Skriptsprachen

Moderne, objekt-orientierte
Skriptsprachen mit Betonung auf Ruby

Teil 1: Ruby im Überblick

Ein erstes Kennenlernen

Was ist Ruby?

- Eine Skriptsprache
  - Variablen werden implizit angelegt, nicht deklariert
  - Interpreter statt Compiler, kein Objekt-Code, *Executable*=Quellcode
  - Automatische Speicherverwaltung

- Eine der modernsten OO-Sprachen
  - In Ruby ist (fast) alles ein Objekt!
  - Sehr „sauberer“ Sprach-Entwurf
  - „Das Beste“ aus diversen Vorläufersprachen/Vorbildern

- Eine Hochsprache
  - Entwickler arbeiten problem-orientiert, nicht system-orientiert
  - Umfangreicher Bestand eingebauter Klassen & Methoden
  - Hohe Produktivität

Was ist Ruby?

- Eine *general purpose*-Sprache
  - Nicht beschränkt auf z.B. Automatisierung (Shell) oder Web-Entwicklung (PHP), sondern für fast alle Aufgaben geeignet
  - Auch für größere Projekte geeignet

- Eine Multi-Paradigma-Sprache:
  - Im Kern rein objekt-orientiert,
  - aber auch mit Elementen prozeduraler, funktionaler und deklarativer Sprachen angereichert
  - Entwickler können daher ihren „Stil“ wählen

- Eine erweiterbare Sprache
  - Ruby basiert auf C-Code. C-Bibliotheken können leicht von Ruby eingebunden werden
  - C-Programme können Funktionen der Ruby-Bibliothek verwenden.

Was ist Ruby?

- Eine dynamische Sprache
  - Code kann zur Laufzeit ergänzt und verändert werden
  - Auch die Standardbibliotheken lassen sich zur Laufzeit ändern
  - Jedes Objekt gibt Auskunft über sich selbst: Welche Methoden werden unterstützt, zu welcher Klasse gehört es?
  - Dynamischer Methodenaufruf: Die zuständige Methode wird erst zur Laufzeit ermittelt.

- Ein Konkurrent am Markt der Programmiersprachen
  - Ruby konkurriert mit Python („das bessere Sprach-Design“?)
  - Ruby verdrängt zunehmend Perl und PHP
  - Ruby lockt Entwickler von Compilersprachen wie Java, C++, C# wegen der hohen Produktivität
  - **Ruby on Rails** ist das zur Zeit produktivste Framework zur Entwicklung Web-basierter Anwendungen. Entwickler lernen Ruby, um Rails nutzen zu können.

Was ist Ruby?

- Ein Entwickler-Traum
  - Der Urheber Yukihiro Matsumoto („Matz“) realisierte 1993 seine „perfekte“ Sprache durch geschickte Kombination erfolgreicher Eigenschaften von Smalltalk, Perl, Eiffel, Python, CLU

    `Demo: Computer Languages History`

  - Mit Ruby entsteht kürzerer und besserer Code in weniger Zeit
  - Mit Ruby macht das Programmieren (wieder) Spaß!

Erste Schritte

Das „klassische“ erste Beispiel

- 5 Zeilen in „C“:

```
#include <stdio.h>
int main( int argc, char **argv ) {
     puts( "Hello, world!" );
     return( 0 );
}
```

- 1 Zeile in Ruby:

```
puts "Hello, world!"
```

```
→  "Hello, world!" an stdout
```

Reduktion auf das Wesentliche!

- **Fragen an Ruby:**
  - Wie startet man so einen Einzeiler?
  - Was ist denn daran objekt-orientiert??

Ruby-Interpreter: 3 Startoptionen

1. Quelldatei erzeugen & ausführen

   a) "hello.rb" mit Editor anlegen,

```
unix%> ruby hello.rb          # Explizit
```

   - oder -

   b) "#!"-Startzeile einfügen, Datei ausführbar machen, direkt aufrufen:

```
unix%> cat hello.rb
#!/usr/bin/env ruby
puts "Hello, world!"
unix%> chmod +x hello.rb
unix%> hello.rb
Hello, world!
```

TIPP:
Bei Windows unnötig, da Ruby per Assoziation mit Ext. ".rb" gestartet wird:

```
C:\temp> hello.rb
Hello, world!
```

Ruby-Interpreter: 3 Startoptionen

2. Per Kommandozeile und execute-Option:

Unix/Linux:

```
unix%> ruby -e "puts \"Hello, world\!\""
Hello, world!
unix%> ruby -e 'puts "Hello, world!"'
Hello, world!
```

Windows:

```
c:\temp> ruby -e "puts \"Hello, world!\""
Hello, world!
c:\temp> ruby -e "puts 'Hello, world!'"
Hello, world!
```

- Mehrzeiler möglich: `ruby -e "..." -e "..."`
- Beliebt für *ad hoc*-Kommandos!

