Ruby

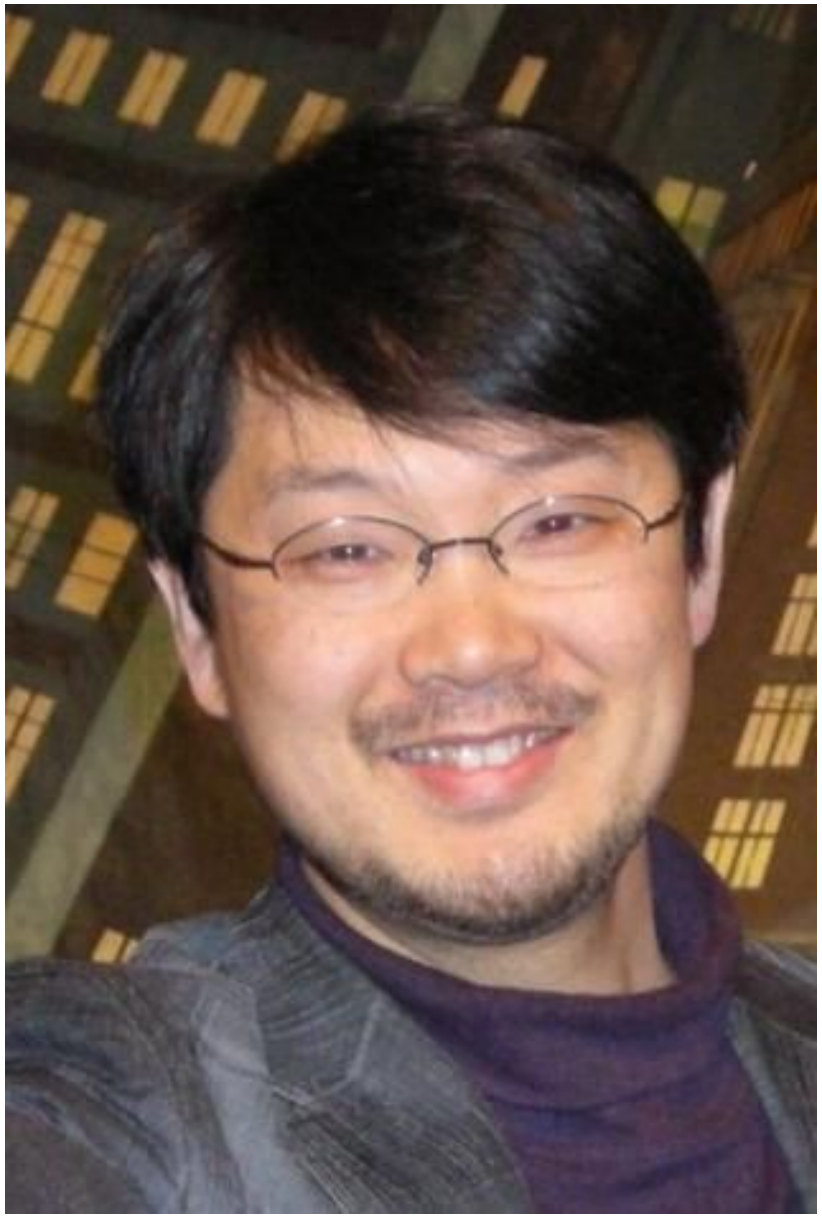
Ein kurzer, nicht vollständiger Überblick
zusammengestellt von Christian Glanz
Mai 2007

Themen

- Yukihiro „Matz“ Matsumoto
- Hello, world!
- ri
- irb
- Input/Output
- Variablen
- Methoden
  - Rückgabewert von Methoden
  - Methodenrückgabewerte für Vergleiche
  - Methodennamen
  - Methoden aufrufen, Punkt & Doppelpunkt
  - Methoden schützen
  - Aliase
  - Methoden dynamisch aufrufen
- Klassen
  - Instanzmethoden & -variablen
  - Klassenmethoden & -variablen
  - Getter & Setter
- Objekte erweitern
- Abstrakte Klassen
- Vererbung
- Module
  - Module & Mix-Ins
- Symbole
- Blöcke
- Iteratoren
  - über Bereiche & Arrays iterieren
- Exceptions
- Reguläre Ausdrücke
- Dateien
- Objekte speichern & laden in YAML
- Development Tools, IDEs
- Webserver mit Threads
- Anhang 1: Vergleich mit anderen Sprachen

