Yukihiro Matsumoto

Treating Code As an Essay

Pro seminar **Beautiful Code**

Jan Lelis

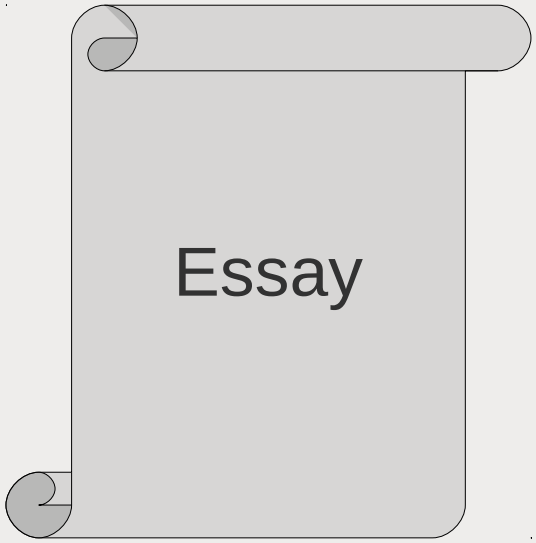
15th of June, 2010

Contents

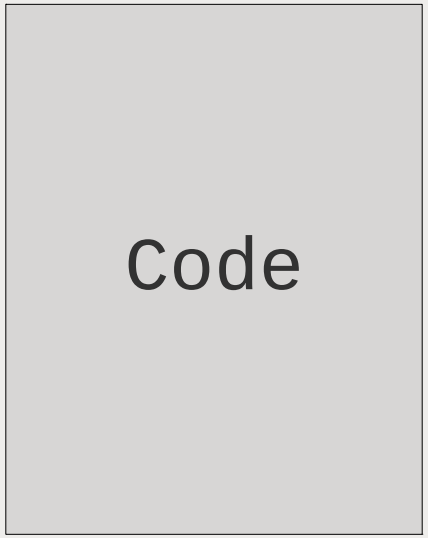
- Introduction (7 slides)
- Ruby: facts and emergence (5 slides)
  - “Beautiful Code” – Ruby syntax and examples
    - Language features (9 slides)
    - Using Ruby as “Domain Specific Language” (3 slides)
    - Example “Quicksort” (2 slides)
- Appendix (2 slides)

Excerpts by Yukihiro Matsumoto are taken from “Beautiful Code” and have this font style.

Treating Code As an Essay?



“What is it about?”



“What does it do?”

Both essays and lines of code are meant - before all else - to be read and understood by human beings.

Hello World

C

```c
#include <stdio.h>

int main(void) {
    printf("Hello, World!");

    return 0;
}
```

Hello World

Java

```
class HelloWorld
{
	public static void main(String args[])
	{
		System.out.print(“Hello World!“);
	}
}
```

