Open Source Frameworks for Rapid Application Development

Marek Krętowski
Krzysztof Bandurski, Tomasz Łukaszuk, Tomasz Rybak

Software Departament
Faculty of Computer Science
Bialystok University of Technology

m.kretowski@pb.edu.pl
k.bandurski@pb.edu.pl, t.lukaszuk@pb.edu.pl, t.rybak@pb.edu.pl

Lecture topic

Ruby

Ruby: Table of content

Introduction

Introduction

Ruby

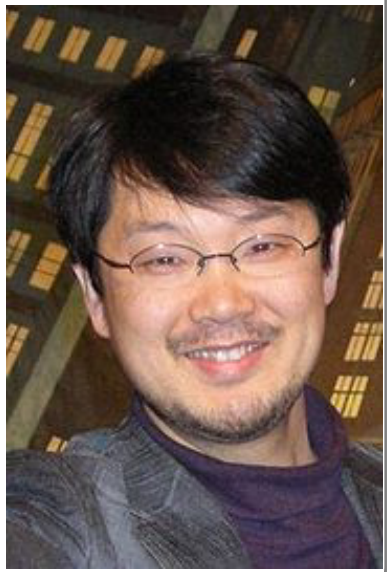
- is a dynamic, reflective, object-oriented programming language
- is based on Perl, Smalltalk, Eiffel, Ada, and Lisp
- combines syntax inspired by Perl with Smalltalk-like features
- supports multiple programming paradigms, including functional, object oriented, imperative and reflective, has a dynamic type system and automatic memory management
- the standard 1.8.7 implementation is written in C, as a single-pass interpreted language
- there is currently no specification of the Ruby language, implementations of the Ruby language: YARV, JRuby, Rubinius, IronRuby, MacRuby and HotRuby, each of them takes a different approach

History I

- **February 24, 1993** - Ruby was conceived by Yukihiro Matsumoto as new language that balanced functional programming with imperative programming

Yukihiro Matsumoto

"I wanted a scripting language that was more powerful than Perl, and more object-oriented than Python. That's why I decided to design my own language"



History II

- **December 21, 1995** - the first public release of Ruby 0.95
- **December, 1995** - the Japanese language ruby-list mailing list
- **December 25, 1996** - Ruby version 1.0
- **1999** - the first English language mailing list ruby-talk
- **September, 2000** - the first English language book ”Programming Ruby”
- **now** - the current stable version is 1.9.2.

Features I

- Thoroughly object-oriented with inheritance, mixins and metaclasses
- Dynamic typing and Duck typing
- Everything is an expression (even statements) and everything is executed imperatively (even declarations)
- Succinct and flexible syntax that minimizes syntactic noise
- Dynamic reflection and alteration of objects to facilitate metaprogramming
- Lexical closures, iterators and generators, with a unique block syntax
- Literal notation for arrays, hashes, regular expressions and symbols
- Embedding code in strings (interpolation)

Features II

- Keyword arguments and default arguments
- Four levels of variable scope (global, class, instance, and local) denoted by sigils and capitalization
- Automatic garbage collection
- First-class continuations
- Strict boolean coercion rules (everything is `true` except `false` and `nil`)
- Exception handling
- Operator overloading
- Built-in support for rational numbers, complex numbers and arbitrary-precision arithmetic
- Custom dispatch behavior (through method_missing and const_missing)

Features III

- Native threads and cooperative fibers
- Full support for Unicode and multiple character encodings (as of version 1.9)
- Native extension API in C
- Interactive Ruby Shell (a REPL)
- Centralized package management through RubyGems
- Implemented on all major platforms
- Large standard library

Semantics

Ruby:

