

紅玉 Ruby

Einführung in Konzepte
und Möglichkeiten

Ablauf dieses Vortrags

- Ausflug... ins Land der aufgehenden Sonne ☺
- Was ist Ruby? – Was will Ruby? – Was bietet Ruby?
- Teil 1: Grundlagen Programmierung
- Teil 2: Objektorientierung
- Teil 3: Stärken, Schwächen, Ausblick

Ausflug ... ins Land der aufgehenden Sonne... ☺

- 1993: Erfinder Yukihiro 'matz' Matsumoto will eine Skriptsprache
 - „mächtiger als Perl“ und
 - „objektorientierter als Python“
- matz ist ein fauler Programmierer (sagt er)
 - Der Computer soll „mehr arbeiten als der Mensch davor“
 - Der Programmierer soll „glücklich sein und sich nicht ärgern“
- 1995: Veröffentlichung Ruby Version 0.95
- seit 24.12.2002 aktuell: Version 1.6.8

Plattformen

- Theoretisch: lauffähig auf jeder Plattform, für die ein ANSI-C-Compiler existiert
- Praktisch existieren Ruby-Interpreter unter anderem für
 - alle Windows-Versionen
 - MS-/PC-DOS
 - Linux
 - BSD-Varianten
 - Mac OS und Mac OS X
 - OS/2
 - AIX
 - BeOS
 - die Cygwin-Umgebung
 - PalmOS

Was ist Ruby? – Was will Ruby?

- Ruby ist
 - interpretiert
 - vollständig objektorientiert
 - portabel
- Ruby will
 - einfach sein
 - universell sein
 - wenig Schreibarbeit machen
 - schnelle Ergebnisse ermöglichen
- Ruby erinnert
 - syntaktisch an Perl und Python
 - an Smalltalk: „Everything is an object“ (even classes themselves are objects)

Was bietet Ruby?

- Ruby bietet, wie viele Skriptsprachen:
 - viele (Bedingungs-)Operatoren & Schleifenarten
 - schwache Typisierung von Variablen
 - ausgeprägte Unterstützung von
 - Arrays und Hashtabellen
 - Strings
 - Regular Expressions
- Ruby bietet aber auch viele OO-Features, u.a.:
 - Klassen, Vererbung, Überschreiben, Überladen, ...
 - Polymorphismus
 - Reflection
- Ruby bietet außerdem
 - MixIns
 - Exception-Handling
 - einige aus C bekannte Funktionen

Bedingungsoperatoren

```
n = 2
if n > 4
  puts("this")
elsif n > 2
  puts("that")
else
  puts("whatever") unless n==1
end
#→ whatever
```

```
a = 0
a = (if a==0 then 1 else 0 end)
||
a = (a==0)?1:0
```

Operatoren

- Wichtige, von Java abweichende Operatoren:

****** Potenzierung: `2**8 == 256`
>> Multiplikation mit der Potenz von 2: `2<<9 == 1024`
<< Division mit der Potenz von 2: `1024>>9 == 2`

- Gleichheitsoperatoren:

== prüft auf Gleichheit der Werte
eql? prüft zusätzlich die Klassen zweier Objekte
equal? vergleicht interne Objektnummern

- Beispiel:

```
s = "Str1"; t = "Str1"; u = "Str2"
```

s vergleicht sich mit	s	t	u
durch ==	true	true	false
durch eql?	true	true	false
durch equal?	true	false	false

Kontrollstrukturen: loop, while, until

```
c = 0
loop {
  print "#{c} "
  c += 1
  print "#{c}, "
  break if c==5
}
#→ 0 1, 1 2, 2 3, 3 4, 4 5,
```

Auch möglich:

```
while (<Bedingung>) do ... end
```

```
until (<Bedingung>) do ... end
```

(Bedingungen können auch nach dem Schleifenkörper stehen)