Hello World

Perl / Python / Ruby

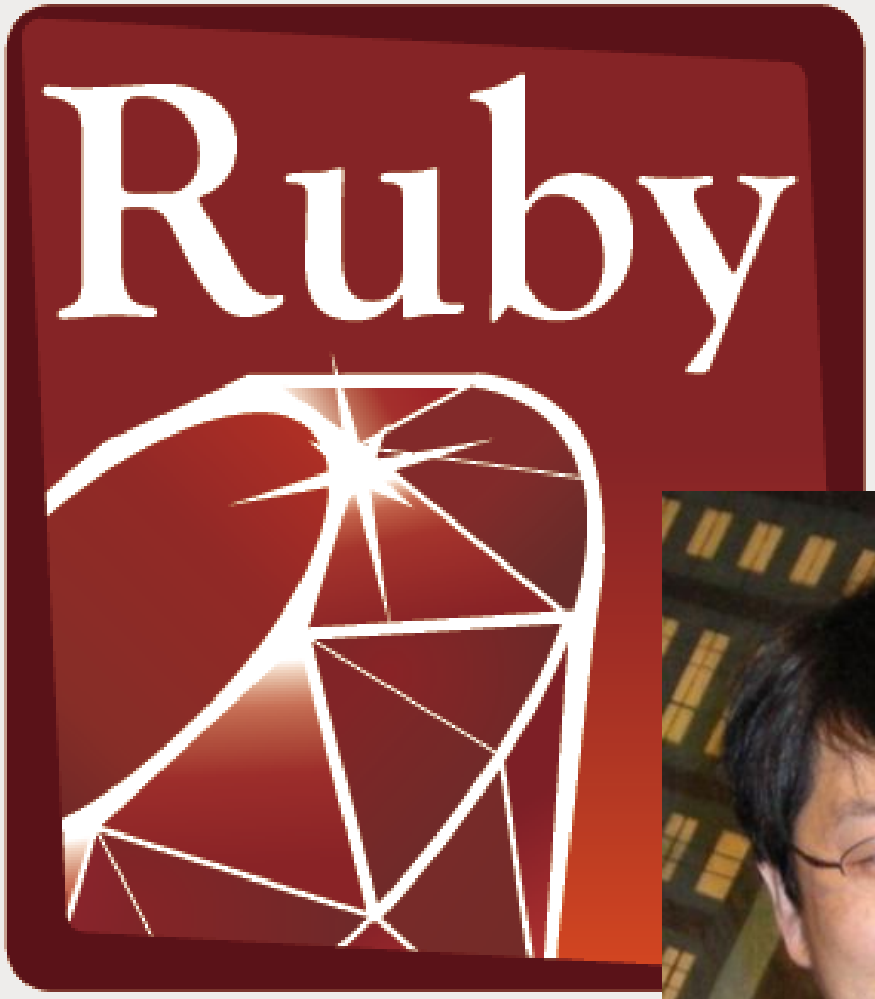
```
print “Hello World!“
```

`#!/usr/bin/perl`

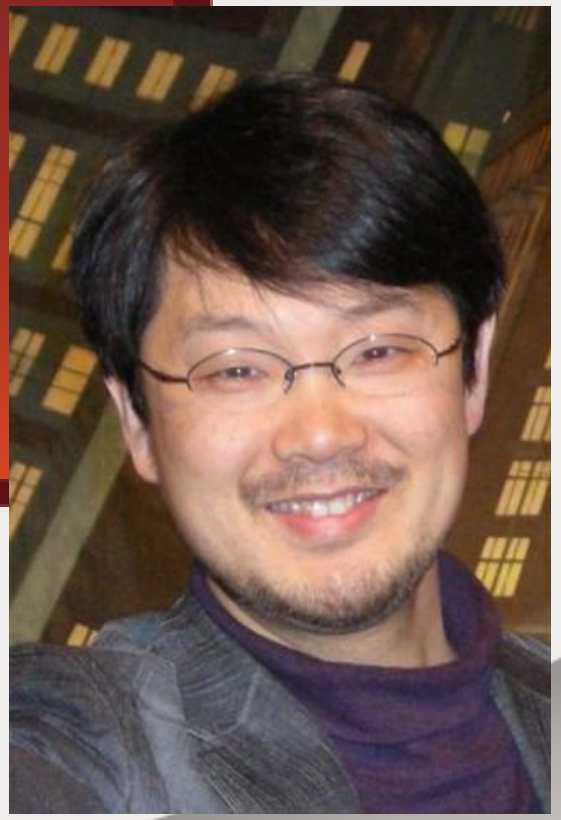
Perl

Most programs are not write-once [...] it is therefore more important by far for humans to be able to understand the program than it is for the computer.

```
                                                                           ;;;;;;
                                                                       ;;;;;;;;;;;
                                                                   ;;;;;;;;;;;;;;;
                                                                ;;;;;;;;;;;;;;;;;
                                                             ;;;;;;;;;;;;;;;;;;;
                                                           ;;;;;;;;;;;;;;;;;;;;
                                                         ;;;;;;;;;;;;;;;;;;;;;
                                     +$I=sub{+s+^+     ;;;;;;;      ;;;;;;;;;
                                  $“x$_[1]+gem;$/x$_#   ;;;;        ;;;;;;;;
                               [0].$_.$/};$W=sub{$~=!q~            ;;;;;;;
                            ~.pop();system($^O=~Win?ClS:#         ;;;;;;;
                          'clear'),print,select$Z,$Z,$Z,!“       ;;;;;;
                         “||$~for@_};$H=sub{+join$/,map($_#     ;;;;;;
                        x$_[0],pop=~m-.+-g),!_};$_=!Mima,s--   ;;;;;
                       “@{['=9+)w'^RINGS]}\%;local@{[Saturn^# ;;;;;
                      wNXIBP]}“-see;s-^#!..+?$/(?=$“+;)--is  ;;;;
                     y-;-'-;s-\w-~-gi;$S=$_;#--Beautiful]  ;;;;
                     @S=m-.+-g;$N=1+.6-!th_,$--=-82-$---  ;;;
                    $_.=$“x-(y---c-$-)for@S;$R=sub{$i#  ;;;  -d
                    =0;join$/,map{$j=$%;join!_,grep#  ;;;  Rhea
                    !($j++%$_[$%]),m-.-g}grep!($i#  ;;;  -Titan
                    ++%$_[0]),@S};$L=join!_,map#  ;;;  -Huygens
                    ~~reverse.$/,@S;@R=(&$I(q-  ;;;  -&&20,051,
                    $_=_^q-q-),&$I(20,41-!q-  ;;;  -,$_=F|K),$
                    I->(15,31,$_=&$R(4-!q-  ;;;  -)),&$I(13-!“
                  ;;“,28,$_=&$R(3)),&${  ;;;  _^_^I}(10,20-!“
                ;;;;;“,$_=$R->(2)),q-  ;;;  -&&$S);@O=map&{“
              ;;;;;;  “&&$H}($_,&${  ;;;  R.!-_}($_))x$_,!“