Yukihiro „Matz“ Matsumoto

- * 1965, Japan
- „Well, Ruby was born in Feb. 23 1993.“
- „I [...] wanted a genuine object-oriented, easy-to-use object-oriented scripting language.“
- „I [...] couldn't find one.“
- „So, I decided to make it.“
- „I posted Ruby 0.95 to the Japanese domestic newsgroups in Dec. 1995.“



Hello, world!

```
puts „Hello, world!“  # --> „Hello, world!“
```

- `puts` = put as string, gibt einen String in einer Zeile aus
- Ruby Quellcode wird in Textdateien gespeichert
  - Dateiendung `.rb`
- Ruby-Programme ausführen:
  - Ruby-Interpreter mit Source als Parameter starten
    - `ruby /home/user/hallo.rb`
  - oder bei .rb-Dateien das Executable-Bit setzen, als erste Zeile `#!/usr/bin/env ruby` eintragen und die Dateien direkt ausführen

ri

- `ri` zeigt Beschreibungen zu Objekten wie Klassen und Methoden an
- Aufruf des Programms an einer Systemkonsole
  - `ri Class` zeigt Informationen über Class an
  - Beispiel mit `ri Object`:

```
---------------------------------------------- Class: Object
+Object+ is the parent class of all classes in Ruby. Its methods
are therefore available to all objects unless explicitly
overridden.

+Object+ mixes in the +Kernel+ module, making the built-in kernel
functions globally accessible.

[...]
```

irb

- interaktive Ruby Konsole
- Quellcode direkt eingeben
  - z.B. Methoden aufrufen, Objekte instanzieren, neue Klassen anlegen und nutzen

- „Hello, world!“ mit `irb`:

```
user@computer:~$ irb
irb(main):001:0> puts "Hello, world!"
Hello, world!
=> nil
irb(main):002:0>
```

Input/Output

- Klasse `IO`, davon erben andere I/O-Klassen wie z.B. `File`
- Beispiele simpler Ein-/Ausgaben mit den Methoden vom Modul `Kernel`

```
puts „Hello, World!“

p „Bitte geben Sie Ihren Namen ein:“

line = gets  #  z.B. Christian

putc line  # --> „Christian\n“
```

Variablen

- keine Deklaration des Typs
- Beginn der „Lebenszeit“ ab der ersten Zuweisung

- Konventionen für Variablennamen:
  - lokale Var.: kleiner Anfangsbuchstabe (`var_1`, `vAR1`)
  - Instanzvar.: beginnt mit @ und einem kleinen oder großen Buchstaben (`@var2`, `@VAR_3`)
  - Klassenvar.: mit @@ (`@@var_4`, `@@VAR5`)
  - Konstante: großer Anfangsbuchstabe (`Var6`, `VAR_7`)
  - globale Var.: beginnt mit $ (`$Var_8`, `$VAR9`)

Methoden

- eine Methode:

```
def max(a, b)
      (a > b) ? a : b
end
```

- Aufruf mit Klammern

```
max(5,6)  # --> 6
```

- Aufruf auch ohne Klammern möglich

```
max 5,6  # --> 6
```

Rückgabewert von Methoden

- mit return

```
def gib_1_zurueck
  # ggf. weitere Anweisungen
  return 1
end

gib_1_zurueck  # --> 1
```

- ohne return die Auswertung der letzten Zeile

```
def gib_auch_1_zurueck
  # ggf. weitere Anweisungen
  1
end

gib_auch_1_zurueck  # --> 1
```

