

Getting **started** with

# **Ruby** Topic Maps

**Benjamin Bock**



# Why Ruby?

**interpreted, object-oriented  
programming**

**features procedural and  
functional paradigm**

**everything is an object**

**dynamic and (but) strong typed**



# What about Rails?

**Model-View-Controller**

**Convention over Configuration**

**Optimized for programmer  
happiness and productivity**



# ... and Ruby Topic Maps?

**Web 2.0 is about integration**

**Ruby and Ruby on Rails are big players there**

**Topic Maps nonexistent in the Ruby world**



# Schedule

- Introduction**
- Software distribution**
- Programming Ruby**
- Ruby Topic Maps**
- Rails Basics**
- Using RTM in Rails**



# Command line tool

## irb: Interactive RuBy

### Windows:

Click start -> run -> "cmd" -> "irb"

### Mac OS X:

Start the Terminal -> type "irb"

### Linux:

Open a shell -> type "irb"

### That's what you get:

```
irb(main):001:0>
```



# Calculator

```
irb(main):001:0> 1+2
```

```
irb(main):002:0> 3*4
```

```
irb(main):003:0> 5**3
```

```
irb(main):004:0> Math.sqrt(81)
```



# Simple Types

# comment until end of line

"String"

1 # number

1.0 # floating point number

1\_000\_000\_000\_000 # BigNum

:symbol # these are important!

true, false, self, nil



# Even More Types

`[1,2,3]` # Array

`{ :key => "value" }` # Hash

`/regular expression/`

`(1..10)` # exclusive Range

`(1...10)` # inclusive Range

# Files are also basic types,

# but what does that mean?



# Numbers

123 1\_234 123.45 1.2e-3

0xffff (hex) 0b01011 (binary)  
0377 (octal)

?a # ASCII character

?\C-a # Control-a

?\M-a # Meta-a

?\M-\C-a # Meta-Control-a



# Strings

```
'no interpolation'
```

```
"#{interpolation}, and backslashes\n"
```

```
%q(no interpolation)
```

```
%Q(interpolation and backslashes)
```

```
%(interpolation and backslashes)
```

```
`echo command interpretation with  
interpolation and backslashes`
```

```
%x(echo command interpretation with  
interpolation and backslashes)
```

```
`ls` # returns output of system command ls
```



# Methods

```
def hello  
  puts "hello"  
end
```

```
def salute(who)  
  puts "Hello #{who}"  
end
```

```
def salute(who="world")  
  puts "Hello #{who.capitalize}"  
end
```



# Method Invocation

```
method  
obj.method  
Class::method
```

```
# keyword parameters: one argument for def method(hash_arg):  
method(key1 => val1, key2 => val2)
```

```
becomes: method(arg1, arg2, arg3):  
method(arg1, *[arg2, arg3])
```

```
#as ugly as you want it to be:  
method(arg1, key1 => val1, key2 => val2, *splat_arg) #{ block }
```

```
# A bit more formal:  
invocation := [receiver ('::' | '.') name [ parameters ] [ block ]  
parameters := ( [param]* [, hashlist] [*array] [&aProc] )  
block      := { blockbody } | do blockbody end
```



# Everything is an Object

```
nil.class
```

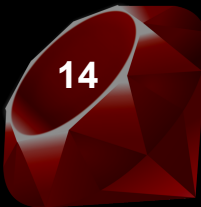
```
true.object_id
```

```
2.75.ceil
```

```
5.times do # this a preview,  
  details later
```

```
  puts "I like Ruby!"
```

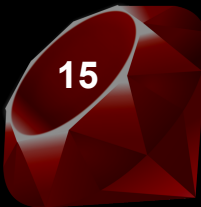
```
end
```



# Everybody is valuable

```
# ... and everything has a value

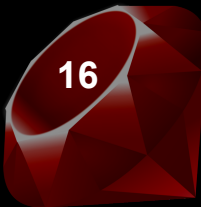
# often no "return" statements needed
def value
  5
end
x = value # x = 5
y = if 1 != 2 # even if has a value
  "right"
else
  "wrong"
end
# y == "right"
```



# But what if ...

```
if condition [then]
  # true block
elsif other_condition # not elseif!
  # second true block
else
  # last chance
end

# also:
z = condition ? true_value : false_value
```