             ;;;;;     “+2..2*~~2  ;;;  @Y=reverse@R#Dione
           ;;;;;;       &${m--  ;;;  S|A|T|U}(@R,$N)||!q-
         ;;;;;;;          b-  ;;;  &$W(@O[0,1,2,1,0!=!q-
        ;;;;;;;            ;;;;  -],!1!~~1);&$W($S.!q-
      ;;;;;;;;;        ;;;;;  -,$L,0.16)for$%..5+!q-
     ;;;;;;;;;;    ;;;;;;;;;    Cassini-;&{$W||q-
    ;;;;;;;;;;;;;;;;;;;;;;         -}(@Y,1.6)
   ;;;;;;;;;;;;;;;;;;;;
  ;;;;;;;;;;;;;;;;;;
 ;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;
;;;;;;;;;;;
  ;;;;;;
```



Beautiful code is really meant to help the programmer be happy and productive.



Yukihiro Matsumoto

Ruby

- Ruby is
  - 2 parts Perl
  - 1 part Python
  - 1 part Smalltalk
- Connects good elements from different languages with a beautiful syntax
- Big standard library
- Lots of extensions (“gems”)

Emergence

- 1995: First version 0.95
  - Long period only with Japanese documentation
- 1999: ruby-talk
  - English mailing list
- 2000: “Programming Ruby” (Pickaxe Book)
  - Book written by the “Pragmatic Programmers”
  - Has been the first English documentation
- 2005:

screenshot of rubyonrails.org

- Rails: A web framework written in Ruby
  - Influenced lots of web frameworks of other languages
  - Uses MVC architecture (Model, View, Controller)
  - Joins “best practices” with a web-DSL

> In order to eliminate redundancy, we follow the DRY principle: Don't Repeat Yourself. If the same code exists in multiple places, whatever you're trying to say becomes obscured.

Ruby Interpreter

- Current version: 1.8.7 / 1.9.1
- Implementations
  - **MRI**: Matz Ruby Interpreter, Reference
  - **JRuby**: on the JVM
  - **IronRuby**: on .NET
  - **MacRuby**: Cocoa, LLVM
  - **Rubinius**: in Ruby with a little C++ core
  - (**Cardinal**: on the Perl6-VM “Parrot”)

A simple program

Human beings are more conservative than you might think, [so Ruby] is an extremely conservative programming language. Ruby sticks to traditional control structures programmers are familiar with, such as `if`, `while`, etc.

```
def fib(n)
  if n == 0 || n == 1
    return 1
  else
    return fib(n-1) + fib(n-2)
  end
end
```

Focus on the gist

- Dynamic
  - No type declarations
- No brackets for simple method calls
- Last statement of a method is automatically the return value, no `return` needed