- „kein“ Rückgabewert mittels `return nil`

Methoden

- Defaultwerte

```
def zweiWerte(par1, par2 = „vorgabe“)
  puts „1: #{par1}, 2: #{par2}“
end

zweiWerte(3, 4)  # --> „1: 3, 1: 4“

zweiWerte(5)  # --> „1: 5, 2: vorgabe“
```

- Zusatzinformation
  - Unterschied zwischen „ “ und ' '

```
def zweiWerte(par1, par2 = „vorgabe“)
  puts '1: #{par1}, 2: #{par2}'
end

zweiWerte(3, 4)  # --> '1: #{par1}, 2: #{par2}'
```

Methodenrückgabewerte für Vergleiche

- alle Objekte außer `nil` und `false` geben bei `is_true(objekt)` `true` zurück

```
def is_true(wert)
  wert ? true : false
end
```

- Operatoren: >, <, >=, <=, ==, !=, ...

Methodennamen

- Namenskonventionen
  - auf = endend sind setter (`var1=`)
  - auf ? endend geben boolschen Wert zurück (`is_a? Class`)
  - auf ! endend haben destruktive Wirkung bzw. ändern Daten, anstatt nur eine geänderte Kopie des Objektes zurückzugeben (`String.capitalize!`)

Methoden aufrufen, Punkt & Doppelpunkt

- Punktnotation

```
var = Array.new(3)
```

- Notation mit Doppelpunkten (eher in Dokumentationen)

```
var = Array::new(3)
```

Methoden schützen

- Wer darf welche Nachrichten an welche Objekte senden?
  - `public`
    - Standard
    - Methoden können von jedem angewendet werden
  - `protected`
    - Methoden können nur von Objekten derselben Klasse oder einer ihrer Unterklassen angewendet werden
  - `private`
    - Methoden können nur vom Objekt auf sich selbst angewendet werden

Methoden schützen

- Alle Methoden unterhalb eines Schlüsselwortes fallen in seinen Einfluss:

```
def pub_meth1
  # ...
end

protected
def prot_meth
  # ...
end

private
priv_meth
  # ...
end

public
def pub_meth2(var)
  # ...
end
```

Methoden schützen

- die Schlüsselwörter mit je einer Liste von Symbolen zu Methoden wirken auf diese Methoden:

```
def pub_meth1
  # ...
end

def prot_meth
  # ...
end

priv_meth
  # ...
end

def pub_meth2(arg)
  # ...
end

public    :pub_meth1, :pub_meth2
protected :prot_meth
private   :priv_meth
```

Aliase

- ggf. Methoden unter einem zweiten Namen sichern
- und dann unter dem ursprünglichen Namen eine neue Methode hinterlegen:

```
def als_String(etwas)
  etwas.to_s
end

als_String(2)  # --> „2“

alias alt_als_String als_String

def als_String(etwas)
  "[" + alt_als_String(etwas) + "]"
end

als_String(2)  # --> „[2]“
```

Methoden dynamisch aufrufen

- dynamische Aufrufe

```
n = 1234

puts n.send(:to_s)   # --> „1234“
puts n.send(„to_s“)  # --> „1234“

puts n.method(:to_s).call  # --> „1234“
puts n.method(„to_s“).call # --> „1234“
```

- `eval` kann Strings parsen und als Rubycode ausführen (langsamer als `call` oder send)

```
code = „puts n.to_s“
eval(code)  # --> „1234“
```

Klassen: Instanzmethoden & -variablen

```
class Person
  def initialize(name)
    @name = name
  end

  def gruesse
    „Hallo, ich heisse #{@name}.“
  end

  def to_s
    @name
  end
end

hugo = Person.new(„Hugo“)

puts hugo  # --> „Hugo“

puts hugo.gruesse  # --> „Hallo, ich heisse Hugo.“
```

Klassen: Klassenmethoden & -variablen

```
class Person
  @@geboren = 0
  def initialize(name)
    # ...
    @@geboren+=1
  end
  # ...
  def Person.melde_geborene
    @@geboren
  end
end

puts Person.melde_geborene
```

