

Ruby





Ruby

A Programmer's Best Friend

Agenda

- ♥ (1) Einleitung
- ♥ (2) Konventionen && Grundlagen
- ♥ (3) Iteratoren && Container
- ♥ (4) Klassen && Vererbung && Objekte
- ♥ (5) Module && Mixins
- ♥ (6) Threads && Synchronisation
- ♥ (7) Fazit
- ♥ (8) Quellen



Ruby

A Programmer's Best Friend

Einleitung

- ♥ Erfinder: Yukihiro Matsumoto (Matz)
- ♥ Entwicklungsbeginn: 14.02.1993
- ♥ Erscheinungsjahr: 1995 in Japan
- ♥ Popularität in Europa Ende der 90er
- ♥ Es wurde auf Linux in C entwickelt
- ♥ **Voll** interpretierte **objektorientierte** Programmiersprache
- ♥ Folgt dem Prinzip der geringstmöglichen Überraschung



- ♥ Was für ein Mensch ist Matz?
- ♥ Warum wurde Ruby ins Leben gerufen?
- ♥ OpenSource(GPL) oder Kommerziell?
- ♥ Im welchen Bereich ist Matz heute aktiv?
- ♥ Warum war Ruby eine lange Zeit nur Japan vorbehalten?



Variablen

Lokale, Methodennamen Meth.-Parameter	Globale	Instanz-	Klassen-	Konstanten, Klassennamen, Modulnamen
name	\$hallo	@name	@@sum	NAME
test123	\$TEST	@test123	@@test	Test123
blablup	\$_	@blablup	@@lalelu	Blablup
flub_ber	\$X42	@flub_ber	@@pu_bi	Flub_ber
_x11	\$ff_x11	@_x11	@@_1x1	Xorg_x11
_321	\$_321	@_321	@@3_2a	AAA_321



Kommentareinleitung:

#

Block:

{...} oder do ... end

Methodendefinition:

def ... end

Bedingungen:

if ... elsif ... else ... end

Mehrwegbedingung:

case ... when ... else ... end

Schleifen:

while ... end,
until ... end,
for ... in ... do
loop Block



Boolsche Ausdrücke:

true, false

Boolsche Operatoren:

and, &&, or, ||, not, !,
defined?

Operatoren:

==, ===, <=>, <, <=, >=,
>, =~, eql?, equal?



♥ Methodendefinition

```
def sayHallo(name) #Typ A
  result = "Hallo " + name
  return result
end
```

```
def _SayHallo(name) #Typ B
  result = "Hallo #{name}"
  return result
end
```

```
def _sayHallo(name) #Typ C
  "Hallo #{name}"
end
```



♥ Bedingungen

```
if $counter < 101                                     #Typ A
  puts "mach weiter"
elsif city == "FuWa"
  puts "nichts wie weg hier!"
else
  puts "Hallo"
end
```

```
info = if counter < 8                                 #Typ B
  "zu wenig"
elsif city == "FuWa"
  "durchhalten bald ist es vorbei!"
else
  "endlich fertig und aus FuWa weg!"
end
```



♥ Bedingungen

```
$Status = if person.valid? ? true : false
```

```
@info = if person.alter >= 18 ? 'J' : 'N'
```

```
unless counter > 101  
  puts "ich will mehr!"  
end
```

```
puts "Jippi"      if counter >= 100  
  
puts "#{sayHallo('HFU')} " if uni == "HFU"  
  
print $Status unless show_status != true
```



♥ Mehrfach-Bedingungen

```
case year
  when 1890..1899 then "Ragtime"
  when /^[A-Z]*$/ then "not ok"
  when 1930...1889 then "Swing"
  when /^[^A-Z]*.*$/ then "ok"
  when 1990 then "Techno"
  else "Metal rockt!"
end
```

```
@@music_style = case MUSIC_TYPE
  when quiet then "Blues"
  when load then "Metal"
end
```



