uma arquivo Hello World! do Ruby.

```
$ echo 'puts "hello_world"' > hello_word.rb
$ ruby hello_word.rb
```

A saída será:

```
hello world
```

Como ocorre com interpretadores como Perl, é possível passar o script por argumento na linha de comando:

```
$ ruby -e 'puts "hello_world"'
```

A saída será a mesma:

```
hello world
```

O ruby tem ainda um shell interativo (IRB) que pode ser usado para realizar testes:

```
$ irb
>> puts 'hello world'
hello world
=> nil
>>
```

2 Estruturas básicas

Nessa sessão veremos as estruturas básicas da linguagem: estruturas de controle e objetos básicos que são instanciados pelo próprio interpretador. Em Ruby, tudo que pode ser atribuído a uma variável é um objeto e todo objeto tem uma classe. Numa linguagem orientada a objetos, a forma básica de interagir com objetos é por meio de mensagens. Cada objeto "entende" um certo conjunto de mensagens definidos em métodos. A executação de um método se dá no envio da mensagem com o nome do método para o objeto. Em particular, toda classe "entende"todos os métodos da classe Object. Veremos tudo isso em detalhes. Mas tomemos agora o método class e object. Esse método devolve a classe que o objeto pertence (ou qual classe foi utilizada para instanciar o objeto). Veja:

```
>> 1.class
=> Fixnum
>> "texto".class
=> String
>> :simbolo.class
=> Symbol
>> true.class
=> TrueClass
>> false.class
=> FalseClass
>> nil.class
=> NilClass
>> Object.class
=> Class
```

```
>> Class.class
=> Class
```

Já falei que em Ruby, TUDO é objeto?

```
>> 1.methods.count
=> 130
> 1.methods.sort
=> [:!, :!=, :!~, :%, :&, :*, :**, :+, :+@, :-, :-@, :/, :<, :<<, :<=, :<=>, :
:=~, :>, :>=, :>>, :[], :^, :__id__, :__send__, :abs, :abs2, :angle, :arg, :be
:ceil, :chr, :class, :clone, :coerce, :conj, :conjugate, ...]
>> 'texto'.methods.count
=> 162
>> 'texto'.methods.sort
=> [:!, :!=, :!~, :%, :*, :+, :<, :<<, :<=, :<=>, :==, :===, :=~, :>, :>=, :[]
:__id__, :__send__, :ascii_only?, :between?, :bytes, :bytesize, :byteslice, ..
```

2.1 Comentários

Comentários são trechos em seu código que não serão processados pelo interpretador. Serve para documentar seu código.

2.1.1 Uma linha

```
# Esta linha eh um comentario.
1 + 1 # este texto a direita do sinal de # tambem eh um comentario.
```

2.1.2 Múltiplas linhas

```
=begin
O texto envolvido por =begin e =end é comentário.
Mas para isto funcionar, o =begin e =end devem estar
na exterma esquerda do seu código, ou seja, na coluna
0 (zero).
=end
```

2.2 Números

2.2.1 Inteiros

```
123                     # Inteiro (Fixnum)
-123                    # Inteiro negatico (Fixnum)
1_123                   # Inteiro (Fixnum)
123_456_789_123_456_789 # Inteiro (Bignum)
0xAB                    # Número Hexadecimal (170)
0377                    # Número Octal (255)
0b001001                # Número binário (9)
```

2.2.2 Pontos Flutuantes

```
123.45                          # Número com ponto flutuante (Float)
1.2e-3                          # Número com ponto flutuante (0.0012)
```

2.2.3 String 'single quotes'

```
>> puts 'texto'
texto
>> puts 'texto'.length
5
>> puts 'texto'.upcase
TEXTO
>> puts 'tex'.+('to')
texto
>> puts 'tex' + 'to' # syntax sugar
texto
>> puts 'tex' << 'to'
texto
>> String.new << 'texto'
texto
>> 'tex%s' % 'to'
texto
```