- is object-oriented language (every data type is an object, even integers, booleans, and `nil`, every function is a method)
- supports inheritance with dynamic dispatch, mixins and singleton methods
- though it does not support multiple inheritance, classes can import modules as mixins
- supports procedural syntax, but all methods defined outside of the scope of a particular object are actually methods of the Object class
- has been described as a multi-paradigm programming language (procedural programming, object orientation, functional programming)
- has support for introspection, reflection and metaprogramming
- interpreter-based threads
- dynamic-typing

Syntax I

Ruby syntax:

- broadly similar to Perl and Python
- class and method definitions are signaled by keywords

  `class ... end`, `def ... end`
- variables are not obligatorily prefixed with a sigil; when used, the sigil changes the semantics of scope of the variable

  `$` is prefixed to global variables

  `@` is prefixed to instance variables

  `@@` is prefixed to class variables
- keywords are typically used to define logical code blocks, without braces (i.e., pair of { and })
- there is no distinction between expressions and statements
- line breaks are significant and taken as the end of a statement (a semicolon may be equivalently used)

Syntax II

- indentation is not significant
- Ruby keeps all of its instance variables completely private to the class and only exposes them through accessor methods (attr_writer, attr_reader, etc), accessor methods in Ruby are created with a single line of code via metaprogramming, invocation of these methods does not require the use of parentheses

Deviations from behaviour elsewhere I

- `99.0 == 99.to_f`
  to denote a floating point without a decimal component, one must follow with a zero digit or an explicit conversion
- `2.5.round` gives 3, `3.5.round` gives 4, `-3.5.round` gives -4
  round to nearest integer, halfway cases away from zero
- `0 ? 1 : 0` evaluates to 1
  all numbers evaluate to true, only `nil` and `false` evaluate to false
- `"abc"[0]` yields 97
  to obtain "a" use `"abc"[0,1]` (a substring of length 1) or `"abc"[0].chr`
- `statement until expression`
  is syntactic sugar over
  `until expression; statement; end`
  so it never runs the statement if the expression is already true
  is equivalent to `while (!(expression)) { statement; }` in C/C++

Deviations from behaviour elsewhere II

- `statement if expression`
  is an equivalent to
  `if (expression) { statement; }`
- operators `and` and `or` for conditional expressions: `and` does not bind tighter than `or`
  Ruby also has expression operators `||` and `&&` which work as expected

Interaction

`irb` - Interactive Ruby Shell, an interactive command-line interpreter which can be used to test code quickly, the Ruby official distribution also includes "irb"

```
1  $ irb
2  irb(main):001:0> puts "Hello, World"
3  Hello, World
4  => nil
5  irb(main):002:0> 1+2
6  => 3
```

Ruby base

Object-oriented programming language

- Everything is an object ... even `nil`, even class (instance of `Class` class)
- `Object` is the default superclass
- Everything is an expression and returns the value
- Everything except `nil` and `false` is true

Variables and constants

- variables: begins with a small letter or the character _, words separated by _
  `my_variable, _var`
- constants: begins with a big letter, best in full capitals
  `MY_CONSTANT`
- global variables: begins with the `$`
  `$global_variable`
- symbols: begins with the `:`
  `:my_symbol`

```
1  f1,f2 = :ruby,:ruby
2  f1.object_id
3  f2.object_id #the same object
4
5  f1,f2 = "ruby","ruby"
6  f1.object_id
7  f2.object_id #two different objects
```

Comments

- One-line comment

```
1  #one-line comment
```

- Multiline comment

```
1  =begin
2    multiline comment
3    multiline comment
4    multiline comment
5  =end
```

Operators

- arithmetic: `+ - * / % **`
- assignment: `= += -= *= /= %= **= |= &= >>= <<= ||= &&=`
- comparison: `== .eql? .equal? === != < > >= <= <=> =~ !~`
- logical: `&& || ! and or not`
- bitwise: `~ | & ^ << >>`
- other: `[] []= .. ... ! not`

```
1  5 == 5.0      # => true
2  5.eql? 5.0    # => false, (the same type and value)
3  5.equal? 5.0 # => false, (the same object_id)
4  5.equal? 5    # => true, (for fixnums and symbols)
5
6  a = false or 5 # => 5, a == false
7  a = false || 5 # => 5, a == 5
8
9  a, b = b, a
10 a, b, c = get_something()
```

