

# Ruby - Objektorientiertes Scripting, aber richtig!

Andreas Krennmair

[krennmair@acm.org](mailto:krennmair@acm.org)

19. März 2003

## **Agenda**

- Ruby's Syntax
- Ruby's coole Konzepte
- Sprachverwandtschaften
- Wofür Ruby geeignet ist und wofür nicht

## **Einleitung**

- Ab 1993 von Yukihiro “Matz” Matsumoto entwickelt
- Fand die “Objektorientierung” von Perl 5 zu eklig und Python zu wenig konsequent
- Entschloss sich, es selber besser zu machen, und begann mit Ruby
- Derzeit in der Version 1.7/1.8 aktuell
- Läuft auf den meisten Unix-Derivaten, Win32, BeOS, Mac OS 9 und DOS

## **Ruby's Features**

- Einfache Syntax. Ruby-Code ist so einfach zu lesen, wie er zu schreiben ist (*sehr einfach*)
- Objektorientierte Features: Klassen, Objekte, Methoden, Datenkapselung, ...
- Mehr Objektorientierung: Mixin-Konzepte, sehr einfache Realisierbarkeit von Design Patterns
- Noch mehr Objektorientierung: alles ist ein Objekt, sogar Klassen!
- Operator Overloading
- Exception Handling
- Iteratoren: konsequent in allen Containern der Standardbibliothek durchgezogen, für eigene Container *sehr* einfach selbst implementierbar
- Garbage Collection. Was wäre eine Skriptsprache ohne Garbage Collection? ;-) )

## **Ruby's Syntax**

- Elemente strukturierter Programmierung
- Anonyme Codeblöcke
- Scopes
- Klassen und Objekte

## **Sequenz**

- Eine Anweisung pro Zeile
- Mehrere Anweisungen können mit Strichpunkt getrennt in eine Zeile geschrieben werden

```
$stderr.puts "Hello, World"  
i = 3 ; j = 4 ; k = 5
```

## **if-Anweisungen**

```
if i == 3 then
  puts "bla"
elsif i == 4 then
  puts "fa"
else
  puts "sel"
end
```

Das then ist übrigens optional.

## **Anweisungs-Modifizier**

```
$stdout.puts "blafasel" if i == 3
$stderr.puts "foobar" unless not (i == 3)
```



## **Mehrfachverzweigung mit case**

```
case inputLine
  when "debug"
    dumpDebugInfo
  when /print (\w+)/
    dumpVariable($1)
  when "quit", "exit", "bye"
    exit
  else
    print "Ungültiger Befehl '#{inputLine}'"
end
```

## **while-Schleife**

```
i = 0
while i<3 do
  puts "#{i}"
  i += 1
end
```

Das `do` ist übrigens ebenfalls optional.

`while` gibt es auch in der Anweisungs-Modifier-Variante:

```
puts "Stop Dave, please" while true
```

## **Anonyme Codeblöcke**

- Anonyme Codeblöcke können Methoden “angehängt” werden
- Angehängter Codeblock kann mit `yield` aufgerufen werden
- Dem Codeblock können Parameter übergeben werden

- Beispiele:

```
einObjekt.eineMethode do
  # Code
end
```

```
einObjekt.methode2 {
  # code
}
```

## **Scopes**

- Scope der Variable wird durch Prefix angegeben
- Kein spezieller Prefix: lokale Variable, Geltungsbereich bis zum Blockende
- \$ Prefix: globale Variable (z.B. \$\_)
- @ Prefix: Membervariablen von Objekten
- @@ Prefix: Klassenvariablen

## **Klassen und Objekte**

- Methodendefinitionen zwischen `class FooBar` und `end` zu finden
- Methoden werden mit `def quux` definiert, Ende wird mit `end` markiert
- Die Methode `initialize` ist der Konstruktor
- Membervariablen sind grundsätzlich ohne Ausnahme "private"
- Ausweg: Zugriffsfunktionen, genannt "Accessors". Können
  - händisch definiert
  - automatisch generiert werden
- Variablen sind übrigens typlose Referenzen auf Objekte

