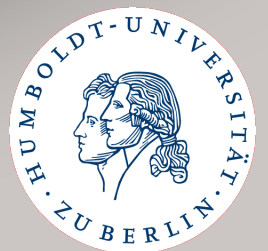


Yukihiro Matsumoto

Treating Code As an Essay

Proseminar **Beautiful Code**

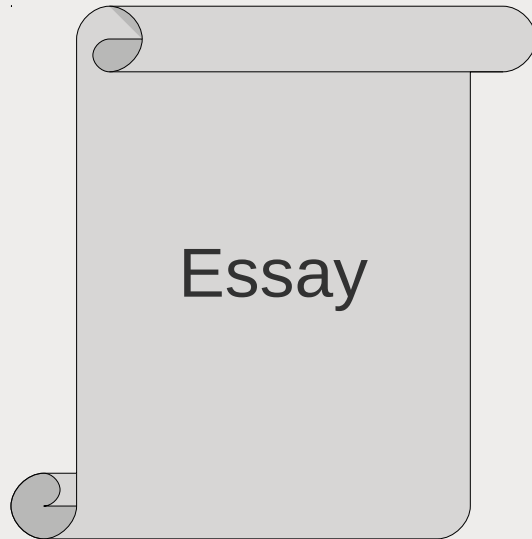
Jan Leis
15. Juni 2010



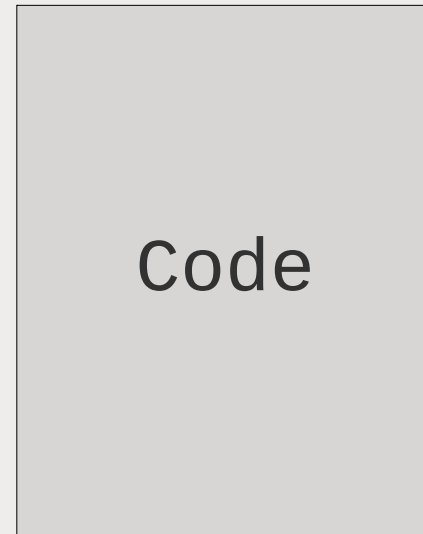
Inhalt

- Einführung (7 Folien)
- Ruby: Fakten und Entstehung (5 Folien)
- „Beautiful Code“ – Ruby Syntax und Beispiele
 - Sprachfeatures (9 Folien)
 - Verwendung als „Domain Specific Language“ (3 Folien)
 - Beispiel „Quicksort“ (2 Folien)
- Anhang (2 Folien) Zitate von Yukihiro Matsumoto aus „Beautiful Code“ sind in diesem Schriftstil

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

```
#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

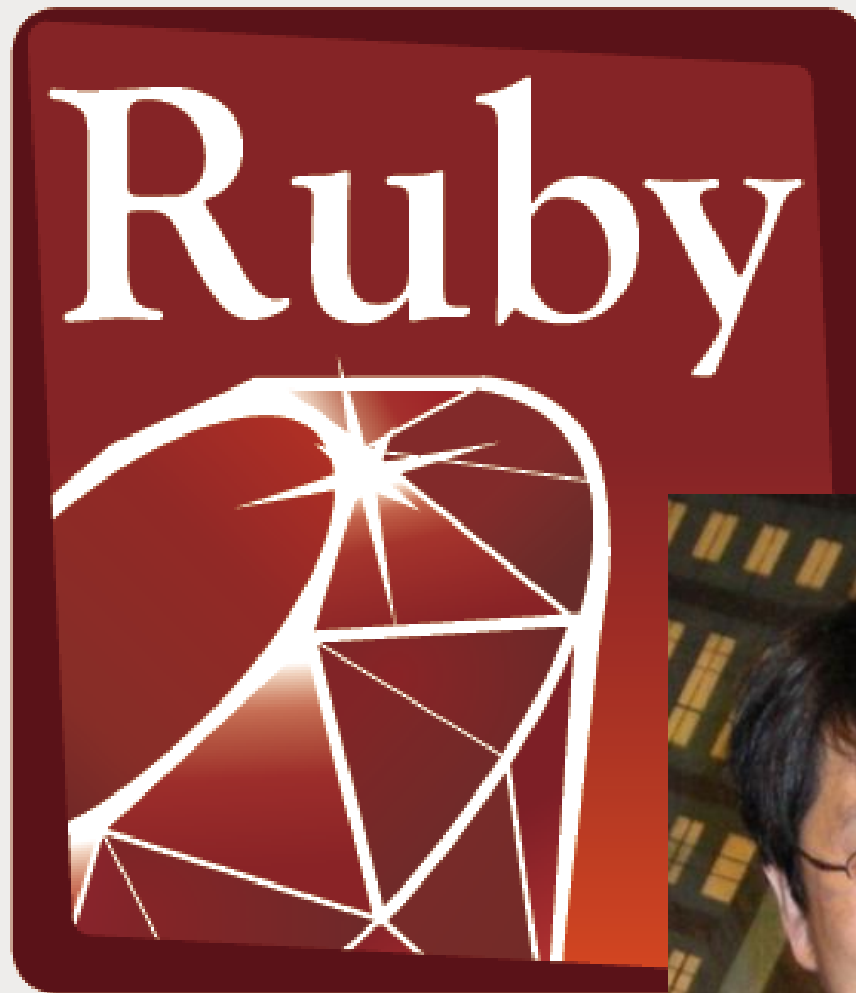
```
print "Hello world!"
```