Schleifen

```
while @@i != $you
  printf "num: %12.4f, uni: %s", 123.45, "HFU"
  break
end
```

```
until num >= 123
  next if num == 5
  num += 1
  print num + "\n"
  redo if @repeat == true
end
```

```
result *= 2 while counter < 10

result += 4 until counter < 33
```



♥ Schleifen::Iteratoren

```
for val in ['execute', 'student', 'braindump'] do
  print val, "--"
end
```

```
for val in 1..100 do
  print val, " "
  retry if gets =~ /^yes/val
end
```

```
for val in objectXYT do          #sinnvolle Methode
  print val, "\n"              #"each" in der Klasse
end                              #definiert!!!
```

```
for val in File.open(Xfile).find_all do
  print val.chomp, "\n"
end
```



♥ Schleifen::Iteratoren

```
loop{  
  break if counter == 14  
  counter += 1  
}
```

```
loop do  
  break if counter == 14  
  counter += 1  
end
```



♥ Schleifen::Iteratoren

```
10.times do
  puts "hello world"
end
```

```
10.times{
  puts "hello world"
}
```

```
0.upto(10) do |val|
  print val
end
```

```
2.step(22,4) { |val| print val }
```

```
100.downto(-5) { |val| puts val }
```



♥ Einfacher Aufruf

```
def call_block_3_times
  yield
  yield
  yield
end

call_block_3_times { puts "Hallo" }
```

```
RUBY> Hallo
RUBY> Hallo
RUBY> Hallo
```



♥ Einfacher Aufruf mit Argument

```
def call_block_3_times
  yield "The Answer is"
  yield 42
  yield (">>RTFM<<")
end

call_block_3_times { |val| puts val }
```

```
RUBY> The Answer is
RUBY> 42
RUBY> >>RTFM<<
RUBY>
```



♥ Methoden und das Proc-Object

```
def do_something(param_1, &block)
  ...
  block.call("First Round")
  block.call("Middle Round")
  block.call("Last Round")
  ...
end

do_something { |val| puts val }
```

#letzter Param. &
#Codeblock wird in
#Proc-Object
#umgewandelt und
#dem Parameter
#zugewiesen

```
RUBY> First Round
RUBY> Middle Round
RUBY> Last Round
RUBY>
```



♥ Der kleine Held::Proc

```
def xMal(input)
  return proc { |val| input * val } #Referenz
end                                #zurückgeben

ptr = xMal(2)
puts ptr.call(5)

ptr = xMal("Hallo ")
print ptr.call(3), "\n"
```

```
RUBY> 10
RUBY> Hallo Hallo Hallo
RUBY>
```



Ruby

A Programmer's Best Friend

Container::Array

- ♥ Speichert eine Sammlung von Objektreferenzen
- ♥ Wächst dynamisch bei Bedarf an
- ♥ Besitzt eine große Anzahl **mächtiger** Methoden
- ♥ Umcodierungen z.B. ASCII nach Base64, ...
- ♥ Sehr leicht verständlich
- ♥ Problemlose Transformation in z.B Hash möglich



♥ Erstellung && Initialisierung

```
myArray = [] #Leeres Array-Literal

myArray = Array.new #entspricht []

myArray = Array.new(2) #[nil, nil]

myArray = Array.new(5, "A") #["A", "A", "A", "A", "A"]

myArray = Array.new(2, Hash.new) #[{}, {}]

myArray = [ 1, "hallo", -12.3 , 'wuf wuf' ]

myArray = %w{ one two three four five }
```