*Para usar os caracteres '' ou '
', você pode usar sequência de* escape *'
' e '

'.*

```
>> puts 'texto \'\\'
texto '\
```

2.2.4 String 'double quotes'

Existe uma diferença entre construir strings com aspas simples e aspas duplas. Strings montadas com aspas duplas, aceitam interpolação de conteúdo para construir a string final.

```
>> puts "o resultado de 1 + 1 é #{ 1 + 1 }."
o resultado de 1 + 1 é 2.
>> puts 'o resultado de 1 + 1 é #{ 1 + 1 }.'
o resultado de 1 + 1 é #{ 1 + 1 }.
```

O valor da expressão a ser interpolada, será o resultado do método 'to_s' do objeto.

2.2.5 Sequências de *escape*

```
>> puts "      world\rhello"
hello world
>> puts "\thello \b\sworld"
     hello world
>> puts '\thello \b\sworld'
\thello \b\sworld
```

- '\"' – double quote
- '\\' – single backslash
- '\a' – bell/alert
- '\b' – backspace
- '\r' – carriage return
- '\n' – newline
- '\s' – space
- '\t' – tab

2.2.6 String multiplas linhas

```
>>> puts <<DOC
 Esta é uma string em múltiplas linhas.
      * item
      * item
      * item
 DOC
```

Resultado:

```
Esta éuma string em múltiplas linhas.
    * item
    * item
    * item
```

Se quiser identar o finalizaror, para usar '¡¡-'.

```
>>> puts <<-DOC
Esta éuma string em múltiplas linhas.
    * item
    * item
    * item
    DOC
```

Resultado:

```
Esta éuma string em múltiplas linhas.
    * item
    * item
    * item
```

2.3 Símbolos

Os símbolos são ideais para serem usados como chave em 'Hash'.

```
:x, :y, :chave
```

Símbolos são alocados uma única vez: ':a.object_id' durante uma execução sempre retornara o mesmo valor. Isso não acontece com string.

O método 'equal?' só devolve 'true' se dois objetos são de fato o mesmo objeto (e instâncias da mesma classe com valores iguais).

```
1.equals?(1)                # => true
:key.equals?(:key)          # => true
"texto".equals?("texto")    # => false
```

- Variáveis

  ```
  x, y, taxa_do_lixo2
  ```

- Números

  ```
  1, -1.2, 6.03e-23
  ```

- String

  ```
  "alguma coisa assim"
  %q(veremos outras formas de declarar strings)
  ```

- Symbols

  ```
  :x, :y, :isso_parece_uma_string
  ```

- Constantes

  ```
  EmpireStateBuilding, NEA, PI
  ```

- Objetos especiais

  true, **false**, **nil**

Símbolos são alocados uma única vez: :a.object_id durante uma execução sempre retornara o mesmo valor. Isso nao acontece com string. O método equal? so devolve true se dois objetos são de fato o mesmo objeto (e instâncias da mesma classe com valores iguais).

2.4 Métodos (mensagens)

```
i = 1
texto = "um texto"; puts texto
a = b = c = 0
1 == 2            # sugar syntax!!!
# metodo de classe
1.methods  # lista todos os metodos daquele objeto
1.send(:even?) # outra forma de enviar mensagens
def fibo(n = 1)
    fibo(n-2) + fibo(n-1) if n >= 2
end
def self.log
  puts "metodo de classe"
end
```

Lembre-se ... voce pode redefinir um método Quase tudo e objeto

2.5 Estruturas de Controle - if

2.6 Exemplo Completo

```
if count > 10
  puts "Try again"
elsif tries == 3
  puts "You lose"
  puts  Number:"
end
```

2.7 Exemplo Simples

```
if radiation > 3000
  puts "Danger"
end
```

2.8 Modificador de Sentenca

```
puts "Danger, Will Robinson" if radiation > 3000
```

2.9 Estruturas de Controle - case

```
print "Enter your grade: "
grade = gets.chomp
case grade
when "A"
  puts 'Well done!'
when "B"
  puts 'Try harder!'
```

```
when "C", "D"
  puts 'You need help!!!'
  puts "You just making it up!"
 end
```

