

# Ruby: What's in it for you?

Venkat Subramaniam

venkats@agiledeveloper.com

<http://www.agiledeveloper.com/download.aspx>

## Abstract

Abstract Object-oriented scripting languages, or agile dynamic languages, as some like to call those, are gaining programmers' attention. Ruby is gaining popularity and acceptance due to its expressive power. The language is light and simple. The dynamic nature allows you to express some constructs that are generally harder in so-called strongly typed languages (C++, Java, C#) must be easier in Ruby. In fact, Microsoft is actively working on some of the Ruby features into future versions of C#, and is keen on making the CLR support dynamic typing. Learning Ruby is critical even if you don't intend to use it directly. It helps you stay ahead. We will take an example-driven approach to look at interesting features and strengths of Ruby, and also discuss some of the weaknesses as well.

About the Speaker Dr. Venkat Subramaniam, founder of Agile Developer, Inc., has trained and mentored thousands of software developers in the US, Canada, Europe, and India. He has significant experience in architecture, design, and development of software applications. Venkat helps his clients effectively apply and succeed with agile practices on their software projects, and speaks frequently at conferences.

He is also an adjunct faculty at the University of Houston (where he received the 2004 CS department teaching excellence award) and teaches the professional software developer series at Rice University School of continuing studies.

Venkat has been a frequent speaker at No Fluff Just Stuff Software Symposium since Summer 2002.



## What's Ruby

- Power of dynamic language
  - Dynamic, agile, OO
  - Derives strengths of Perl, Smalltalk, Python
  - Elegant
- Alternative to Java/C++/C# for small/medium size projects
- Serves to write small additional tasks on a project
- Great to write unit tests with
- Can take care of all those "side" projects/tasks you need to get done

## History

- Created by Yukihiro Matsumoto (Matz) in 1993
- Slowly being accepted in the west
- Popularized by some key players in the industry
- Gaining strength in various applications
  - Ruby-On-Rails (ROR)

## Tools

- Ruby interpreter

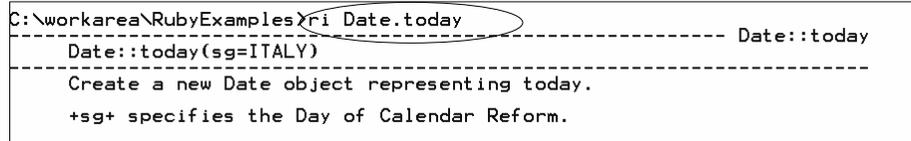


```
C:\WINDOWS\system32\cmd.exe
C:\workarea\RubyExamples>ruby Test.rb
Hello
```

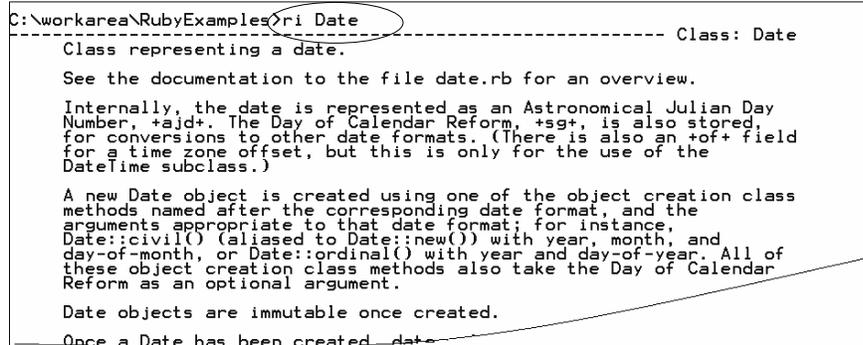
- Comes with FreeRIDE IDE
- Mondrian IDE
- Command line and Notepad(2) works great as well!
  - The following blog entry shows you how
  - <http://tinyurl.com/as8z7>