♥ Indizierung

Indizes:	0	1	2	3	4	5	6
	-7	-6	-5	-4	-3	-2	-1
<code>a =</code>	<code>"tux"</code>	<code>"bsd"</code>	<code>"cat"</code>	<code>"dig"</code>	<code>"ps"</code>	<code>"mv"</code>	<code>"gnu"</code>
<code>a[2, 4]</code>			<code>"cat"</code>	<code>"dig"</code>	<code>"ps"</code>		
<code>a[-3]</code>					<code>"ps"</code>		
<code>a[1..3]</code>		<code>"bsd"</code>	<code>"cat"</code>	<code>"dig"</code>			
<code>a[-3..-1]</code>					<code>"ps"</code>	<code>"mv"</code>	<code>"gnu"</code>
<code>a[4..-2]</code>					<code>"ps"</code>	<code>"mv"</code>	
<code>a[-8]</code>	<code>nil</code>						



♥ Zuweisung

```
myArray = [0, "gaga", "*lol*", -1.0125]
```

```
myArray[1] = [1, 2]
```

→ [0, [1, 2], "*lol*", -1.0125]

```
myArray[6] = 666
```

→ [0, [1, 2], "*lol*", -1.0125, nil, nil, 666]

```
puts myArray[1]
```

```
RUBY> 1  
RUBY> 2  
RUBY>
```



Ruby

A Programmer's Best Friend

Container::Hash

- ♥ Speichert eine Sammlung von Objektreferenzen
- ♥ Vorteil gegenüber Array:
Indizes beliebigen Objekt-Typs und nicht nur ganze Zahlen!!!
- ♥ Nachteil gegenüber Array:
Elemente sind **nicht geordnet**



Ruby

A Programmer's Best Friend

Container::Hash

♥ Erstellung && Initialisierung && Zuweisung

```
myHash = {} #Leeres Hash-Literal
```

```
myHash = Hash.new #entspricht {}
```

```
myHash = Hash["one", 1, "two", 2]
```

➔ {"one" => 1, "two" => 2}

```
myHash = Hash.new("Cartman") #Standardwert
```

```
myHash["Kenny"] = "McCormick"
```

```
puts myHash["Eric"]
```

```
print myHash
```

```
RUBY> Cartman
```

```
RUBY> KennyMcCormick
```



- ♥ Elternklasse aller Klassen ist “Object”
- ♥ Die Klasse “Object” mischt das Modul “Kernel” mit ein, später dazu mehr...
- ♥ Klassen sind einfach zu definieren und nie geschlossen! → (jederzeit editierbar)
- ♥ Eine Fülle an Standard-Methoden vorhanden



Definition

```
class Song_A
  def initialize(name, artist, duration)
    @name      = name
    @artist    = artist
    @duration  = duration
  end
end
```

#Was ist der Unterschied?

```
class Song_B
  def initialize(name, artist, duration)
    @@name     = name
    @@artist   = artist
    @@duration = duration
  end
end
```



♥ Definition

```
class Song
  def initialize(name, artist, duration)
    @name      = name
    @artist    = artist
    @duration  = duration
  end

  def name
    @name
  end

  def name=(new_name)
    @name = new_name
  end
  # [...]
end
```



♥ Definition

```
class Song
  attr_reader :name, :artist, :duration
  attr_writer :name, :artist, :duration

  def initialize(name, artist, duration)
    @name      = name
    @artist    = artist
    @duration  = duration
  end
end

attr_reader } attr_accessor(:name, :artist, :duration)
attr_writer }
```



Zugriffskontrolle

```
class Visibility
  def public_method_1
    # [...]
  end

  protected
  def protected_method_1
    # [...]
  end

  private
  def private_method_1
    # [...]
  end
end
```

▶ nächste Folie



Zugriffskontrolle

```
public
  def public_method_2
    #[...]
  end
end
```

```
class Visibility
  #[...]

  public      :public_method_1, :public_method_2
  protected  :protected_method_1
  private    :private_method_1
end
```



♥ Es ist nur Einfachvererbung möglich

```
class SuperSong < Song
  attr_accessor(:annotation)

  def initialize(name, artist, duration, annotation)
    super(name, artist, duration)
    @annotation = annotation
  end
end
```