> *Brevity* is one element that helps make code beautiful. [...] Because there is a definite cost involved in scanning code with the human eye, programs should ideally contain no unnecessary information.

```
def fib(n)
  return 1 if n <= 1
  fib(n-1) + fib(n-2)
end
```

Data structures (selection)

- Everything is an object, even simple data types

data type	literal	description
Integer	`1`	
Range	`1..5`	
Float	`1.0`	floating-point representation
String	`'a'`	
Symbol	`:a`	string for identifying purpose
Regexp	`/a/`	regular expression
Array	`[1,'a']`	list of objects, fixed order
Hash	`{:a => 1,}`	Collection of key => value pairs

Data structures: strings & symbols

```
a = 'Hello'           #=> Hello
a << ' University'    #=> Hello University
a.delete! 'o'         #=> Hell University
a.empty?              #=> false
a[5,3]                #=> Uni
```

```
'Beautiful Code' =~ /Spaghetti/ #=> nil
'Beautiful Code' =~ /Beaut.*/   #=> 0
```

```
'hello'.size #=> 5
:hello.size  # NoMethodError
```

Data structures: arrays & hashs

```
a = []                  #=> []
b = [11,4,7,10]         #=> [11, 4, 7, 10]
a << 2 << 4 << 6        #=> [2, 4, 6]
a + b                   #=> [2, 4, 6, 11, 4, 7, 10]
a & b                   #=> [4]
b.sort                  #=> [4, 7, 10, 11]
```

```
a = {:a_key => 'is here', :another => 3}

a[:a_key] = 'is not here'
a[6] = 15

puts a  # outputs:
        #  {6=>15, :a_key=>“is not here“,
        #  :another=>3}
```

Anonymous functions

- Methods can take “blocks” when called

  (Anonymous functions / Closures / Lambdas / Procs)
- Blocks are nicely integrated into the syntax
  - Build with do and end or { and }
  - High significance regarding programming style

```
a = [1,2,3]
i = 0
while i < a.size
  # do something with a[i]
  i += 1
end
```

```
a = [1,2,3]
a.each do |e|
  # do something with e
end
```

Functional Style

- Meaningful syntax because of the Closures

```
5.times{
  puts “Hello World!“
}
```

- Useful methods for object collections (“Enumerables”) like arrays

```
a = [1,2,3,4].map{ |ele| ele*ele } #=> [1, 4, 9, 16]
```

```
r = 36..42
r.member? 13                    #=> false
r.max                           #=> 42
r.select{ |ele| ele%3 == 0 }    #=> [36, 39, 42]
```

Variable names

- Constants- and class names are capitalized
- Method- and variable names begin with a lowercase letter
- Instance variables of an object and class variables are prefixed with an @
- Global variables are prefixed with $

Classical object system

```ruby
class Example
  def initialize(input = '', params = {})
    @data, @options = input, params
  end

  def options
    @options
  end

  def options=(value)
    @options = value
  end
end
```

```ruby
a = Example.new 'my_data', :user => 'Yukihiro'

puts a.options[:user] # Yukihiro
a.options[:user] = 'matz.'
```

Meta programming

- All classes are “open”
  - Existing classes expandable
- `eval`
  - Generate and execute code at runtime
- Reflection

```
5.is_a? Integer  #=> true
```

- `method_missing`
  - Is executed, when the method name is unknown

When information is duplicated, the cost of maintaining consistency can be quite high.

Ruby for simple intern DSLs

- Domain Specific Languages (DSL)
  - Formal language for specific problem areas
  - Goal: ease of use
- Distinction in
  - *Extern:* new language
  - *Intern:* using an existing language
- Tools of Ruby enable creation of well understandable and readable code
  - Method calls look like keywords

DSL: Rake (“Ruby Make”)

- Written in Ruby (instead of extra language)

```ruby
task :default => [:test]
task :test do
  ruby “test/unittest.rb“
end
```

```ruby
task({:default => [:test]})
task(:test, &lambda(){
  ruby “test/unittest.rb“
})
```

- Both variants correct
  - First considerably better readable

We often feel beauty in simple code. [...] when programs are obscure rather than comprehensible, the results are bugs, mistakes, and confusion.