Conditional statements

- `if ... [then|:] ... [elsif ...] [else ...] end`
- `... if ...`
- `unless = if not`
- `... ? ... : ...`
- `case ... when ... [else ...] end`

```
1  x = if a > 100
2      5000
3  elsif a > 5
4      5
5  else
6      1
7  end
8
9  puts "Hello" if a > 100
```

```
1  x = case x
2      when 0...5
3          1
4      when 5..100
5          50
6      else
7          10**9
8      end
```

Loops

- `while ... [do] ... end`
- `... while ...`
- `until` = `while not`
- `for ... in ... [do] ... end`
- `break`, `next`, `redo`, `retry`
- `times`, `upto`, `downto`

```
1  while a > 10
2      a /= 3
3  end
4
5  puts "Iteration #{i+=1}" while i < 10
6
7  for i in 1..8
8      puts i
9  end
```

Functions (or methods)

- invoke function:

  `my_function` or `my_function()`
- invoke with arguments:

  `my_function a, b, c` or `my_function(a, b, c)`
- invoke using the return value:

  `a = my_function(a, b, c);puts my_function(a, b, c)`
- defining

```
1  def my_function(a, b, c)
2      do_something
3      statement_return_sth
4  end
```

- default arguments:

  `def my_function(a, b = true) ...`
- list of arguments:

  `def my_function(a, *other_args) ...`

Functions (or methods)

- Bang methods
  - ends with `!`
  - potentially dangerous
  - modify the object
  - may have non-modify counterparts producing new objects
  - e.g. `sort!`, `upcase!`, `reverse!`
- Asking methods
  - ends with `?`
  - usualy returns `true` or `false`
  - e.g. `empty?`, `include?`, `nil?`

Classes

- defining `class ... end`
- name begins with a big letter - CamelCase
- `initialize` method - constructor
- `inspect` method - <anObject:0x83678>
- `to_s` method
- instance variables - begins with `@`, e.g. `@inst_variable`
- class variables - begins with `@@`, e.g. `@@class_variable`
- accessors - `attr_reader`, `attr_writer`, `attr_accessor`
- instance method - `def method_name ... end`
- class method - `def self.method_name ... end`
- levels of access methods: public, protected, private
- `method_missing`

Classes

```ruby
class MyClass
    attr_reader :value
    def initialize(value)
        @value = value
    end
    private
    def introduce
        "My value is #{@value}"
    end
    public
    def self.public_method(arg)
        ...
    end
    def inspect
        "My id is #{object_id}"
    end
    def to_s
        "My type is #{self.class}"
    end
    def method_missing(method_id)
        puts "No method #{method_id}!"
    end
end
```

```ruby
a = MyClass.new(3)

puts a.introduce
# No method introduce

p a
# My id is 54765890

puts a
# My type is MyClass

puts a.b
# No method b
```

Classes - Inheritance

- only one-base
- Object - root
- `self`
- `super`

```
1  class MyClass
2      def introduce_yourself
3          puts "My name is " + self.class.to_s
4      end
5  end
6
7  class YourClass < MyClass
8      def introduce_yourself
9          super
10         puts "Something from YourClass"
11     end
12     def to_s
13         "YourClass"
14     end
15 end
```

Modules and mixins

- Modules create namespace
- may be included in another module or class - `include`
- it is possible to include more than one module (without a hierarchy)

```
1  module A
2      def a1
3          puts 'a1 is called'
4      end
5  end
6  module B
7      def b2
8          puts 'b2 is called'
9      end
10 end
11 module C
12     def c3
13           puts 'c3 is called'
14     end
15 end
```

```
16 class Test
17     include A
18     include B
19     include C
20     def display
21       puts 'included modules'
22     end
23 end
24 object=Test.new
25 object.display
26 object.a1
27 object.b2
28 object.c3
29 # included modules
30 # a1 is called
31 # b2 is called
32 # c3 is called
```