```
#!/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{$i#
  =0;join$/,map{$j=$%;join!_,grep#
  !($j++%$_[$%]),m-.g}grep!($i#
  ++%$_[0]),@S};$L=join!_,map#
  --reverse.$/,@S;@R=(&$I(q-
  $__^q-q-),&$I(20,41-!q-
  I->(15,31,$_=&$R(4-!q-
  ;;",28,$_=&$R(3)),&${
  ;;";,$_=$R->(2),q-
  ;;";" &&$H}($_,&${
  ;;"; @Y=reverse@R#Dione
  &${m--
  b-
  ;;"; &$W(@0[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 ist
 - 2 Teile Perl
 - 1 Teil Python
 - 1 Teil Smalltalk
- Verbindet gute Elemente aus verschiedene Sprachen mit schöner Syntax
- Große Standard-Library
- Viele Erweiterungen („gems“)

Entstehung

- 1995: Erste Version 0.95
 - Lange Zeit nur mit japanischer Dokumentation
- 1999: ruby-talk
 - Englischsprachige Maillingliste
- 2000: „Programming Ruby“ (Pickaxe Book)
 - Buch von den „Pragmatic Programmers“
 - War die erste englische Dokumentation
- 2005:



Web development that doesn't hurt

Ruby on Rails® is an open-source web framework that's optimized for programmer happiness and sustainable productivity. It lets you write beautiful code by favoring convention over configuration.

Screenshot von rubyonrails.org

- Rails: In Ruby geschriebenes Webframework
 - Beinflusste viele Webframeworks anderer Sprachen
 - Nutzt MVC Architektur (Model, View, Controller)
 - Verbindet „Best Practises“ mit einer 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

- Derzeitige Version: 1.8.7 / 1.9.1
- Implementationen
 - **MRI**: Matz Ruby Interpreter, Referenz
 - **JRuby**: auf der JVM
 - **IronRuby**: auf .NET
 - **MacRuby**: Cocoa, LLVM
 - **Rubinius**: in Ruby mit kleinem C++ Kern
 - (**Cardinal**: auf der Perl6-VM „Parrot“)

Ein einfaches Programm

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
```

Konzentration auf das Wesentliche

- Dynamisch
 - Keine Typdeklarationen
- Keine Klammern bei einfachen Methodenaufrufen
- Letzte Statement einer Methode ist automatisch der Rückgabewert, kein return notwendig

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
```

Datenstrukturen (Auswahl)

- Alles sind Objekte, auch simple Datentypen

Datentyp	Literal	Beschreibung
Integer	<code>1</code>	Ganzzahl
Range	<code>1..5</code>	Bereich
Float	<code>1.0</code>	Fließkommazahl
String	<code>'a'</code>	Zeichenkette
Symbol	<code>:a</code>	Zeichenkette für Identifizierungszwecke
Regexp	<code>/a/</code>	Regulärer Ausdruck
Array	<code>[1, 'a']</code>	Liste von Objekten, feste Reihenfolge
Hash	<code>{:a => 1, }</code>	Sammlung von Schlüssel => Wert Paaren

Datenstrukturen: Strings & Symbole

```
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
```

Datenstrukturen: 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}
```

Anonyme Funktionen

- Methoden nehmen beim Aufruf „Blöcke“ an
(Anonyme Funktionen / Closures / Lambdas / Procs)
- Blöcke sind gut in Syntax integriert
 - Werden mit `do` und `end` oder `{` und `}` gebildet
 - Hoher Stellenwert im Programmierstil

```
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
```

Funktionaler Stil

- Aussagekräftige Syntax aufgrund der Closures

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

- Nützliche Methoden für Objektsammlungen („Enumerables“) wie 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]
```

Variablennamen

- Konstanten- und Klassennamen werden groß geschrieben
- Methoden und lokale Variablen fangen mit Kleinbuchstaben an
- Instanzvariablen eines Objektes und Klassenvariablen beginnen mit @
- Globale Variablen starten mit \$