## Documentation

- ri



```
C:\workarea\RubyExamples>ri Date.today
----- Date::today
Date::today(sg=ITALY)
-----
Create a new Date object representing today.
+sg+ specifies the Day of Calendar Reform.
```



```
C:\workarea\RubyExamples>ri Date
----- Class: Date
Class representing a date.
See the documentation to the file date.rb for an overview.
Internally, the date is represented as an Astronomical Julian Day
Number, +ajdt+. The Day of Calendar Reform, +sg+, is also stored,
for conversions to other date formats. (There is also an +of+ field
for a time zone offset, but this is only for the use of the
DateTime subclass.)
A new Date object is created using one of the object creation class
methods named after the corresponding date format, and the
arguments appropriate to that date format; for instance,
Date::civil() (aliased to Date::new()) with year, month, and
day-of-month, or Date::ordinal() with year and day-of-year. All of
these object creation class methods also take the Day of Calendar
Reform as an optional argument.
Date objects are immutable once created.
-----
Once a Date has been created, date
```

- RDoc at <http://www.ruby-doc.org>

## Less Clutter

- Code less, do more
- Java:

```
public class Hello
{
    public static void main(String[] args)
    {
        System.out.println("Hello");
    }
}
```

GOJ

- Ruby

```
puts 'Hello'
```

## Variables

- Dynamic typing

```
x = 1
puts x

x = Date.today()

puts x

x = 'test'
puts x
```

- Output: 

```
1
2006-02-02
test
```

# Types

- Everything is an object

```
puts -1.abs
puts 'test'.capitalize
puts 'HELLO'.downcase
puts 'HELLO'.downcase() # () is optional
puts ('No need for ()')
```

```
1
Test
hello
hello
No need for ()
```

```
puts 1.class

puts 'We start with Bignum and end with Fixnum'
val = 123456789000
6.times do
  puts val.class
  val /= 10
end
```

```
Fixnum
We start with Bignum and end with Fixnum
Bignum
Bignum
Bignum
Fixnum
Fixnum
Fixnum
```

# Global and Predefined Variables

- This is hideous
- You may be setting some global variable and not know it (familiarize yourself)
- Some predefs: \$\_, \$0, \$\$, \$&, \$!, ...

```
str = gets.chomp
puts str

# or you can do
gets.chomp
puts $_

puts $0
```

```
input1
input1
input2
input2
```

C:/workarea/RubyExamples/GlobalPredefinedVariables/Predefined.rb

```
require 'English'
# Provides descriptive names for
# cryptic predefined variables

puts $0
puts $PROGRAM_NAME
```

## Conventions

- Lower case start or underscore
  - Method, parameters, local variables
- Upper case start
  - Class name, module name
- Multiword
  - Instance variables use underscore
  - Class name uses mixed case
- Method names ending with
  - ! Imply sideeffect (modify object)
  - ? Return boolean
  - = set value (can appear as l-value)
- nil is an object that represents nothing
  - nil is false when used in comparison
- \$ - global
- @ - instance variable
- @@ - class variable

## "" vs " (Double vs. Single Quotes)

- " is a simple string
- Expressions within "" are evaluated

```
today = Date.today
puts '#{today}'
puts ""#{today}""
```

```
#{today}
2006-02-02
```

- Better to use ' if you don't have expressions

## More on Strings

```
str1 = ['This', 'is', 'an', 'array', 'of', 'strings']
str2 = %w{and this one is array too}

puts str1.length
puts str2.length

str3 = %Q/You can write a string "like" this/
str4 = %q~or like this one. The char next to %q is the delimiter~

puts str3
puts str4
puts %q[This too]
puts %q(and this)
puts %q!It is #{Time.now}!
puts %Q!It is #{Time.now}!

str5 = <<REALLY_DONE
whatever string you
want to write, keep writing until you
are REALLY_DONE.
REALLY_DONE
#Must be on first column. Use <<-REALLY_DONE to indent this

puts str5
```

```
5
6
You can write a string "like" this
or like this one. The char next to %q is the delimiter
This too
and this
It is #{Time.now}
It is Thu Feb 02 08:40:46 Central Standard Time 2006
whatever string you
want to write, keep writing until you
are REALLY_DONE.
```