Blocks

- `do ... end`
- `{ ... }`
- are passed to the function
- `yield`

```
1  10.times do |i|
2     puts i
3  end
4
5  10.times {|i| puts i}
6
7  def give_me_something
8     sth = rand.to_s
9     yield(sth)
10 end
11 give_me_something { |x| puts "I've got #{x}" }
```

Lambda expressions

- blocks are not objects, but they can be; class `Proc`
- methods `lambda` and `proc`
- methods can not be passed to other methods, but you can pass objects of `Proc` class
- method can not return an other method, but may return an object of `Proc` class

```
1  hey = lambda { print "hey" }
2  hello = proc do
3      print "hello"
4  end
5
6  hey.call          # => hey
7  hello.call        # => hello
8  hey.class         # => Proc
9  hello.class       # => Proc
```

```
1  p = lambda { |text| print text }
2  p.call("Hop")   # => Hop
3
4  def repeat(how_many, what)
5     while how_many > 0
6          what.call(how_many)
7          how_many -= 1
8     end
9  end
10 l = lambda { |x| print x }
11 repeat(3, l)       # => 321
```

Exceptions

```ruby
begin
    # Do something
rescue
    # Handle exception
else
    # Do this if no exception was raised
ensure
    # Do this whether or not an exception was raised
end

age = 18
raise "Must be 65 or older for Medicare." if age < 66
```

Built-in types

- numbers: Integer, Fixnum, Bignum, Float, Rational
  ```
  7, 1245, -45.0
  ```
- strings
  ```
  "string example", 'other example'
  ```
- ranges
  ```
  (4..34)
  ```
- arrays
  ```
  [3, 6, "text", ['a', 78]]
  ```
- hashes
  ```
  { :water => 'wet', :fire => 'hot', :iron =>
  'cold' }
  ```
- regular expressions
  ```
  /regex*/
  ```

Examples

Hello world

Listing 1: hello.rb

```
1  #!/usr/bin/ruby
2  print "Hello World\n"
```

Listing 2: run in a Ruby shell irb

```
1  puts "Hello World!"
```

Some basic Ruby code

```ruby
# Everything, including a literal, is an object, so this works:

-199.abs
# 199

"ruby is cool".length
# 12

"Your mother is nice.".index("u")
# 2

"Nice Day Isn't It?".downcase.split("").uniq.sort.join
# " 'acdeinsty'
```

Terminal IO

```
1  #!/usr/bin/ruby
2  print "This is the first half of Line 1. "
3  print "This is the second half.", "\n"
4  puts "This is line 2, no newline necessary."
5
6  printf "There were %7d people at the %s.\n", 439, "Auditorium"
7
8  print "Name please=>"
9  name = gets
10 print "Your name is ", name, "\n"
```

```
This is the first half of Line 1. This is the second half.
This is line 2, no newline necessary.

There were     439 people at the Auditorium.

Name please=>Jon Green
Your name is Jon Green
```

Strings

```
1  a = "\nThis is a double quoted string\n"
2  a = %{\nThis is a double quoted string\n}
3  a = %Q{\nThis is a double quoted string\n}
4  a = <<BLOCK
5
6  This is a multi-line double quoted string
7  BLOCK
8  a = %/\nThis is a double quoted string\n/
9
10 a = 'This is a single quoted string'
11 a = %q{This is a single quoted string}
```