Aber Ruby ist mächtiger :-)
Jede Klasse kann die Funktionalität beliebig vieler
Mixins enthalten!!! (später dazu mehr)



♥ Erstellung && Itialisierung

```
mySong = Song.new("In Search For!", "In Flames", 202)

print "Name:      ", mySong.name, "\n"

print "Artist:    ", mySong.artist, "\n"

print "Duration:  ", mySong.duration seconds, "\n"


mySong.artist = "bang ;-)"

print "let's ", mySong.artist, "\n"
```


```
RUBY> Name:      In Flames
RUBY> Artist:    In Search For!
RUBY> Duration:  202 seconds
RUBY> let's bang ;-)
```



♥ Gegen Veränderung schützen

```
mySuperSong = SuperSong.new("lalelu", "?", 0, ":- (")  
mySuperSong.freeze  
mySuperSong.artist[3] = "A"  TypeError  
print "Frozen? --> ", mySuperSong.frozen?, "\n"
```

```
RUBY> Frozen? --> true  
RUBY>
```

Einmal eingefroren  **immer** eingefroren!!!



♥ Interne Informationen abrufen

#Welche Methoden sind public?

myObject.`public_methods`

#Welche Instanz-Methoden sind protected?

myObject.`protected_instance_methods`

#Welche Klassen-Methoden sind private?

myObject.`private_class_methods`

#Welche Werte haben meine Instanz-Variablen?

myObject.`inspect`

public
protected
private



- ♥ Eine Möglichkeit um:
Methoden, Klassen und Konstanten in eine Gruppe zusammenzufassen
- ♥ Besitzen einen **Namensraum** zur
Konfliktvermeidung von gleichen Namen
- ♥ Implementieren die **Mixin-Fähigkeit**
- ♥ Es sind keine Instanzen von Modulen möglich, da
sie **keine Klassen** sind!!!



♥ Definition

```
module Message
  SIZE = 8192

  def Message.send(msg)           #Modul-Methode
    puts "PUTS: " + msg
    print "PRINT: " + msg, "\n"
    printf "PRINTF: %s \n", msg
  end

  def send_2(msg)                 #Instanz-Methode
    print msg
  end

  #[...]
end
```

▶ nächste Folie

Modulmethode \equiv Klassenmethode



Definition

```
class Hallo
  def initialize(val)
    print "Hallo: ",val,"\n"
  end

  #[...]

end

#[...]

end
```



Ruby

A Programmer's Best Friend

Module::Mixins

♥ Die indirekte **n-fache Vererbung ohne Nachteile**

```
class Test
  include Message
  def initialize(val)
    @hallo = Hallo.new(val)
    Message.send("123")
  end
end
test = Test.new("students")
test.send_2(666) #Instanz-Methode von Modul "Message"
```

```
RUBY> Hallo: students
RUBY> PUTS: 123
RUBY> PRINT: 123
RUBY> PRINTF: 123
RUBY> 666
```



♥ Das Zauberstück: Interaktion mit dem Modul

```
class Test
  require "message" #Modul, externe Datei "message.rb"
  include Message   #Referenz auf das Modul
  def initialize(val)
  end
end
test = Test.new("students")
test.send_2(666)    #Instanz-Methode vom Modul
```

RUBY> 666

Bei Änderung von Modul-Methoden, wirkt sich dies **sofort** auf alle Klassen/Objekte aus, selbst **während der Laufzeit!!!**



♥ Make one get six

```
class Song
  include Comparable

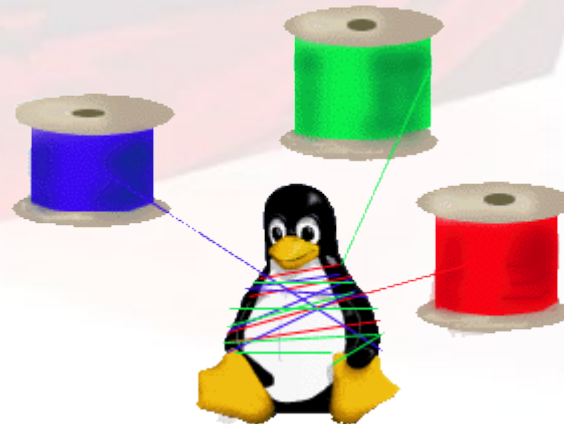
  def <=>(otherSong)
    self.duration <=> otherSong.duration
  end
end
```