## Writing Class

```
class Car
# default is public
attr_reader :year, :miles, :color
attr_writer :color

  def initialize(year)
    @year = year
    @color = "Red"
    @miles = 0
    @fuel_level = 0
  end

  def fuel=(level)
    @fuel_level = level
  end

  def fuelLevel
    @fuel_level
  end

private
  #...
protected
  #...
public
  def drive()
    @miles += 1
  end

  #...
end
```

```
c1 = Car.new(2006)
puts "Car made in #{c1.year}"
puts "Miles: #{c1.miles}"
c1.drive()
puts "Miles: #{c1.miles}"
puts "Car color: #{c1.color}"
c1.color = "Black"
puts "Car color: #{c1.color}"
puts "Fuel level: #{c1.fuelLevel}"
c1.fuel = 0.5
puts "Fuel level: #{c1.fuelLevel}"
```

```
Car made in 2006
Miles: 0
Miles: 1
Car color: Red
Car color: Black
Fuel level: 0
Fuel level: 0.5
```

# Array

```
val = [1, 'test', 2.3]
for aVal in val
  puts aVal
end
puts val[2]
puts val[5]
puts val[5] = 'ha'
print 'Now the list: '
puts val.join(',')
```

```
1
test
2.3
nil
ha
Now the list: 1, test, 2.3, , , ha
```

# Range

```
str = 'hello'

puts str[1..4]
puts str[0..-4]
puts str[0..-1]
puts str[2..-2]
puts str[2..-4]

puts str
```

```
ello
hello
hello
llo
hello
```

# Hashes

```
langs = {'C++' => 'Stroustrup', 'Java' => 'Goslin',  
        'C#' => 'Hejlsberg', 'Ruby' => 'Matz'}  
  
puts langs['Ruby']  
  
for key, value in langs  
  puts key + ":" + value  
end  
  
print langs[blah] = '  
puts langs[blah]'  
  
hash = Hash.new(0)  
hash['test'] = 'always'  
  
puts "hash[test] = #{hash['test']}"  
print 'hash[blah] = '  
puts hash[blah]
```

```
Matz  
C#:Hejlsberg  
C++:Stroustrup  
Ruby:Matz  
Java:Goslin  
langs[blah] = nil  
hash[test] = always  
hash[blah] = 0
```

# if...

```
val = 5  
  
if val > 4  
  puts 'hello val is > 4'  
end  
  
puts 'haha' if val > 4
```

```
hello val is > 4  
haha
```

## Writing a function

- def a function just like you would in Java
- No need to return, the value of last statement is returned

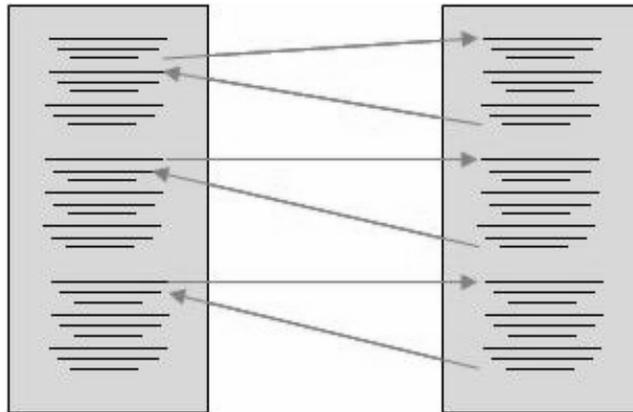
```
def countTill(number)
  for i in 1..number
    puts i
  end
end
countTill(10)
```

```
10
9
8
7
6
5
4
3
2
1
```

## Closures

- An anonymous chunk of code you can invoke
  - take arguments
  - return a value
  - reference and use variables declared in its surrounding scope
- Some what like anonymous inner classes in Java, but more powerful and convenient
- How is this different from a method?
- You directly call a method
- Closure is really cool, it is used for the method you call to call back
- Now we are talking about coroutines