Schleifenkontrollen:

redo wiederholt die aktuelle Schleifeniteration

next beginnt sofort mit der nächsten Iteration

break Schleife wird sofort verlassen

Kontrollstrukturen: for, each

- for-Schleife:

```
for c in 1..12
  puts c
end
```

→ ergibt : 1 2 3 4 5 6 7 8 9 10 11 12

- each: Iteration über Mengen von Elementen:

```
a = [5,6,8,1]
```

```
a.each { |element| puts element } #→ 5 6 8 1
```

Objekt
Methode

Parameter

Block

- Ruft den Block einmal für jedes Element im Array a, und übergibt dabei jedes Element als Parameter an den Block

Arrays und deren Operationen

- Funktionalität ähnlich wie in Java...
- ...aber andere Notation:
 - Implizit: `a = [1, "2"]`
 - Explizit: `a = Array.new(2)`
`a[0] = 1; a[1] = "2"`
- ... und viel mehr Operationen:
 - Zusammenhängen mit und ohne Duplikate
 - Vervielfältigen, Vereinigung, Schnittmenge, Differenzbildung
 - Umkehren der Reihenfolge, Herausschneiden von Bereichen, ...
 - Verwendung als Stacks oder als Queues

- Beispiel Schnittmenge:

```
b = [1, "2"]
```

```
c = ["test", "1", "2"]
```

```
h = c & b
```

```
→ h = ["2"]
```

- Beispiel Verwendung als Stack:

```
d = [1, 2, 3]
```

```
d.push(4) # → [1, 2, 3, 4]
```

```
d.pop    # → 4
```

Hashtables und deren Operationen

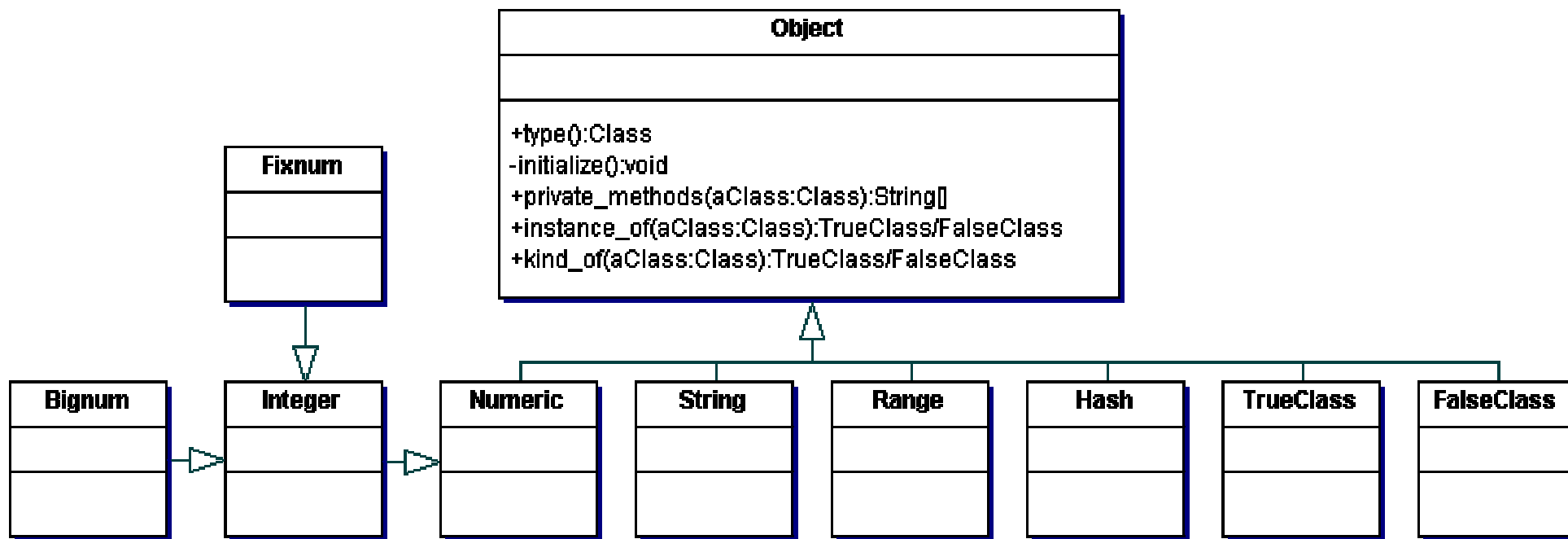
- Hashtables sind assoziative Arrays, Keys sind Objekte
- sind ungeordnet, Zugriff erfolgt über Keys, nicht über Indizes
- Erzeugung:
 - `a = { "key1" => "tschüss", "key2" => "welt" }`
 - `b = { "key1" => "hallo", "key3" => "es war schön" }`
- Zugriff:
 - `b["key3"]` # → "es war schön"
- Beispiele für Methoden:
 - `b.update(a), b.values(), b.keys(), b.size()`
 - `b.has_key?("key1")` # → true
 - `b.has_value?("hallo")` # → true
 - `b.index("hallo")` # → key1
 - `b.delete("key3")` # → es war schön
- Iteration über Hashtables:
 - `b.each_value { |value| puts value }` # → tschüss welt