Nach Definition der Methode ”<=>”, erhalten wir auto. vom Modul “Comparable” sechs weitere , ohne diese selbst definieren zu müssen, hinzu.

Gratis-Methoden: <, <=, ==, >=, > und **between?**



- ♥ Sind in Rubys Interpreter implementiert, dadurch vollständig portierbar und nicht vom Betriebssystem abhängig
- ♥ Teilen sich den gleichen Prozess-Adressraum





Erstellung und Ausführung

```
myThreads = []
counter   = 0

3.times{
  counter += 1
  myThreads << Thread.new(counter) do
    puts counter
  end
}

myThreads.each { |t| t.join }
```

```
RUBY> 1
RUBY> 2
RUBY> 3
RUBY>
```



Methoden

<code>Thread.join</code>	<code>#wartet auf Beendigung</code>
<code>Thread.list</code>	<code>#Array von Thread-Objekten</code>
<code>Thread.critical</code>	<code>#Globale Bedingung</code>
<code>Thread.alive?</code>	<code>#Wird ausgeführt?</code>
<code>Thread.priority=</code>	<code>#Setzen der Priorität</code>
<code>Thread.current</code>	<code>#Gibt aktuellen Thread</code>
<code>Thread.stop</code>	<code>#Schlafen setzen</code>
<code>[...]</code>	



♥ Klasse Mutex

```
require 'thread'
mutex = Mutex.new

counter_1 = counter_2 = 0

counter = Thread.new {
  2.times do
    mutex.synchronize {
      counter_1 += 1
      counter_2 += 1
    }
  }
end
}
```



♥ Klasse ConditionVariable

```
require 'thread'
mutex      = Mutex.new
cond_var   = ConditionVariable.new

t1 = Thread.new {
  mutex.synchronize do
    puts "t1: zentral, warte auf cond_var"
    cond_var.wait(mutex)
    puts "t1: habe alles was ich brauche"
  end
}
```

▶ nächste Folie



♥ Klasse ConditionVariable

```
t2 = Thread.new {
  mutex.synchronize do
    puts "t2: zentral und fertig"
    cond_var.signal
    puts "zentral und geich beendet"
  end
}

t1.join
t2.join
```

```
RUBY> t1: zentral, warte auf cond_var
RUBY> t2: zentral und fertig
RUBY> zentral und geich beendet
RUBY> t1: habe alles was ich brauche
RUBY>
```



Ruby

A Programmer's Best Friend

Fazit

- ♥ Ruby ist **pure objektorientierte** Programmiersprache
- ♥ Ruby besitzt eine **sehr saubere Syntax**
- ♥ Dank **Garbage Collector** keine Speicherlecks
- ♥ **Module** sind sauberer als multiple Inheritances
- ♥ **OpenSource**
- ♥ Viele Anwendungsgebiete möglich (HTTP,...)
- ♥ Integrierter **Debugger** erleichtert die Fehlersuche
- ♥ Code **1:1 portabel** auf sehr viele Betriebssysteme



Ruby

A Programmer's Best Friend

Quellen

- ♥ Programmieren mit Ruby [Addison-Wesley 2002, ISBN:3-8273-1965-X]
- ♥ <http://www.ruby-lang.org/>
- ♥ <http://www.rubycentral.com/>
- ♥ [http://de.wikipedia.org/wiki/Ruby_\(Programmiersprache\)](http://de.wikipedia.org/wiki/Ruby_(Programmiersprache))