## **Beispiel für eine Klasse**

```
class FooBar

  def initialize(param1 = 0)
    @foo = param1
  end

  def foo
    @foo
  end

  def foo=(f)
    @foo = f
  end

end
```

## **Iteratoren**

- Problem: man will möglichst generisch auf alle in einem Container gehaltenen Elementen zugreifen
- Lösung: Iteratoren
- Es wird an eine Iteratormethode ein Codeblock übergeben, welche dann von der Iteratormethode für jedes Element im Container aufgerufen wird
- Ein Iterator, der in allen Containern der Ruby-Standardbibliothek zu finden ist, ist `each`
- Beispiel:

```
namen = [ "Hans", "Franz", "Patrick", "Rudi", "Alex" ]
namen.each { |n| puts "#{n} ist ziemlich cool." }
```

## **Mehr Iteratoren**

- Iteratoren werden für die Konstruktion verschiedenster Schleifen verwendet

- Beispiele:

```
summe = 0
durchlaeufer = 99.times do |x|
  summe += x
end
```

```
puts "Durchschnitt aller Zahlen von 0 bis 99:"
puts "#{summe/durchlaeufer}"
```

```
1.upto(10) do |i|
  i.downto(1) { print "*" }
  puts ""
end
```



## **Iteratoren selberrmachen (1)**

- Wie bereits erwähnt, können an Methoden anonyme Codeblöcke angehängt werden
- Diese Codeblöcke können mit `yield` aufgerufen werden

- Beispiel:

```
def funktion
  yield 4
  yield 5
end
```

```
funktion { |zahl| puts "#{zahl}*{zahl} = #{zahl*zahl}" }
```

- Ausgabe:

```
4*4 = 16
5*5 = 25
```

## **Iteratoren selbermachen (2)**

Beispiel:

```
class MyContainer
  # ...
  def each
    i = 0
    while i < @arr.length do
      yield @arr[i]
    end
  end
end
```

## **Klassen und Objekte zur Laufzeit erweitern**

- Alle Objekte können zur Laufzeit modifiziert werden
- Nachdem Klassen auch Objekte sind, können sie auch zur Laufzeit modifiziert werden
- Modifikationen können direkt im Code gemacht werden, oder aus Dateien geladen werden
- Programme können sich also zur Laufzeit selbst modifizieren

```
class TestClass
  def func
    puts "foo"
  end
end

x = TestClass.new
x.func
i = 3
```

*Ruby - Objektorientiertes Scripting, aber richtig!*

0-19

```
if i == 3 then
  class TestClass
    def func
      puts "bar"
    end
  end
end
end
x.func
```

*Ruby - Objektorientiertes Scripting, aber richtig!*

0-20

## **Sprachverwandtschaften**

- Ruby und Perl
- Ruby und Smalltalk

## **Ruby und Perl**

- Perl war Vorlage bei der Entwicklung von Ruby
- Viele Dinge in Ruby wurden aus Perl übernommen, z.B.
  - Perl-kompatible Regular Expressions
  - Die implizite `$_` Variable
  - sehr schnell erlernbar

## **Ruby und Smalltalk**

- Smalltalk ist eine der ersten objektorientierten Programmiersprachen
- Folgende Features wurden aus Smalltalk übernommen:
  - Alles ist ein Objekt
  - Anonyme Codeblöcke
  - Iteratoren
  - Late Binding

## **Wofür Ruby verwendet wird**

- Prototyping
- Textverarbeitung
- Künstliche Intelligenz und “Machine Learning”
- Mathematische Berechnungen
- ...



## **Wofür Ruby nicht geeignet ist**

- Echtzeitanwendungen
- wenn hohe Ausführungsgeschwindigkeit gefragt ist
- Steuerungen für wichtige Systeme (Flugzeuge, Raumfahrt, ...)
- ...

## **Wo kann ich mehr zu Ruby erfahren?**

- <http://www.ruby-lang.org/en/>
- <http://www.pragmaticprogrammer.com/ruby/>
- <http://www.rubynet.org/>
- <http://www.rubydoc.org/>
- <http://www.rubygarden.org/>
- ...

**Endlich! die letzte Folie!**

Noch Fragen?