2.10 Estruturas de Controle - while

```
while weight < 100 and numPallets <= 30
  pallet = nextPallet()
  weight += pallet.weight
  numPallets += 1
end
```

2.11 Modificador de Sentenca

```
square = square*square  while square < 1000
```

2.12 Estrutura de Controle - for

```
for i in 0..5
   puts "Value is #{i}"
end
```

2.13 Estrutura de Controle - until

```
until weight >= 100 ||numPallets > 30
  pallet = nextPallet()
  weight += pallet.weight
  numPallets += 1
end
```

2.14 Modificador de Sentenca

```
square = square*square  until square >= 1000
```

2.15 Desafio - FizzBuzz

Escreva um programa que imprima o número de 1 a 100. Mas, para múltiplos de três, imprima "Fizz" no lugar do número e para múltiplos de cinco imprima "Buzz". Para números que são múltiplos de ambos três e cinco imprima "FizzBuzz".

`http://www.rubyquiz.com/quiz126.html`

2.16 Solução FizzBuzz

```
# Escreva um programa que imprima o numero de 1 a 100.
# Mas, para multiplos de tres, imprima "Fizz" no lugar do
# numero e para multiplos de cinco imprima "Buzz". Para
```

```ruby
# numeros que sao multiplos de ambos tres e cinco
# imprima "FizzBuzz"
# http://www.rubyquiz.com/quiz126.html
# Solucao tosca
1.upto(100) do |i|
  if i % 5 == 0 and i % 3 == 0
          puts "FizzBuzz"
  elsif i % 5 == 0
          puts "Buzz"
  elsif i % 3 == 0
          puts "Fizz"
  else
          puts i
  end
end
```

3 Containers

3.1 Array

```ruby
a = [ 3.14159, "pie", 99 ]
a.type   #         Array
a.length #         3
a[0]     #         3.14159
a << 1
a[3]     #         1
a[-2]    #         99
b = Array.new
b << a   #    [[3.14159, "pie", 99, 1]]
b[0..3] = a    #    [3.14159, "pie", 99, 1]
b[0, 2] = 1    #    [1, 1]
c = %w{a b c d e }  # => ["a", "b", "c", "d"]
```

3.2 Hash

```ruby
h = {'dog' => 'canine', 'cat' => 'feline', 'donkey' => 'asinine'}
h.length         #         3
h['dog']         #         "canine"
h['cow'] = 'bovine'
h[12]    = 'dodecine'
h['cat'] = 99
h          # => {"cow"=>"bovine", "cat"=>99, 12=>"dodecine",
"donkey"=>"asinine", "dog"=>"canine"}

a = [[1, 'a'],[2, 'b'],[3, 'c'], [4, 'd']]
b = Hash[a]
```

```
# => {1=>"a", 2=>"b", 3=>"c", 4=>"d"}
```

3.3 Blocos e Iteradores

Passando blocos

```
(1..12).each { |i| puts i}
[1, 2, 4].each do |i|
    puts i
end
```

Blocos de código

```
(1..20).each{|x| puts x}
```

Influência do Smalltalk:

```
1 to: 20 do: [:x | x printN1]
```

3.4 Métodos de um Enumerable

```
all?, any?, collect, detect, each_cons, each_slice, each_with_index, entries,
enum_cons, enum_slice, enum_with_index, find, find_all, grep, include?, inject,
map, max, member?, min, partition, reject, select, sort, sort_by, to_a,
to_set, zip
```

3.5 Exemplos com Enumeraveis

```
names = %w{ Frye Leela Zoidberg }
names.find {|name|  name.length>4}          # => "Leela"
names.find_all { |name| name.length > 4}
     #=> ["Leela", "Zoidberg"]
names.grep /oidberg/
# => ["Zoidberg"]
names.group_by {|name|  name.length}
   # =>  {4=>["Frye"], 5=>["Leela"], 8=>["Zoidberg"]}
```