Rückblick Grundlagen, Ausblick OO

```
n = 2
r = 1..12
a = {"key1" => "tschüss"}
s = "tschüss"
b = [5, "6", 8..9]

puts n.class() # → Fixnum
puts r.class() # → Range
puts a.class() # → Hash
puts s.class() # → String
puts b.class() # → Array
```

Auszug aus der Klassenhierarchie von Ruby:



Klassen, Vererbung, einfache Reflection

- Schon bei den Arrays haben wir gesehen, wie ein Objekt der Klasse Array erzeugt wird:

```
a = Array.new(2)
puts a.class      #→ Array
```

- Definition einer Klasse **K**: Definition einer Subklasse **S**:

```
class C
end
```

```
class S < C
end
```

- Erforschen der Klassenhierarchie durch Reflection:

Ist S Subklasse von K?

```
puts S < K    → Ausgabe: true
```

Ist K Subklasse von S?

```
puts S > K    → Ausgabe: false
```

- Einfachvererbung
- Alternative zur Vererbung über Klassen sind Module (später)

Klassen, Methoden

- Definition einer Methode:

```
class MeineKlasse
  def meineErsteMethode()
    puts "Fred"
  end
end
```

- Späteres, erneutes Definieren einer Klasse wirkt inkrementell:

```
class MeineKlasse
  def meineZweiteMethode(aParameter)
    puts "Bob ist #{aParameter}!"
  end
end
```

→ Methode `meineZweiteMethode` wird der Klasse hinzugefügt, die Klasse hat nun `meineErsteMethode()` und `meineZweiteMethode(...)` als Methoden (Erneutes Definieren einer Methode mit selbem Namen überschreibt bisherige)

- Auch Systemklassen lassen sich erweitern und ändern!

Konstruktor

- Überschreiben der Methode `initialize`:

```
class MeineKlasse
  def initialize()
    puts "initialize wurde gerufen"
  end
end
```
- Instanziierung:
`obj=MeineKlasse.new()` #→ initialize wurde gerufen (aber wie? 😊 s.u.)
- Frage: Wo ist Methode `new()` definiert?
 - Es gibt keine Methode `new()` in `MeineKlasse`
 - Es gibt keine Methode `new()` in `Object`
- Antwort liegt im „First-Class-Objects“-Konzept von Ruby
 - „Classes in Ruby are first-class objects. Each is an instance of class `Class`.“
 - Beim Erstellen einer neuen Klasse mit `class MeineKlasse...end` wird ein Objekt vom Typ `Class` instanziiert und der globalen Konstante zugewiesen, die sich `MeineKlasse` nennt

Klassenmethoden und „Singleton-Methoden“

- Definition einer Klassenmethode:

```
class MeineKlasse
  def MeineKlasse.statisch()
    puts "stat"
  end
end
```

- Aufruf einer Klassenmethode:

```
MeineKlasse.statisch() # → stat
```

- „Singleton-Methoden“ gelten nur für ein bestimmtes Objekt
 - ändern das Verhalten einzelner Objekte
 - Code-Beispiel: Datei SingletonMethoden1.rb