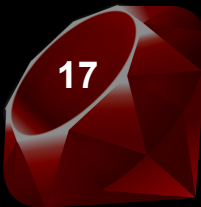


# Can you tell me the truth?

```
# Yes, it's 42
```

```
if 0 then  
  "this is the case"  
else  
  "but this will not happen"  
end
```

```
# Only false and nil are not true!  
# That's handy for == nil checks.
```



# the case is as follows

```
case my_var
  when "hi", "hello", "hallo", "salut"
    puts "it was an greeting"
  when String
    puts "it was a string..."
  when (1..100)
    puts "A number between 0 and 100"
  when Numeric
    puts "A number not in the range above"
  else
    puts "What did you give me here?"
  end
end
```



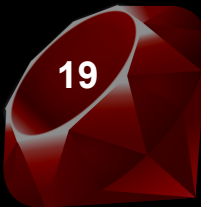
# Back to the Question

```
# there are other conditionals
```

```
return false if you_dont_like_to_answer
```

```
unless you_are_already_asleep  
  i_will_go_on  
end
```

```
# this is not a loop!
```



# Iteration Iteration Iteration Iteration

```
5.times { |x| puts x }  
["alice", "bob"].each do |name|  
  puts "Hello #{name}"  
end  
  
list = [6,7,2,82,12]  
for x in list do  
  puts x  
end
```



go\_on until audience.asleep?

```
loop do
  body
end
```

```
{while,until} bool-expr [do]
  body
end
```

```
begin
  body
end {while,until} bool-expr
# we have: break, next,redo, retry
```



# Blocks (a.k.a. Closures)

```
search_engines =  
%w[Google Yahoo MSN].map do |engine|  
  "http://www." + engine.downcase +  
  ".com"  
end  
  
%w[abcde fgghi jkl mn op q].sort_by { |  
  w|  
  w.size  
}
```



# Object Oriented Constructs

```
$global_variable
module SomethingLikePackageInJava
  class SameLikeJava < SuperClass
    @instance_variable
    @@class_variable
    def initialize()
      puts "I am the constructor"
    end
    def method(param, optional_param="default")
      "This String is returned, even without return
statement"
    end
  end
end
end
SomeModule::CONSTANT
```



# Meaningful Method Names

```
# usage: some_object.valid?class Person
class Person
  def name; @name; end
  def name=(new_name)
    @name = new_name
  end
  def valid?
    return true unless name && name.empty?
    false
  end
end
end
class Person # almost the same
  attr_accessor :name
  def valid?; name && ! name.empty?; end
end
p = Person.new
person.valid?
person.name = "Benjamin"
person.valid?
```



# Only one aunt may die

```
# single inheritance only
```

```
# alternative model: mixins
```

```
class MyArray
```

```
  include Enumerable
```

```
  def each
```

```
    # your iterator here
```

```
  end
```

```
end
```

```
# Enumerable provides:
```

```
all? any? collect detect each_with_index entries find  
find_all grep group_by include? index_by inject map max  
member? min partition reject select sort sort_by sum to_a  
to_set zip
```



# Ruby is flexible

```
class Numeric
  def plus(x)
    self.+(x)
  end
end
```

```
y = 5.plus 6
# y is now equal to 11
```



# Redefining Methods

```
warn("Don't try this at home or at all")
```

```
class Fixnum
  def +( other )
    self - other
  end
end
```

```
5 + 3
```

```
# => 2
```



# Introspection

```
# Tell me what you are
```

```
5.class
```

```
"qwerty".class
```

```
# Tell me what you do
```

```
[1,2,3].methods
```

```
# this is how I got the list above
```

```
Enumerable.instance_methods.sort.join(" ")
```

```
# Would you? Please...
```

```
some_variable.respond_to? :each
```



# Missing Methods

```
# id is the name of the method called, the * syntax
  collects
# all the arguments in an array named 'arguments'
def method_missing( id, *arguments )
  puts "Method #{id} was called, but not found. It has
    " +
      "these arguments: #{arguments.join(", ")}"
end

__ :a, :b, 10
# => Method __ was called, but not found. It has these
# arguments: a, b, 10
```

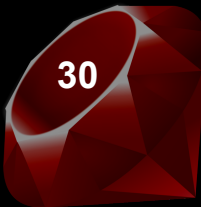


# eval is evil...

```
class Klass
  def initialize
    @secret = 99
  end
end

k = Klass.new
k.instance_eval { @secret } # => 99

# generally:
eval("some arbitrary ruby code")
```