3.6 Mais exemplos com Enumeraveis

```
names = %w{ Frye Leela Zoidberg }
names.map {|name| name.downcase}
# => ["frye", "leela", "zoidberg"]
names.reduce {|acc, name| name.length <= 5 ? acc + name : acc }
# => "FryeLeela"
names.join ", "
# => "Frye, Leela, Zoidberg"
```

4 Blocos

4.1 yield

```
def proxy_method
  puts "Calling command at: #{Time.new}"
  yield
proxy_method { puts "hello world proxified! "}
#ou com paremtros
def proxy_method
   yield(Time.new)
proxy_method {|time| puts "hello world proxified  at #{time}"}
```

4.2 call

```
def proxy_method(&method)
  # argumento com & precisa ser o ultimo da lista
  puts "Calling command at: #{Time.new}"
  method.call
proxy_method { puts "hello world proxified! "}
#ou com paremtros
def proxy_method (&method)
   method.call(Time.new)
proxy_method {|time| puts "hello world proxified  at #{time}"}
```

4.3 Proc x Lambda

```
fx = Proc.new {|x| x**2}
fxy = proc {|x,y| x+y}
# calling
fx.call(2) # => 4
fxy[2,3,4] #=> 5
fx = lambda {|x| x**2}
fxy = lambda {|x,y| x+y}
# calling
fx.call(2) # => 4
fxy.call(2,3,4) #=> exception na cara!
Proc.new e proc sao equivalentes
```

4.4 Lambda "Calculus"

Listing 1: "Derivada em Ruby"

```
def d(f)
   lambda {|a|
     h = 0.0000000001 # um valor pequeno para h
     h = h * a        if a < 1 && 0 < a
     (f[a+h]-f[a])/h
   }
f = lambda {|x| x**2}
puts d(f)[4]
```

5 Objetos em Ruby

```ruby
class BookInStock
  def initialize(isbn, price)
    @isbn = isbn
    @price = Float(price)
  end

  def to_s
    "ISBN:#{@isbn}, price: #{@price}"
  end
end
stock = BookInStock.new
# ou
stock = BookInStock.new (1234, 10.39)
#invocando metodo
puts stock.to_s
```

5.1 Variaveis e Escopo

Variáveis Locais	`x name thx1138 _x _26`
Variáveis de Instancia	`@name @X @_ @plan9`
Variáveis de Classe	`@@total @@N @@x_pos`
Variáveis Globais	`$debug $CUSTOM $_ $plan9`
Nomes de Classe	`String BigDecimal`
Constants	`FEET_PER_MILE DEBUG`

5.2 Atributos de instância - forma tradicional

```ruby
class BookInStock
  def isbn
    @isbn
  end

  def isbn=(value)
    @isbn = value
  end

  def price
    @price
  end
```

5.3 Atributos de instância - forma declarativa

```ruby
class BookInStock
  attr_accessor :isbn
```

```
  attr_reader :price
end
```

5.4 Herança

5.4.1 Exemplo de Heranca

```
class SpecialStock < BookInStock
```

5.5 Herança - Singleton Pattern

5.5.1 forma tradicional

```
class Logger
  private_class_method :new
  @@logger = nil
  def Logger.create
    @@logger = new unless @@logger
    @@logger
  end
end
```

5.5.2 módulo Singleton

```
require 'singleton'
class Logger
  include Singleton

  def initialize
    @log = File.open("log.txt", "a")
  end
  def log(msg)
    @log.puts(msg)
  end
Logger.instance.log('message 2')

stock =  BookInStock.new
class << stock
   def alter_price
         price * 1.4
   end
```