Objektorientierung

- Zahlen sind Objekte
- Operatoren symbolisieren Methodenaufrufe:
 - `1+2` ist aus Bequemlichkeit möglich, aber eigentlich lautet der Aufruf
 - `1.+(2)` wobei `+` der Methodename ist und `2` der Übergabeparameter
- Operatoren können überschrieben werden

Geheimhaltung: Variablen

- Sichtbarkeit

- Präfix @ Instanz-Variablen
- Präfix @@ Klassen-Variablen
- Präfix \$ globale Variablen
- GROSS geschrieben, ohne Präfix Konstanten und Klassennamen
- klein geschrieben, ohne Präfix lokale Variablen (klein geschrieben)

- Zugriff auf Instanz-Variablen nur mittels Methoden:

```
def instanzVar()  
  return @instanzVar  
end
```

```
def instanzVar=(newInstanzVar)  
  @instanzVar = newInstanzVar  
end
```

- Kurzschreibweise:

```
attr_reader :instanzVar
```

```
attr_writer :instanzVar
```

- Lesen und Schreiben:

```
attr_accessor :instanzVar
```

- Aufruf bei vorhandener Sichtbarkeit:

```
obj.instanzVar
```

```
obj.instanzVar=50
```

Geheimhaltung und weitere Infos: Methoden

- Modifizierung der Sichtbarkeit durch Voranstellen von
 - `private` nur das aktuelle Objekt selbst hat Zugriff
 - `protected` aktuelles Objekt oder Objekt einer Subklasse hat Zugriff
 - `public` von überall aus zugreifbar
- Sichtbarkeit von Methoden kann zur Laufzeit geändert werden
- Methoden können nicht überladen werden, statt dessen verwendet man optionale Parameter

```
def method(a, *b) # nein, das ist kein Pointer ;-)  
end
```

 - `*b` bedeutet, dass 0..* Parameter übergeben werden können

Modules

- sind nicht-instancierbare Klassen
- können weder erben noch vererben
- sammeln Methoden, Klassen und Konstanten
- begrenzen den Namensraum
- werden mit **include** anderen Klassen zur Verfügung gestellt
 - daher die Bezeichnung „MixIn“
- Deklaration:

```
module EinModule
  # ein paar Klassen
  # ein paar Methoden
  # ein paar Konstanten
end
```
- Verwendung:
 - Code-Beispiel: Datei ModuleDemo1.rb

Fehlerbehandlung

```
begin
  # heikler Code
rescue [<Ausnahmeklasse> [, <Ausnahmeklasse>+] [=] <Variable>]
  # Fehlerbehandlungs-Code
else
  # Code, wenn keine Ausnahme gefangen wurde
ensure
  # Code, der auf jeden Fall ausgeführt wird
end
```

- Wie bei Java hierarchisch geordnete Ausnahmeklassen
- Auch Hardware-Interrupts lassen sich fangen

Einsatzgebiete (1)

- Systemverwaltung
 - Viele Möglichkeiten zur Operation auf Strings (z.B. RegExp)
 - Skripte lassen sich von der Shell aus aufrufen
 - in Ruby geschriebene XML-Parser und –Generatoren einbindbar
- GUI-Applikationen
 - keine eigenen GUI-Funktionen
 - Abstützung auf Bibliotheken von Drittanbietern
 - bereits enthalten: Interface zu Tk
 - weitere erhältliche Interfaces: GTK, Trolltech Qt, OpenGL, ...
 - dabei voller Zugriff auf die Eigenschaften der GUI-Elemente
- Anbindung von Datenbanken
 - Abstützung auf Bibliotheken von Drittanbietern
 - entweder proprietäre Interfaces für je ein RDBMS (MySQL, DB2, Oracle)
 - oder Ruby/DBI als DBMS-unabhängige Schnittstelle (Link siehe Ressourcen)

Einsatzgebiete (2)