Strings

string assignment and concatination

```
1  #!/usr/bin/ruby
2  myname = "Jon Green"
3  myname_copy = myname
4  print "myname      = ", myname, "\n"
5  print "myname_copy = ", myname_copy, "\n"
6  print "\n=========================\n"
7  myname << "-Red"
8  print "myname      = ", myname, "\n"
9  print "myname_copy = ", myname_copy, "\n"
```

the double less than sign (<<) is a Ruby String overload for concatination

```
myname      = Jon Green
myname_copy = Jon Green

=========================
myname      = Jon Green-Red
myname_copy = Jon Green-Red
```

Strings

the `String.new()` method

```
1  #!/usr/bin/ruby
2  myname = "Jon Green"
3  myname_copy = String.new(myname) # <--------------
4  print "myname      = ", myname, "\n"
5  print "myname_copy = ", myname_copy, "\n"
6  print "\n=========================\n"
7  myname << "-Red"
8  print "myname      = ", myname, "\n"
9  print "myname_copy = ", myname_copy, "\n"
```

```
myname      = Jon Green
myname_copy = Jon Green

=========================
myname      = Jon Green-Red
myname_copy = Jon Green
```

Strings

the Ruby String class works like an array of characters

```
1  #!/usr/bin/ruby
2  myname = "Jon was here"
3  print myname[6, 3], "\n"
4  myname[6, 3] = "is"
5  print myname, "\n"
```

```
was
Jon is here
```

Strings

- the addition (+) sign means to add strings together (strings concatenation)
- the multiplication (*) sign means string together multiple copies
- the % method works like the `sprintf()` command in C
- strings comparision with <=>

```
1  mystring = "Jon" + " " + "was" + " " + "here"
2  print mystring, "\n"
3  mystring = "Cool " * 3
4  print mystring, "\n"
5  mystring = "There are %6d people in %s" % [1500, "the Ballroom"]
6  print mystring, "\n"
7  print "frank" <=> "frank", "\n"
8  print "frank" <=> "fred", "\n"
9  print "frank" <=> "FRANK", "\n"
```

```
Jon was here
Cool Cool Cool
There are   1500 people in the Ballroom
0
-1
1
```

Loops

The elipses (...) indicate the range through which to loop. The `for` is terminated by an `end`. You don't need braces for a loop.

```
1  for ss in 1...5
2      print ss, " hello\n";
3  end
```

```
1 hello
2 hello
3 hello
4 hello
```

The `1...5` means 1 TO BUT NOT INCLUDING 5
The `1..5` means 1 through 5

Loops

```
1  presidents = ["Ford","Carter","Reagan","Bush1","Clinton","Bush2"]
2  for ss in 0...presidents.length
3      print ss, ": ", presidents[ss], "\n";
4  end
```

```
0: Ford
1: Carter
2: Reagan
3: Bush1
4: Clinton
5: Bush2
```

Backwards iteration doesn't work in Ruby – it must iterate up.

```
1  presidents = ["Ford","Carter","Reagan","Bush1","Clinton","Bush2"]
2  for ss in 0...presidents.length
3      print ss, ": ", presidents[-ss-1], "\n";
4  end
```

array[-1] is the last item, array[-2] is the second to last, etc.

Loops

`while` loops

```ruby
#!/usr/bin/ruby
ss = 4
while ss > 0
    puts ss
    ss -= 1
end
puts "======================"
while ss < 5
    puts ss
    ss += 1
    break if ss > 2
end
```

```
4
3
2
1
======================
0
1
2
```

Iterators and blocks

Another way to loop through an array is to use an iterator (`each`) and a block (`{|prez| puts prez}`).

```
1  presidents = ["Ford","Carter","Reagan","Bush1","Clinton","Bush2"]
2  presidents.each {|prez| puts prez}
3
4  (1..5).each do |i|
5      puts i
6  end
```

```
Ford
Carter
Reagan
Bush1
Clinton
Bush2
1
2
3
4
5
```