Getter & Setter

```
class Person
  # ...
  def name=(neuer_name)
    @name = neuer_name
  end
  def name
    @name
  end
end

m = Person.new(„Hugo“)

m.name = „Bert“  # unser m kommt ins Zeugenschutzprogramm

puts m.name  # --> „Bert“
```

Getter & Setter

- getter & setter halbautomatisch:

```
class Person
  attr_accessor :name
end

m = Person.new(„Hugo“)

m.name = „Bert“

puts m.name  # --> „Bert“
```

- statt `attr_accessor` kann man auch `attr_reader` und `attr_writer` einzeln benutzen

Objekte erweitern

- Objekte kann man dynamisch erweitern:

```
class Person
  # ...
  def stell_dich_vor
    „Ich bin #{@name}.“
  end
end

m = Person.new(„Hugo“)

def m.verabschiede_dich  # klappt aber z.B. nicht wenn m = 2
  „Tschuess!“
end

print m.stell_dich_vor + „\n“ + m.verabschiede_dich
# --> „Ich bin Hugo.
#      Tschuess!“
```

- bei m = 2 „TypeError: can't define singleton method "verabschiede_dich" for Fixnum“

Abstrakte Klassen

- enthalten gemeinsame Methoden und Attribute für ihre Unterklassen
- lassen selbst keine Exemplarbildung zu (new-Methode nicht vorhanden bzw. nicht nutzbar)

```
# Normale Klassen:
String.new  # OK
Array.new  # OK

# Beispiel für Abstrakte Klasse:
Integer.new  # „undefined method `new' for Integer:Class“
```