- Windows-Programmierung
 - OLE-Unterstützung: Windows-Applikationen mit entsprechender Schnittstelle können aufgerufen und gesteuert werden (z.B. Office-Applikationen, Internet Explorer, ...)
- Dynamische Erzeugung von HTML-Code im Web-Server
 - zwei Ansätze:
 - Erzeugung eines HTML-Dokuments ähnlich wie bei Java Servlets
 - Einbettung von Ruby-Code in HTML (wie PHP/JSP): eRuby
 - dadurch Einsatz als Application Server möglich

Einsatzgebiete (3)

- Integration bestehenden C-Codes
- Netzwerk-Programmierung
 - Umfangreiche mitgelieferte und herunterladbare Modules
 - Low-Level:
 - Sockets
 - TCP/UDP
 - High-Level:
 - SMTP und POP3
 - IMAP4
 - NNTP
 - Telnet
 - SOAP, mit SOAP4R (Link siehe Ressourcen)

Einsatzgebiete (4)

- Zu weiteren Möglichkeiten siehe Ruby Application Archive (RAA):
<http://raa.ruby-lang.org>
- Willkürlicher Auszug aus der Liste des RAA:
 - log4r Logging (inspiriert durch log4j für Java)
 - Blogtari! Automatisiertes Generieren eines Weblogs
 - mp3info Zugriff auf low-level-Infos von MP3s (Bitrate, Länge, ...)
 - NETRuby Ruby on Common Language Runtime
 - gesamt: 898 Projekte (Stand: 4.6.2003)

IDEs und Teamentwicklung

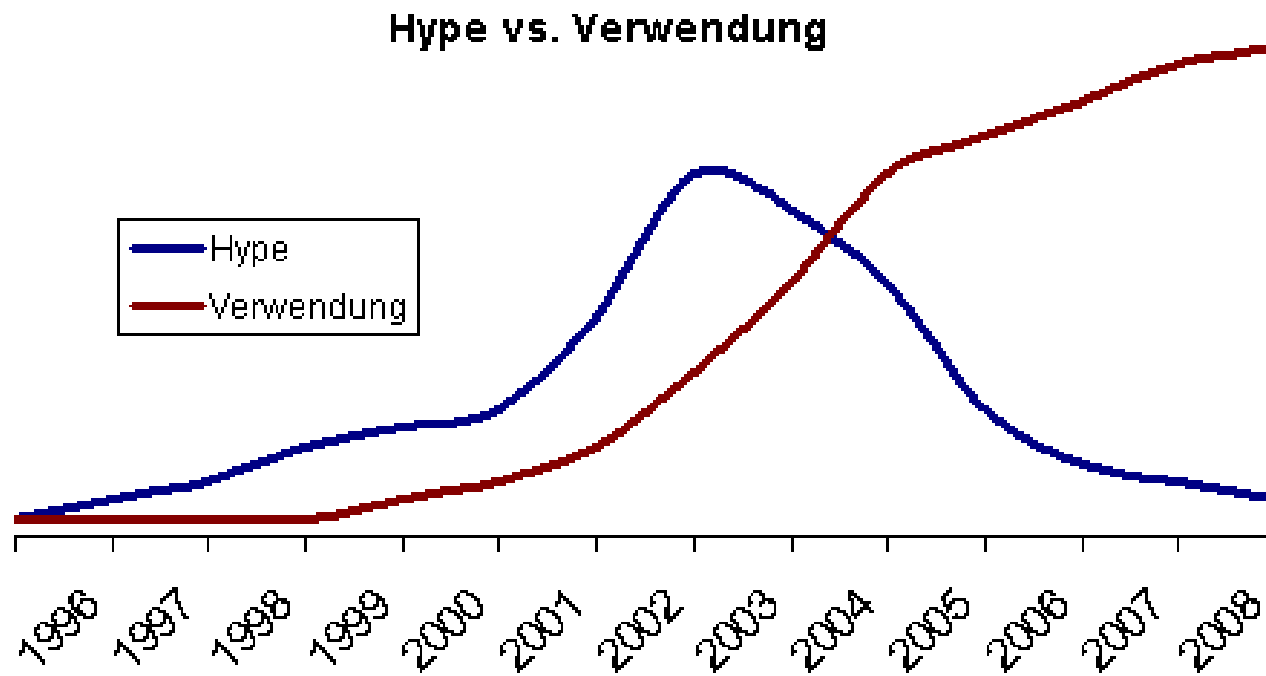
- IDEs
 - [FreeRIDE](#) (Free Ruby IDE, in Ruby geschrieben, Features ok, nicht stabil)
 - [rubyclipse](#) (Link siehe Ressourcen)
 - [SciTE](#) (universeller Editor mit Highlighting für Ruby)
 - [UltraEdit](#) (universeller Editor mit erschöpfenden Features)
 - Einbinden des Interpreters leicht, Ausgabe erfolgt in UltraEdit-Fenster
 - Wordlist im Internet schwer zu finden
- Teamentwicklung
 - Ruby ist file-basiert → CVS ☺