Klassisches Objektsystem

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

  def options
    @options
  end

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

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

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

Metaprogrammierung

- Alle Klassen sind „offen“
 - Bestehene Klassen erweiterbar
- `eval`
 - Code zur Laufzeit generieren und ausführen
- Reflektion
 - ```
5.is_a? Integer #=> true
```
- `method_missing`
  - Wird ausgeführt, wenn keine Methodename unbekannt

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

# Ruby für einfache interne DSLs

- Domain Specific Languages (DSL)
  - Formale Sprache für bestimmtes Problemfeld
  - Ziel: Einfache Bedienbarkeit
- Unterscheidung in
  - *Extern*: Neue Sprache
  - *Intern*: Nutzung einer bestehenden Sprache
- Mittel von Ruby erlauben gut verständlichen und einfach lesbaren Code
  - Methodenaufrufe wirken wie Schlüsselwörter

# DSL: Rake („Ruby Make“)

- In Ruby geschrieben (anstatt in Extra-Sprache)

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

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

- Beide Varianten korrekt
  - Erste wesentlich besser lesbar

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-Definitionen

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

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

```
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

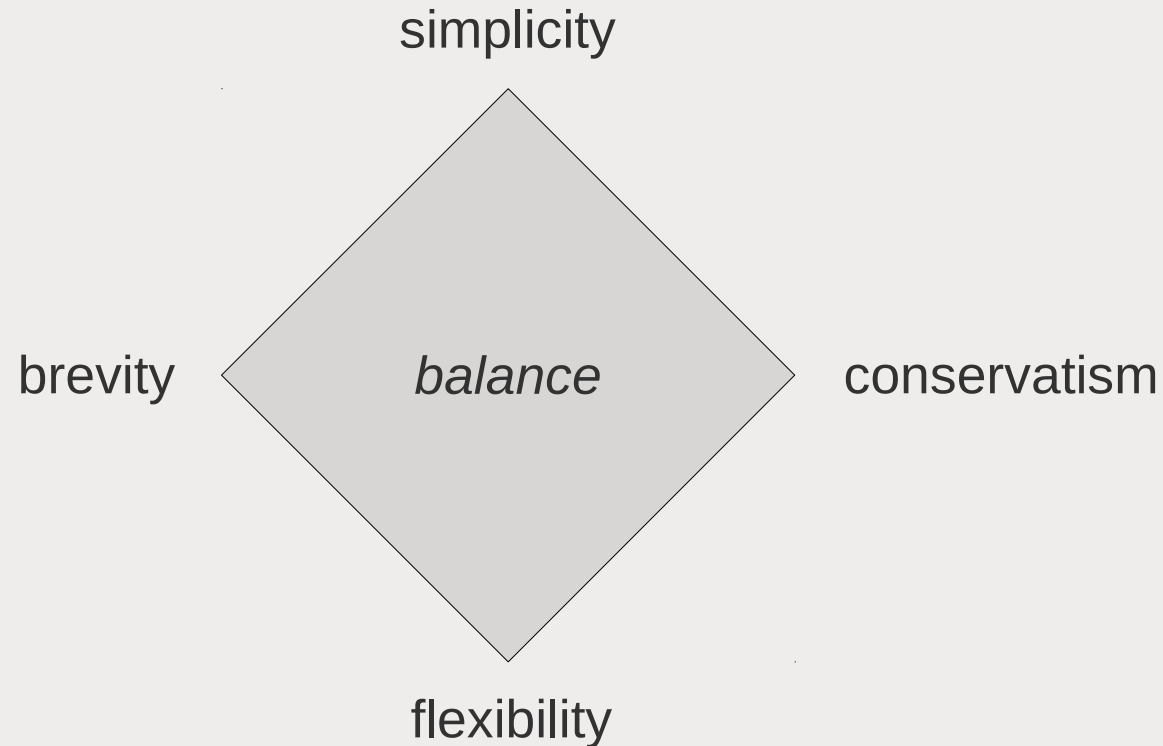
- Quicksort

```
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!

# Quellen, Ressourcen

- Oram, Wilson: „Beautiful Code“ (O'Reilly, 2007)
- Websites
  - [ruby-lang.org](http://ruby-lang.org) (Offizielle Seite)
  - [rbjl.net](http://rbjl.net) (Mein Ruby Blog)
- Perl-Saturn von *eyepopslikeamosquito*
  - [http://www.perlmonks.org/?node\\_id=397958](http://www.perlmonks.org/?node_id=397958)
- Bilder
  - Yukihiro Matsumoto: public domain, Wikimedia
  - Ruby Logo: © Ruby Association LLC
- Impress Template: [hagetaka0](#)