5.6 Criando um Enumerable

* Basta implementar o metodo each.

```
class Node
  include Enumerable
  attr_accessor :next, :previous, :v
```

```
 def initialize(v = {})
    @v = v
  end
 def to_s
    v.to_s
  end

linked_list.rb (continuacao)
 def <<(node)
    node.next = self.next
    node.previous = self
    self.next.previous = node unless self.next.nil?
    self.next = node
  end
 def remove
    node = self.previous
    node.next = self.next
    self.next.previous = node
    self
 end

  def each
    node = self.next
    until node == self || node.nil?
      yield node
      node = node.next
    end
  end
```

6 Mais sobre métodos

6.1 Lista de parâmetros

```
def my_new_method                    # No arguments
  # Code for the method would go here
end

def my_other_new_method(arg1, arg2, arg3)  # 3 arguments
  # Code for the method would go here
end

def cool_dude(arg1="Miles", arg2="Coltrane", arg3="Roach")
# defaults
  "#{arg1}, #{arg2}, #{arg3}."
end
```

6.2 Truques com parâmetros

Aridade não definida

```
def varargs(arg1, *rest)
  "Got #{arg1} and #{rest.join(', ')}"

varargs("one")  # "Got one and "
varargs("one", "two") # "Got one and two"
varargs "one", "two", "three" # "Got one and two, three"

def varargs(arg1, hash)
 puts "#{arg1} - #{hash}"
end

varargs (1, :a => 1)
end
```

6.3 Array para argumentos

Expandindo array para parâmetros

```
def five(a, b, c, d, e)
  "I was passed #{a} #{b} #{c} #{d} #{e}"
end

five(1, 2, 3, 4, 5 )          #  "I was passed 1 2 3 4 5"
five(1, 2, 3, *['a', 'b'])  # "I was passed 1 2 3 a b"
five(*(10..14).to_a)          #  "I was passed 10 11 12 13 14"
```

6.4 Proc para bloco

Convertendo proc para bloco

```
print "(t)imes or (p)lus: "
times = gets.chomp
print "number: "
number = gets.to_i

if times =~ /^t/
  calc = proc { |n| n*number }
else
  calc = proc { |n| n+number }
end

puts((1..10).collect(&calc).join(", "))
```

7 Exceptions, Catch and Throw

```
opFile = File.open(opName, "w")

while data = socket.read(512)
  opFile.write(data)
end
```

7.1 Exceptions

```
opFile = File.open(opName, "w")
begin
  # Exceptions raised by this code will
  # be caught by the following rescue clause
  while data = socket.read(512)
    opFile.write(data)
  end

rescue SystemCallError
  $stderr.print "IO failed: " + $!
  opFile.close
  File.delete(opName)
  raise
end
```

7.2 Catching exception

Nomeando a exceção

```
begin
  eval string
rescue SyntaxError, NameError => boom
  # OLHA! sem usar o $!
  print "String doesn't compile: " + boom
rescue StandardError => bang
  print "Error running script: " + bang
end
```

7.3 Ensure

Garante que um bloco é chamado

```
f = File.open("testfile")
begin
  # .. process
```

```
rescue
  # .. handle error
ensure
  f.close unless f.nil?
end
```

7.4 Rescuing a Method

Begin Rescue

```
def some_method
  begin
    danger_danger
    true # good return
  rescue Error
    false # error return
  end
end
```

Better code

```
def some_method
  danger_danger
  true # good response
rescue Error
  false # error response
end
```

7.5 Raise Exceptions

Formas típicas de se lançar uma exceção

```
raise # sem mensagem

# adicionando uma string...
raise "Missing name" if name.nil?

if i >= myNames.size
  raise IndexError, "#{i} >= size (#{myNames.size})"
end

# passando o stackTrace via Kernel::caller
raise ArgumentError, "Name too big", caller
```

7.6 Especializando Exceções

Declaração

```
class RetryException < RuntimeError
  attr :okToRetry

  def initialize(okToRetry)
    @okToRetry = okToRetry
  end
end
```

Como lançar

```
def readData(socket)
  data = socket.read(512)
  if data.nil?
    raise RetryException.new(true), "transient read error"
  end
  # .. normal processing
end
```