DSL: “Ruby on Rails”

- Model definitions

```ruby
class Entry < ActiveRecord::Base
  has_many :comments
  has_and_belongs_to_many :tags
end
```

```ruby
class Comment < ActiveRecord::Base
  belongs_to :entry
  validates_presence_of :title
end
```

```ruby
a = Entry.find_by_id 42
b = a.comments.find_by_title 'hi'
b.title = ''
b.save # fails..
```

Live Coding

- Quicksort

> [Another] important element in the concept of "beautiful code" is flexibility [which I define] as *freedom from enforcement from tools*.

Live Coding

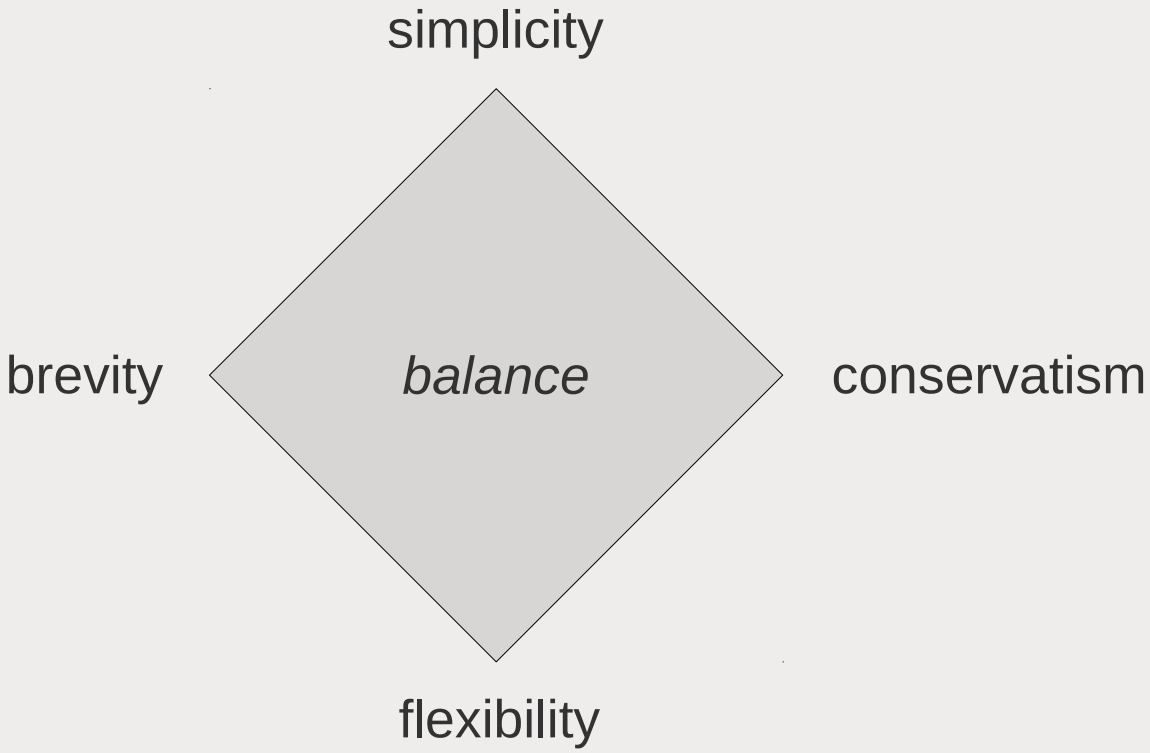
- Quicksort

```
class Array
  def qsort
    return self if self.size <= 1

    left, right = [], []
    pivot = self.shift
    self.each do |ele|
      ele <= pivot ? left << ele : right << ele
    end

    left.qsort + [pivot] + right.qsort
  end
end
```

So, what is “Beautiful Code”?



And if you also make sure to have fun writing and reading code, you will experience happiness as a programmer.

Happy Hacking!

Sources, Resources

- Oram, Wilson: “Beautiful Code” (O'Reilly, 2007)
- Websites
  - ruby-lang.org (official page)
  - rbjl.net (my Ruby blog)
- Perl Saturn by *eyepopslikeamosquito*
  - http://www.perlmonks.org/?node_id=397958
- Pictures
  - Yukihiro Matsumoto: public domain, Wikimedia
  - Ruby Logo: © Ruby Association LLC
- Impress template: hagetaka0