Iterators and blocks

The examples of other ruby iterators:

```
1  [1, 2, 3].collect { |element| element + 1} #=> [2, 3, 4]
2
3  a = [1, 2, 3]                                  #=> [1, 2, 3]
4  a.collect! { |element| element + 1 }           #=> [2, 3, 4]
5  a                                              #=> [2, 3, 4]
6
7  [1, 2, 3, 4, 5, 6].delete_if { |i| i%2 == 0 }    # => [1, 3, 5]
8
9  (36..100).detect { |i| i%7 == 0 }              #=> 42
10
11 9.downto(0) { |i| print i }                    #=> 9876543210
12
13 [3, 6, -5].each_index { |i| print i.to_s + " " } #=> 0 1 2
14
15 (0..30).find_all { |i| i%9 == 0 }              #=> [0, 9, 18, 27]
16
17 (1..6).partition { |i| i%2 == 0 }       #=> [[2, 4, 6], [1, 3, 5]]
18
19 5.times { |i| print "#{i} "}                    #=> 0 1 2 3 4
20
21 1.upto(3) { |i| print i }                      #=> 123
```

Closures

The closure is a block of code passed to the method.

```
1  3.times { print "Bla" } #=> BlaBlaBla
2
3  class Fixnum
4      def times
5          for i in (1...self)
6              yield i
7          end
8      end
9  end
```

Branching

```
1  democrats = ["Carter", "Clinton"]
2  republicans = ["Ford", "Reagan", "Bush1", "Bush2"]
3  party = ARGV[0]
4  if party == nil
5      print "Argument must be \"democrats\" or \"republicans\""
6  elsif party == "democrats"
7      democrats.each { |i| print i, " "}
8  elsif party == "republicans"
9      republicans.each { |i| print i, " "}
10 else
11     print "All presidents were either Democrats or Republicans"
12 end
13 #-------------------------------------------------------------
14 if party != nil
15     democrats.each { |i| print i, " "} if party == "democrats"
16     republicans.each { |i| print i, " "} if party == "republicans"
17     print "All presidents were either Democrats or Republicans"\
18         if (party != "democrats" && party != "republicans")
19 end
```

The `if` keyword must be on the same line as the action

Only a single action can precede the `if` keyword

Collections

Constructing and using an array

```
1  a = [1, 'hi', 3.14, 1, 2, [4, 5]]
2
3  a[2]                  # 3.14
4  a.[](2)               # 3.14
5  a.reverse             # [[4, 5], 2, 1, 3.14, "hi", 1]
6  a.flatten.uniq        # [1, "hi", 3.14, 2, 4, 5]
7  a.pop                 # [4, 5]
8  a                     # [1, "hi", 3.14, 1, 2]
9  a.push('seven')       # [1, "hi", 3.14, 1, 2, "seven"]
10 a.shift(2)            # [1, "hi"]
11 a                     # [3.14, 1, 2, "seven"]
12 a.unshift(4, 'abc')   # [4, "abc", 3.14, 1, 2, "seven"]
13 a[1..3]               # ["abc", 3.14, 1]
14 a[2,4]                # [3.14, 1, 2, "seven"]
15 a[2,3] = []           # []
16 a                     # [4, "abc", "seven"]
```

Collections

Constructing and using an associative array (called hashes in Ruby)

```ruby
hash = { :water => 'wet', :fire => 'hot', :iron => 'cold' }

hash[:fire]           # "hot"

hash.each do |key, value|
      puts "#{key} is #{value}"
end
# water is wet
# fire is hot

hash.select {|key,value| value!='hot'}
# [[:water, "wet"], [:iron, "cold"]]
hash.delete :water
# {:fire => "hot", :iron => "cold"}
hash.delete_if {|key,value| value=='hot'}
# {:iron => "cold"}
```