7.7 Especializando Exceções II

Tratanto a exceção

```
begin
  stuff = readData(socket)
  # .. process stuff
rescue RetryException => detail
  retry if detail.okToRetry
  raise
end
```

7.8 Catch e Throw

Desvio incondicional com labels

```
def promptAndGet(prompt)
  print prompt
  res = readline.chomp
  throw :quitRequested if res == "!"
  return res
end
```

```
catch :quitRequested do
  name = promptAndGet("Name: ")
  age  = promptAndGet("Age:  ")
  sex  = promptAndGet("Sex:  ")
  # ..
  # process information
end
```

8 Módulos

Uso

1. Criar namespace (evitar conflito de nomes)
2. Mixin (permitir herança de traços – como se fosse uma cópia do conteúdo do módulo no local incluído)

8.1 Declaração

```
module Trig
  PI = 3.141592654
  def Trig.sin(x)
   # ..
  end
  def Trig.cos(x)
   # ..
  end
end
```

8.1.1 Uso

```
require "./trig"
puts Trig.sin(Trig::PI / 3.0)
```

8.2 Mixins

Applying mixin

```
class BigInteger < Number
  # Adiciona metodos de instancia de Stringify
  include Stringify

  # Adiciona metodos de classe de Math
```

```
  extend Math

  # Adiciona um constructor com um parametro
  def initialize(value)
    @value = value
  end
end
```

8.3 Applying mixin

```
bigint1 = BigInteger.new(10)

puts bigint1.intValue    # --> 10

bigint2 = BigInteger.add(-2, 4)
puts bigint2.intValue    # --> 2

puts bigint2.stringify    # --> 'Two'

bigint2.extend CurrencyFormatter
```

9 Pacotes Básicos

9.1 BigDecimal

```
require 'bigdecimal'

BigDecimal.new('1.23) #=> #<BigDecimal:7ffe0b052bc8,'0.123E1',18(18)>
```

9.2 OpenStruct

```
require 'ostruct'
```

9.3 Test

```
require "test/unit"

class TesteFoo  < Test::Unit::TestCase

  def test_eFoo_foo
    assert_same(1, 0, "Que pena")
  end
end
```

9.4 ERB

- Sistema de Template padrão do Ruby
- Uma classe como outra qualquer
- Via linha de comando é possível parsear um arquivo erb

```
require 'erb'

template = ERB.new('1 + 1 = <%= 1 + 1 %>')
template.result # => '1 + 1 = 2'
```

9.5 Net::HTTP

```
require "net/http"
require "uri"
require 'methodize'

def get_page (string)
  uri = URI.parse(string)
  response = Net::HTTP.get_response(uri)
  response.body
end
```

9.6 JSON

```
require "net/http"
require "uri"
require 'json'
require 'methodize'

def get_page (string)
  uri = URI.parse(string)
  response = Net::HTTP.get_response(uri)
  json = JSON.parse(response.body)
  json.extend(Methodize)
end
```

9.7 YAML

9.7.1 Arquivo yaml

```
simple symbol: !ruby/symbol Simple
shortcut syntax: !ruby/sym Simple
symbols in seqs:
  - !ruby/symbol ValOne
```

```
  - !ruby/symbol ValTwo
  - !ruby/symbol ValThree
symbols in maps:
  - !ruby/symbol MapKey: !ruby/symbol MapValue
```

9.7.2 Ruby code

```
require "yaml"

config = YAML.load_file("config.yml") # From file
p config
```

9.7.3 Result

```
{"simple symbol"=>:Simple, "shortcut syntax"=>:Simple,
"symbols in seqs"=>[:ValOne, :ValTwo, :ValThree],
"symbols in maps"=>[{:MapKey=>:MapValue}]}
```

Referências

[1] Martin Abadi and Luca Cardelli. *A Theory of Objects*. Springer, corrected edition, August 1996.