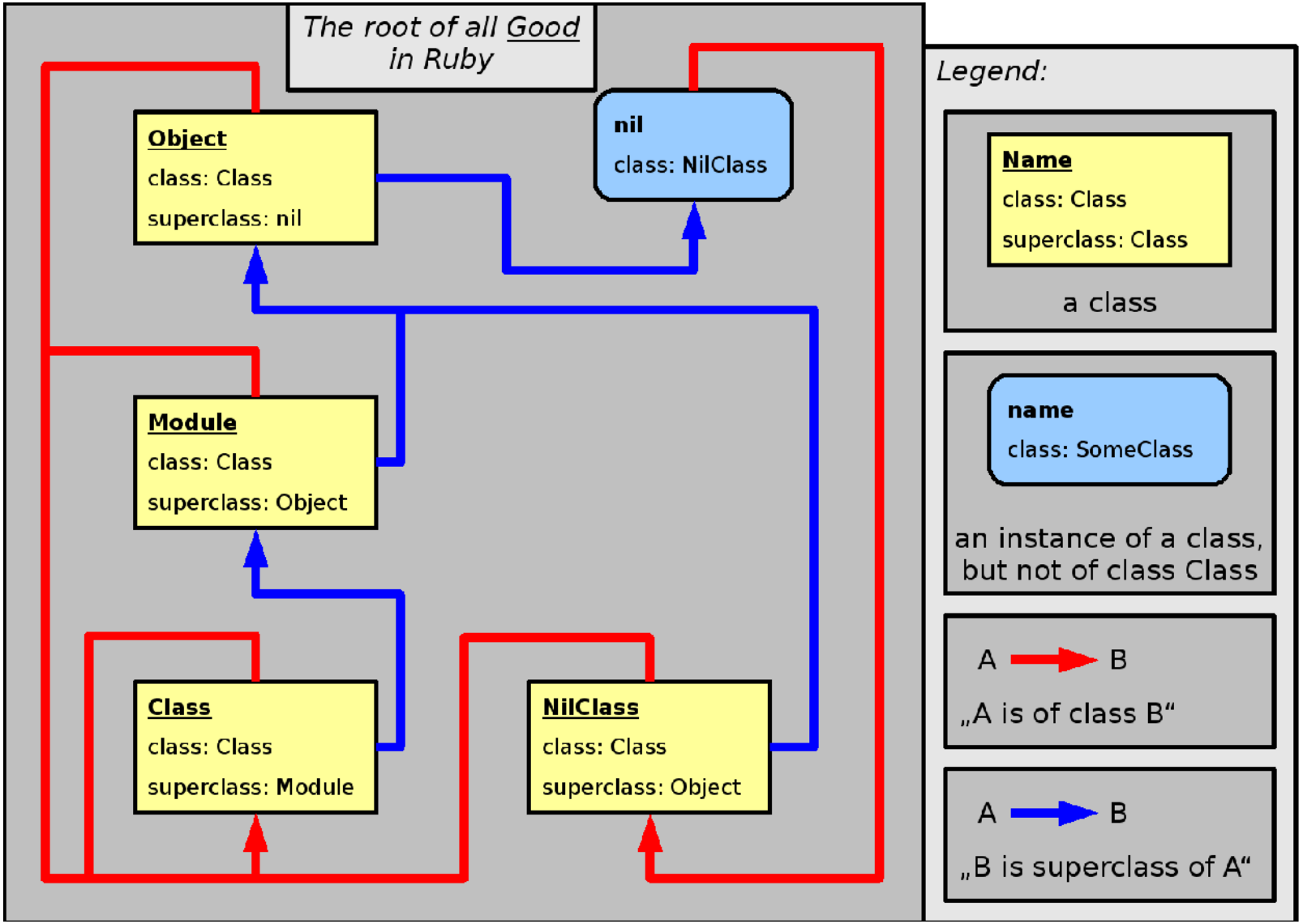
Vererbung

- Klassen erben von genau einer Oberklasse (Standard ist `Object`)

```
class Oberklasse
end

class Erbe < Oberklasse
  def sag_oberklasse
    puts self.class.superclass
  end
end

x = Erbe.new

x.sag_oberklasse  # --> „Oberklasse“
```

Vererbung

- ganz oben in der Hierarchie (vereinfacht):



frei nach http://case.lazaridis.com/attachment/wiki/RubyObjectModel/TheRubyObjectModel.png

Module

- Module erinnern etwas an abstrakte Klassen und Java Interfaces, sind aber flexibler einsetzbar

```
include Math  # das Mathematikmodul wird geladen
puts PI  # --> „3.141592653589792“

module MeinModul
  PI = 3
  def berechne_Kreisflaeche(radius)
    radius*radius*PI
  end
  def MeinModul.info
    "Mathematik mal anders"
  end
end

include MeinModul  # MeinModul geladen
puts PI  # --> „3“
puts berechne_Kreisflaeche(2)  # --> „12“
puts MeinModul.info  # --> „Mathematik mal anders“
```

Module & Mix-Ins

- Module kann man in Klassen, andere Module und sogar einzelne Exemplare laden (Mix-In)

```
class Klasse
  include MeinModul
end

module NochEinModul
  include MeinModul
end

o = „Hugo“
o.extend(MeinModul)
o.berechne_Kreisflaeche(2)  # --> 12
```

Symbole

- Symbole sind Namen (Variablen kennzeichnen Referenzen)
- Syntax `:symbol`

```
module EinModul
  class Christian
  end

  $gv1 = :Christian
end

Christian = 1234
$gv2 = :Christian  # auch hier wird noch der alte :Christian verwendet!

Peter = 9876
$gv3 = :Peter

puts $gv1.object_id  # --> 161898
puts $gv2.object_id  # --> 161898
puts $gv3.object_id  # --> 162138
```

Blöcke

- ein Blockobjekt enthält eine Folge von Anweisungen
- meist werden do und end statt {} benutzt

```
class Blockverarbeiter
  def gib_block_12(&block)
    block.call(12)
  end

  def fuehr_block_aus
    if block_given? then yield end  # yield führt Blöcke aus
  end
end

bv = Blockverarbeiter.new

bv.gib_block_12 { |arg1| puts arg1 }  # --> 12

bv.fuehr_block_aus { puts „Hallo“ }  # --> „Hallo“
```