## Coroutines



## Coroutines with closures

- I have a method that creates even numbers until a given limit
- What do you want to do with these even numbers?
- That is up to you, it depends on what you want to do at the place of call
- So, my method is going to create even numbers, but will yield to your code so you can do something with it

## Closures

```
def createEven(number)
  for i in 2..number
    if i % 2 == 0
      yield i
    end
  end
end

#You can simplify above code
#using step method. But, that
#will use what we're trying to
#learn here!

createEven(10) { |val| puts val }

print 'Total of even numbers from 1 to 10 is: '
total = 0
createEven(10) { |i| total += i }
puts total
```



*(lots of syntax sugar)*

```
2
1
0
Total of even numbers from 1 to 10 is: 30
```

## Closures As Parameter

```
class Pump
  def initialize(&calc)
    @myCalc = calc
  end

  def compute(val)
    @myCalc.call(val)
  end
end

p1 = Pump.new() { |v| puts "Using calc1 to work on #{v}" }
p1.compute(3)
p1.compute(4)

p2 = Pump.new() { |i| puts "I got #{i}" }
p2.compute(3)
p2.compute(4)
```

```
Using calc1 to work on 3
Using calc1 to work on 4
I got 3
I got 4
```

## Closures and Collections

```
lst = [1, 2, 3, 1, 8]

print 'Elements in the list are: '
lst.each {|val| print "#{val} " }
puts

puts 'Each element incremented by one:'
puts lst.collect {|it| it + 1 }

print 'Look for val > 3: '
puts lst.find {|it| it > 2 }

puts 'Look for all val > 3: '
puts lst.find_all {|it| it > 2 }

print 'Total of all elements is: '
puts lst.inject(0) {|carryover, it| carryover + it }

print 'Total of all elements plus 5 is: '
puts lst.inject(5) {|carryover, it| carryover + it }

print 'All elements joined by ~: '
puts lst.join('~')
```

```
Elements in the list are: 1 2 3 1 8
Each element incremented by one:
2 3 4 5 9
Look for val > 3: 3
Look for all val > 3:
[3, 8]
Total of all elements is: 15
Total of all elements plus 5 is: 20
All elements joined by ~: 1~2~3~1~8
```

## Accessing File

```
File.open('FileAccess.rb', 'r') do |file|
  file.each_line { |line| puts line }
end
```

```
File.open('FileAccess.rb', 'r') do |file|
  file.each_line { |line| puts line }
end
```

## Adding Methods

```
class String
  def greet
    "Hello " + to_s() + ', pleased to meet you!'
  end
end

bob = "Bob"
puts bob.greet
```

```
Hello Bob, pleased to meet you!
```

## Regex

- Very strong support for Regular Expressions
- =~ compares regular expressions

```
str = "It a rainy day in Houston"
puts str

puts 'Take umbrella' if str =~ /rainy/
```

```
It a rainy day in Houston
Take umbrella
```

# Unit Testing

```
require 'test/unit'

class TestCode < Test::Unit::TestCase
  def test1
    assert(true)
  end

  def test2
    assert_equal(5, 2+3)
  end
end
```

```
Loaded suite C:/workarea/RubyExamples/UnitTesting/Testing
Started
Finished in 0.0 seconds.
2 tests, 2 assertions, 0 failures, 0 errors
```

## Strengths and Weaknesses

- +
  - Convention over configuration
  - Lightweight
  - Very suitable for small to medium size projects
- -
  - Dynamic typing
    - unit testing is imperative
  - Not yet fully accepted by mainstream industry

## References

1. <http://www.rubycentral.com>
2. <http://www.ruby-doc.org>
3. Programming Ruby, Dave Thomas, with Chad Fowler and Andy Hunt  
(<http://www.pragmaticprogrammer.com/titles/ruby/index.html>)
4. Download examples/slides from  
<http://www.agiledeveloper.com/download.aspx>

*Thanks for being a*