Stärken

- Sehr mächtige, universelle Sprache
 - konsistente Syntax, daher schnell zu erlernen
- Durchgehend objektorientiert
- Sehr viele Module verfügbar
- Sehr starke Community (Links siehe Ressourcen)
- Offenheit der Sprache
- Dynamik
- Vergleichsweise gute Performance (schneller als Python)

Schwächen

- Geringer Bekanntheitsgrad
- Keine großen Unternehmen, von denen Ruby unterstützt wird; so gut wie keine Stellenausschreibungen mit „Ruby“ im Text
- Nicht viele Bücher erhältlich (und die „hyped“ fast ausschließlich)
- Großes RAA, jedoch scheinbar kein definierter Integrationsprozess dieser Ideen in Basis-Funktionalität der Sprache, Erfinder matz koordiniert Weiterentwicklung alleine
- Integration der Bibliotheken von Drittanbietern schwach dokumentiert
- IDEs ausschließlich Beta-Ware
- Vorteile gegenüber ähnlichen Sprachen fallen erst auf den zweiten Blick auf
- Leidensdruck zum Umstieg von Perl oder Python fehlt



Fragen und Antworten

Danke für Eure Aufmerksamkeit 😊

Interesse geweckt? → Ressourcen

- Ruby-Homepage und Download <http://www.ruby-lang.org/en/>
- One-Click-Windows-Installer <http://rubyinstaller.sourceforge.net>
- Ruby-„Zentrale“
<http://www.rubycentral.com>
<http://www.rubycentral.com/faq/rubyfaqall.html> (FAQ)
- Ein Wiki zu Ruby <http://www.rubygarden.org/>
- Ruby-Newsgroup <news://comp.lang.ruby>
- Cooler Dialog, bei dem man ganz nebenbei Ruby lernt <http://visibleworkings.com/little-ruby/>

- Einführungen in Ruby
<http://www-106.ibm.com/developerworks/linux/library/l-ruby1.html>
<http://www.pragmaticprogrammer.com/ruby/> (Tips, Tricks und Erfahrungsberichte)

- Interessante Diskussionen (nicht nur zu Ruby)
<http://www.linuxjournal.com/article.php?sid=5915>
<http://www.linuxjournal.com/article.php?sid=4834>

- Bücher zu Ruby Röhrl, Schmiedl, Wyss: "Programmieren mit Ruby", ISBN 3-89864-151-1
- Vergleiche zwischen Ruby und anderen Sprachen
<http://www.ruby-lang.org/en/compar.html>
<http://mail.python.org/pipermail/python-list/1999-October/014769.html>

- eRuby <http://www.modruby.net>
- Ruby/DBI <http://ruby-dbi.sourceforge.net>
- Weitere Technologien und Tools
<http://jruby.sourceforge.net/> (Java implementierung des Ruby Interpreters)
<http://www.jamesbritt.com/articles/blogatari.html> (Blogtari!)
<http://raa.ruby-lang.org/list.rhtml?name=soap4r> (SOAP for Ruby, SOAP4R)
<http://log4r.sourceforge.net/> (Logging mit Ruby, log4r)
<http://rubyclipse.sourceforge.net> (IDE als Plugin für das Eclipse-Framework)

- Mail an den Entwickler <mailto:matz@netlab.co.jp>