Ruby-Interpreter: 3 Startoptionen

Vorsicht bei Interpretation von Sonderzeichen:

```
unix%> ruby -e "puts \"My\t world\""
My       world
unix%> ruby -e 'puts "My\t world"'
My       world
unix%> ruby -e "puts 'My\t world'"
My\t world
```

```
c:\temp> ruby -e 'puts "My\t world"'
My       world
c:\temp> ruby -e "puts 'My\t world'"
My\t world
```

Analogie zum Verhalten der Unix-Shell !

Ruby-Interpreter: 3 Startoptionen

3. Mit "Interactive Ruby" (irb):

```
unix%> irb
irb(main):001:0> puts "Hello, world!"
Hello, world!
==> nil
irb(main):002:0> exit
unix%>
```

Erwarteter Output

Rückgabewert des Ausdrucks "puts ..."!

```
c:\temp> irb
irb(main):001:0> puts "Hello, world!"
Hello, world!
==> nil
irb(main):002:0> exit
c:\temp>
```

Ganz analog zu Unix!

DAS soll objekt-orientiert sein?

- Ja!
  - Die OO ist implizit vorhanden
  - Sie wird hier nicht aufgezwungen
  - Ruby ist kein "Prinzipienreiter", sondern verabreicht den Entwicklern viel "*syntactic sugar*" - wie etwa hier.

- Ausführlicher:

```
$stdout.puts( "Hello, world!" );# Ausführlich ...
$stdout.puts "Hello, world!"    # oder "versüßt"
```

- Erläuterungen:
  - **`$stdout`** ist ein (vordefiniertes) Objekt der eingebauten Klasse "IO"
  - **`puts()`** ist eine Methode dieses Objekts
  - Notation: Präfix „**$**“ kennzeichnet globale Variablen

OO in Ruby

- Wenn Objekte „einfach so“, also ohne Deklaration, erscheinen können - wie erfahre ich denn, zu welcher Klasse ein Objekt zählt?

- Allgemeiner:
  Kann man die Klasse eines Objekts zur Laufzeit ermitteln?
  - Natürlich!

```
puts $stdout.class
→ IO          # Objekt "$stdout" zu Klasse "IO"
```

  - Objekte geben über sich selbst Auskunft!
  - Dazu stellt Ruby eine Reihe von Methoden bereit, die alle Objekte besitzen.
  - Realisiert als Methoden der Klasse "Object".

```
"Object" ist gemeinsame Oberklasse aller Klassen.
```

Objekte in Ruby

- Ist wirklich alles in Ruby ein Objekt?
  - Wie steht‘s denn mit Strings und Zahlen?
- Probieren wir es aus, z.B. mit **`irb`**:
  - Strings:

```
"abc".class                    → String
'A'.class                      → String
```

  - Zahlen:

```
42.class                       → Fixnum
3.14159.class                  → Float
1234567890123.class            → Bignum
```

- Und Programmcode?
  - Hier ist Ruby weniger radikal als etwa Smalltalk: Programmcode ist i.a. kein Objekt - es gibt aber die Klasse "Proc":

```
p = Proc.new {|name| puts "Hello, #{name}!"}
p.class                        → Proc
p.call "Dave"                  → Hello, Dave!
```

Objekte: Begriffsbildung

- Ein Wort zur Begriffsbildung
  - Ein Objekt ist ein Exemplar einer Klasse
  - Der Begriff "Instanz" ist ein Übersetzungsfehler - meiden!

    engl. "*instance*" = Exemplar, Beispiel

    "Script languages are cool. Take Ruby for instance!"

    Das deutsche Wort "Instanz" wird in anderem Zusammenhang verwendet, etwa: "juristische Instanzen" ("Landgericht", "Oberlandesgericht", ...)

Python-Interpreter: Startoptionen

1. Quelldatei erzeugen & ausführen

   a) "hello.py" mit Editor anlegen,

```
unix%> python hello.py          # Explizit
```

   - oder -

   b) "#!"-Startzeile einfügen, Datei ausführbar machen, direkt aufrufen:

```
unix%> cat hello.py
#!/usr/bin/env python
print "Hello, world!"
unix%> chmod +x hello.py
unix%> hello.py
Hello, world!
```

TIPP:
Bei Windows unnötig, da Ruby per Assoziation mit Ext. ".py" gestartet wird:

```
C:\temp> hello.py
Hello, world!
```

Python-Interpreter: Startoptionen

2. Per Kommandozeile und execute-Option:

Unix/Linux:

```
unix%> python -c "puts \"Hello, world\!\""
Hello, world!
unix%> python -c 'puts "Hello, world!"'
Hello, world!
```

Windows:

```
c:\temp> python -c "puts \"Hello, world!\""
Hello, world!
c:\temp> python -c "puts 'Hello, world!'"
Hello, world!
```

- Option –c nur einmal möglich: `python -c "..."`
- Beliebt für *ad hoc*-Kommandos!

Python-Interpreter: Startoptionen

3. Interactiver Python-Modus:

```
unix%> python
Python 2.5.1 (r251:54863, May 18 2007, 16:56:43)
[GCC 3.4.4 (cygming special, gdc 0.12, using dmd
   0.125)] on cygwin
Type "help", "copyright", "credits" or "license"
   for more information.
>>> print "Hello, world!"
Hello, world!
>>> exit()
unix%>
```