# ...but may be really helpful

```
eval(string [, binding [, filename [,lineno]]) => obj  
mod.class_eval(string [, filename [, lineno]]) => obj
```

```
# there are many blocks like this in the RTM source  
code
```

```
module_eval(<<<-EOS,  
  "(__PSEUDO_FILENAME_FOR_TRACES__)", 1)  
  def #{method_name_variable}  
    some custom method using #{more} #{variables}  
  end  
EOS
```



# Something (not so) exceptional

```
begin
  # some code
  # may raise an exception
rescue ExceptionType => ex
  # exception handler
else
  # no exception was raised
ensure
  # this stuff is done for sure
end

raise Exception, "Message"
```



# Things we do not cover here

alias

here docs

regex details

access restriction

predefined variables: \$!, \$@, \$: ...

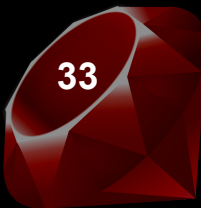
backslash constructions: \n \t

and some other

stuff we do

not need

today



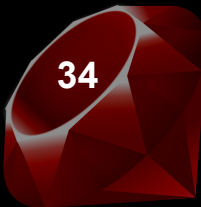
# Sources

<http://www.ruby-lang.org/en/about/>

<http://www.ruby-lang.org/en/documentation/ruby-from-other-languages/>

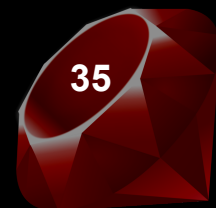
<http://www.zenspider.com/Languages/Ruby/QuickRef.html>

<http://www.ruby-doc.org/core/>



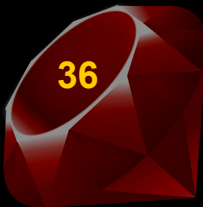
# Break

```
require 'timeout'  
begin  
  status = Timeout::timeout(30.minutes) do  
    ingestion = Thread.start do  
      loop { get_coffee; drink; sleep(30); }  
    end  
    chat = Thread.start do  
      loop { sleep(15); talk; listen; smile; }  
    end  
  end  
rescue Timeout::Error  
  puts "Let's get back to work"  
end
```



# Demo session

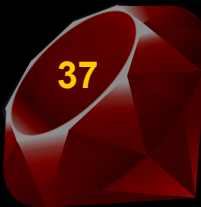
# Seeing is believing... :-)



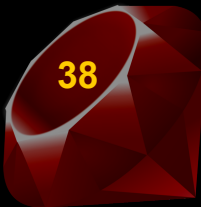
```
# loading the Ruby Topic Maps library
# in a script or in environment.rb
require 'rtm'

# a new module called RTM will be
# loaded into the default namespace

# To get direct access to constants like
# PSI, a hash of TMDM subject indicators
# use
include RTM
```



```
# Connecting to a back end
# Memory
RTM.connect
# File database, default is tmdm.sqlite3
RTM.connect_sqlite3
# Custom database file
RTM.connect_sqlite3("filename.db")
# Standard MySQL database
RTM.connect_mysql("database_name",
    "user_name", "password", "host")
```



```
# At first time use, we have to generate  
all the tables in the database
```

```
RTM.generate_database
```

```
# for now, we use memory database
```

```
RTM.connect
```

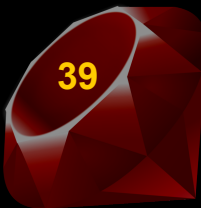
```
# or
```

```
RTM.connect_memory
```

```
# enable SQL statement logging
```

```
RTM.log
```

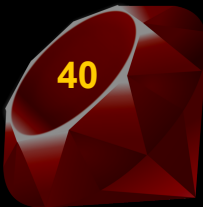
```
# defaults to STDOUT, configurable as Logger
```



# Ruby on Rails

# Get on the Train!

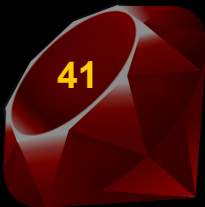
# Can I see your ticket, please?



# Hands-on

# Now, it's your turn.. :-P

(again)



# Fin

```
begin
```

```
  puts "Questions?"
```

```
  x = gets
```

```
  puts response_to(x)
```

```
end until x =~ /(no|exit|quit)/i
```

```
puts "Thank you very much?"
```