Regular Expressions

```ruby
string1 = "Ruby regular expressions"
print "yes" if string1 =~ /r.*l/   # yes
print "yes" if string1 !~ /e.*z/   # yes
print "yes" if string1 =~ /^[A-Z]/ # yes

string1 = "I will drill for a well in walla walla washington."
regex = Regexp.new(/w.ll/)
matchdata = regex.match(string1)
while matchdata != nil
    puts matchdata[0]
    string1 = matchdata.post_match
    matchdata = regex.match(string1)
end
# will
# well
# wall
# wall

string1 = "I will drill for a well in walla walla washington."
string1.gsub!(/(w.ll)/){$1.upcase}
# I WILL drill for a WELL in WALLa WALLa washington.
```

Classes

```ruby
class Person
      attr_reader :name, :age
      def initialize(name, age)
            @name, @age = name, age
      end
      def <=>(person) # Comparison operator for sorting
          @age <=> person.age
      end
      def to_s
            "#@name (#@age)"
      end
end

group = [
     Person.new("Bob", 33),
     Person.new("Chris", 16),
     Person.new("Ash", 23)
]
puts group.sort.reverse
```

```
Bob (33)
Ash (23)
Chris (16)
```

Classes

Inheritance

```
1  class Student < Person
2      attr_accessor :index
3          def initialize(name, age, index)
4                  super(name, age)
5           @index = index
6          end
7          def to_s
8                  "Student: #@name (#@age, #@index)"
9          end
10 end
11
12 s = Student.new('Frank', 21, 758745)
```

Classes

Open Classes

In Ruby, classes are never closed: you can always add methods to an existing class.

```ruby
# re-open Ruby's Time class
class Time
        def yesterday
                self - 86400
        end
end

today = Time.now # => Thu Aug 14 16:51:50 +1200 2008
yesterday = today.yesterday # => Wed Aug 13 16:51:50 +1200 2008
```

monkey-patching, duck punching

"Well, I was just totally sold by Adam, the idea being that if it walks like a duck and talks like a duck, it's a duck, right? So if this duck is not giving you the noise that you want, you've got to just punch that duck until it returns what you expect." – Patrick Ewing

Unconstant number of arguments passed to the function

The last parameter can be started from the mark *, which means that any number of parameters will be transformed into an array.

```
1  def reverse_array(*b)
2      if b.size == 1
3          b
4      else
5          reverse_array(*b[1..-1])+[b[0]]
6      end
7  end
8
9  print reverse_array("!\n","ld","wor",", ","llo","He")
10 # Hello, world!
```

Marshalling

`Marshal` class, methods `dump` and `load`

```
1  sth = [3, {:a => "abc"}, "the_end"]
2  File.open("file.dat", "w+") do |f|
3      Marshal.dump(sth, f)
4  end
5
6  File.open("file.dat") do |f|
7     @sth = Marshal.load(f)
8  end
```

Additional informations

Implementations

- Ruby 1.9 - has a single working implementation written in C that utilizes a Ruby-specific virtual machine
- Ruby 1.8 - has two main implementations: the official Ruby interpreter often referred to as the Matz's Ruby Interpreter or **MRI** (the most widely used), and **JRuby**, a Java-based implementation that runs on the Java Virtual Machine
- other implementations: Cardinal, IronRuby, MacRuby, MagLev, Rubinius, Ruby.NET, XRuby, HotRuby (runs Ruby source code on a web browser and Flash)
- Ruby is available on many operating systems such as Linux, Mac OS X, Microsoft Windows, Windows CE and most flavors of Unix

Repositories and libraries

- Ruby Application Archive (RAA) - http://raa.ruby-lang.org/
- RubyForge - http://rubyforge.org/, as of November 2009, it hosts over 8,000 projects and has over 41,000 registered users
- RubyGems - a package manager for the Ruby programming language that provides a standard format for distributing Ruby programs and libraries
- GitHub

References

- http://en.wikipedia.org/wiki/Ruby_(programming_language)
- Ruby Basic Tutorial by Steve Litt (http://www.troubleshooters.com/codecorn/ruby/basictutorial.htm)
- http://apohllo.pl/texts/1_ruby_basics_pl.pdf by Aleksander Pohl
- ...