Iteratoren

- hier spielen Blöcke eine wichtige Rolle

```
class Obstkiste
  attr_accessor :a
  def initialize
    @a = Array.new
  end
  def pack_hinein(obst)
    @a << obst
  end
  def each
    for obst in @a do yield(obst) end
  end
end

ok = Obstkiste.new
ok.pack_hinein(„Orangen“)
ok.pack_hinein(„Zitronen“)

ok.each { |fruechte| puts fruechte }  # --> „Orangen“  „Zitronen“
```

• *über Bereiche & Arrays iterieren*

```
bereich = 3..7

bereich.each do |element|
  puts element
end



array = [1, 2, 3, 4, 5]

array.each do |ea|
  puts ea
end
```

Exceptions

```
class MeineException < Exception
end

def verarbeiteInt(i)
  if i.is_a?(Integer)
    puts i
  else
    raise MeineException
  end

  rescue MeineException => e
    puts "asdf: #{e}"

  rescue => e
    puts "irgendeine Exception: #{e}"

  ensure
    puts "dies muss raus"
end

verarbeiteInt(12) --> 12; „dies muss raus“

verarbeiteInt("asdf") --> „asdf: MeineException“; „dies muss raus“
```

Reguläre Ausdrücke

- Suchen und Ersetzen in Strings mittels Mustern
- seit Jahrzehnten in Unix, konzeptionell seit den 1940er Jahren
- populär durch Perl
- wenig Code --> komplexe Effekte („mächtig“)

- in Ruby objektorientiert gekapselt und traditionell wie auch objektorientiert nutzbar
- Objekte der Klasse Regexp

```
ra = /b/  # traditionell
"abcd" =~ ra  # --> 1, die Position des b im String, erster Index ist 0
ra.class  # --> Regexp

ra = Regexp.new('b')  # objektorientiert
"abcd" =~ ra  # --> 1
ra.class  # --> Regexp
```

Dateien

- erst Schreiben, dann Lesen

```
datei = File.new("testdatei","w+")  # w wie „write“, + für „anhängen“
loop do
  zeile = gets
  break if zeile.chomp.length == 0
  datei.puts(zeile)
end
datei.close

puts File.size("testdatei")

datei = File.open("testdatei","r")  # r wie „read“
while (zeile = datei.gets)
  puts zeile
end
datei.close
```

Objekte speichern & laden in YAML

- um nicht in einem unleserlichen Binärformat zu speichern, kann man YAML statt des eingebauten Marshallers zu nutzen

```
require 'yaml'  # die YAML-Bibliothek laden
class Person
  def initialize(name)
    @name = name
  end
  def to_yaml_properties
    %w{@name}
  end
  def to_s
    @name.to_s
  end
end

m = Person.new(„Christian“)

in_datei = YAML.dump(m)  # in_datei könnte man jetzt in eine Datei schreiben
puts in_datei  # --> --- !ruby/object:Person
               #     name: Christian

m = YAML.load(in_datei)
puts m.to_s  # --> „Christian“
```

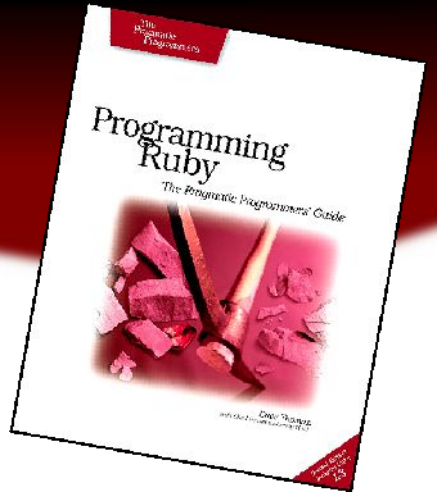
Development Tools, IDEs

- Kommandozeilen Tools
  - ruby (interpreter)
  - irb (interactive shell)
  - ri (zeigt Hilfe zu Klassen / Methoden)
  - rdoc (erzeugt Dokumentationsdateien)
- andere:
  - Ruby Development Tools für Eclipse (IDE)
  - SciTE (IDE)
  - Freeride (IDE)
  - Mondrian (IDE)
  - Arachno (IDE)
  - (...)

Webserver mit Threads

- zeigt im Browser die Uhrzeit des Servers an aus dem Moment, wenn der Code ausgeführt wurde

```
require "socket"
port = 9999
server = TCPServer.new("localhost", port)
while (session = server.accept)
  Thread.new do
    puts "Incoming request is #{session.gets}"
    session.print "HTTP/1.1 200/OK\r\nContent-type: text/html\r\n\r\n"
    session.print "<html><body><h1>#{Time.now}</h1></body></html>\r\n"
    session.close
  end
end
```



- Quellen & weitere Informationen zu Ruby:
  - **Buch: „Programming Ruby The Pragmatic Programmer's Guide“** von Thomas, Fowler, Hund
  - http://www.stal.de/Downloads/OOP2005/ruby.pdf
  - http://edi01.informatik.fh-wiesbaden.de/~werntges/lv/ruby/pdf/ws2003/ruby-vertiefung.pdf
  - Webseiten:
    - **Homepage: http://www.ruby-lang.org/de**
    - **Bücherliste**: http://www.sapphiresteel.com/Ruby-and-Rails-Books-the-essential
    - **„Programming Ruby“ Buch**: www.rubycentral.com/book
    - **Windows Installer**: http://rubyinstaller.rubyforge.org
    - **Ruby2**: http://wiki.rubygarden.org/Ruby/page/show/Rite
    - **Blogverzeichnis**: http://www.rubycorner.com
    - **Newsblog**: http://www.rubyinside.com
      - als RSS-Feed: http://www.rubyinside.com/feed
    - **einige „Einzeiler“**: http://www.fepus.net/ruby1line.txt
    - **reguläre Ausdrücke**: http://www.troubleshooters.com/codecorn/littperl/perlreg.htm
    - **Libraries**: http://www.rubyforge.org
    - **Ruby Production Archives**: http://www.rubyarchive.org
    - **Portal and Wiki**: http://www.rubygarden.org
    - **Dokumentation**: http://www.ruby-doc.org
  - Newsgroup:
    - comp.lang.ruby (sehr aktiv, auch Matz schreibt hier!)

Anhang 1: Vergleich mit anderen Sprachen

	Ruby	Objective-C	SmallTalk-80	C++	Java	Python	CLOS	Perl5	XOTcl
Typing	dynamic	dynamic/static	dynamic	static	static	dynamic	dynamic	mostly dynamic	dynamic
Runtime access to method names	yes	yes	yes	no	yes	yes	yes*	yes	yes (including method definitions)
Runtime access to class names	yes	yes	yes	no	yes	yes	yes*	yes	yes (including class definitions)
Runtime access to instance variable names	yes	yes	yes	no	yes	yes	yes*	yes	yes
Forwarding	yes	yes	yes	no	no (?)	yes	??	yes	yes
Metaclasses	yes	yes	yes	no	yes	yes	meta objects	no	yes
Inheritance	mix-in	single	single	multiple	single	multiple	mix-in	multiple	mix-in, multiple (per object or per class)
Access to super method	super	super	super	Superclass::method	super	(Python 2.1 syntax): super(MyClass,self).methodName	call-next-method	SUPER::methodName	next
root class	Object	Object (can have multiple)	Object (can have multiple)	none	Object	none	t	"UNIVERSAL" package	Class/Object
Receiver name	self	self	self	this	this	self	none, true generic functions	$_[0]	[self]
Private Data	yes	yes	yes	yes	yes	yes	no	no	no
Private methods	yes	no	no	yes	yes	yes	no	no	no
Class Variables	yes	no	yes	yes	yes	yes	yes	yes	yes
Templates	not needed(1)	not needed	not needed	yes	no	not needed	not needed	not needed	not needed
Garbage Collection	yes	yes	yes	no	yes	yes	yes	reference counting	optional with volatile keyword (reference counting)

von http://www.approximity.com/ruby/Comparison_rb_st_m_java.html