1.– 4. September 2014
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

The Joy of Programming

Eine Einführung in Ruby

Johannes Held

mail@johannesheld.net

Mathema Herbstcampus 2014
03.09.2014

Ruby

A dynamic, open source programming language with a focus on simplicity and productivity. It has an elegant syntax that is natural to read and easy to write.

`http://www.ruby-lang.org`

”I hope to see Ruby help every programmer in the world to be productive, and to enjoy programming, and to be happy. That is the primary purpose of Ruby language.”

–Yukihiro “Matz” Matsumoto

Ruby

A dynamic, open source programming language with a focus on simplicity and productivity. It has an elegant syntax that is natural to read and easy to write.

`http://www.ruby-lang.org`

”I hope to see Ruby help every programmer in the world to be productive, and to enjoy programming, and to be happy. That is the primary purpose of Ruby language.”

–Yukihiro “Matz” Matsumoto

Inhalt

Ruby

Yukihiro “Matz” Matsumoto

Anleihen an *Perl, Smalltalk, Eiffel, Ada, Lisp*

- 24.02.1993: v0.1
- 21.12.1995: v0.95 *veröffentlicht*
- August 2003: v1.8
  JIS X 3017, ISO/IEC 30170
- 2005 Ruby on Rails
- Dezember 2007: v1.9
  Ruby License & BSD-Licsense
- 24.02.2013: v2.0
  method keyword arguments, lazy enumeration
- aktuell v2.1.2

Ruby

Yukihiro “Matz” Matsumoto

Anleihen an *Perl, Smalltalk, Eiffel, Ada, Lisp*

- 24.02.1993: v0.1
- 21.12.1995: v0.95 *veröffentlicht*
- August 2003: v1.8
  JIS X 3017, ISO/IEC 30170
- 2005 Ruby on Rails
- Dezember 2007: v1.9
  Ruby License & BSD-Licsense
- 24.02.2013: v2.0
  method keyword arguments, lazy enumeration
- aktuell v2.1.2

Ruby

Yukihiro “Matz” Matsumoto

Anleihen an *Perl, Smalltalk, Eiffel, Ada, Lisp*

- 24.02.1993: v0.1
- 21.12.1995: v0.95 *veröffentlicht*
- August 2003: v1.8
  JIS X 3017, ISO/IEC 30170
- 2005 Ruby on Rails
- Dezember 2007: v1.9
  Ruby License & BSD-Licsense
- 24.02.2013: v2.0
  method keyword arguments, lazy enumeration
- aktuell v2.1.2

Ruby

Yukihiro “Matz” Matsumoto

Anleihen an *Perl, Smalltalk, Eiffel, Ada, Lisp*

- 24.02.1993: v0.1
- 21.12.1995: v0.95 *veröffentlicht*
- August 2003: v1.8
  JIS X 3017, ISO/IEC 30170
- 2005 Ruby on Rails
- Dezember 2007: v1.9
  Ruby License & BSD-Licsense
- 24.02.2013: v2.0
  method keyword arguments, lazy enumeration
- aktuell v2.1.2

Ruby

Yukihiro “Matz” Matsumoto

Anleihen an *Perl, Smalltalk, Eiffel, Ada, Lisp*

- 24.02.1993: v0.1
- 21.12.1995: v0.95 *veröffentlicht*
- August 2003: v1.8
  JIS X 3017, ISO/IEC 30170
- 2005 Ruby on Rails
- Dezember 2007: v1.9
  Ruby License & BSD-Licsense
- 24.02.2013: v2.0
  method keyword arguments, lazy enumeration
- aktuell v2.1.2

Ruby

Yukihiro “Matz” Matsumoto

Anleihen an *Perl, Smalltalk, Eiffel, Ada, Lisp*

- 24.02.1993: v0.1
- 21.12.1995: v0.95 *veröffentlicht*
- August 2003: v1.8
  JIS X 3017, ISO/IEC 30170
- 2005 Ruby on Rails
- Dezember 2007: v1.9
  Ruby License & BSD-Licsense
- 24.02.2013: v2.0
  method keyword arguments, lazy enumeration
- aktuell v2.1.2

Section 4

Core Features

Core Features

Alles ist ein Objekt

```ruby
3.times do
  puts 'Ruby is made for programmers!'
end
```

```ruby
# => Ruby is made for programmers!
# => Ruby is made for programmers!
# => Ruby is made for programmers!
```

Core Features

Alles ist ein Objekt

```ruby
3.times do
  puts 'Ruby is made for programmers!'
end
```

```ruby
# => Ruby is made for programmers!
# => Ruby is made for programmers!
# => Ruby is made for programmers!
```

Core Features

Blöcke

```ruby
File.open('file.txt', 'w') do |f|
  f.write 'write' # no \n
  f << '<<'       # no \n
  f.puts 'puts'   # \n
  f.puts 'puts'
end
```

```ruby
File.readlines('file.txt').each do |line|
  puts line
end
# => write<<puts
# => puts
```

Core Features

Blöcke

```ruby
File.open('file.txt', 'w') do |f|
  f.write 'write' # no \n
  f << '<<'       # no \n
  f.puts 'puts'   # \n
  f.puts 'puts'
end
```

```ruby
File.readlines('file.txt').each do |line|
  puts line
end
# => write<<puts
# => puts
```

Core Features

Blöcke

```ruby
search_engines =
  %w[Google Yahoo Bing].map do |engine|
    'http://www.' + engine.downcase + '.com'
  end
p search_engines
# ["http://www.google.com", "http://www.yahoo.com",
#  "http://www.bing.com"]
```

Core Features

Lazy Enumerators

```
range = 1..Float::INFINITY
range.map {|x| x + x}
     .select {|x| x % 3 == 0}
     .first(7)
```

```
range.lazy.map {|x| x + x}
          .select {|x| x % 3 == 0}
          .first(7)
# => [6, 12, 18, 24, 30, 36, 42]
```

Core Features

Lazy Enumerators

```
range = 1..Float::INFINITY
range.map {|x| x + x}
     .select {|x| x % 3 == 0}
     .first(7)
```

```
range.lazy.map {|x| x + x}
          .select {|x| x % 3 == 0}
          .first(7)
# => [6, 12, 18, 24, 30, 36, 42]
```

Core Features

Regular Expressions

```
1  r = /^http:\/\/(?:www.)?(?<host>.*)$/
2  r = %r[^http://(?:www.)?(?<host>.*)$]
3
4  puts 'http://ruby-lang.org'.match(r)[1]
5  puts 'http://ruby-lang.org'.match(r)[:host]
6  # => ruby-lang.org
```

Core Features

Alles ist im Fluss

Monkey Patch

```ruby
puts 5.days.ago
# => 2014-08-20 20:42:04 +0200
```

```ruby
class Fixnum
  def days
    self * 24 * 60 * 60
  end

  def ago
    Time.now - self
  end
end
```

besser: *gem timerizer*

Core Features

Alles ist im Fluss

Monkey Patch

```ruby
puts 5.days.ago
# => 2014-08-20 20:42:04 +0200
```

```ruby
class Fixnum
  def days
    self * 24 * 60 * 60
  end

  def ago
    Time.now - self
  end
end
```

besser: *gem timerizer*

Core Features

Externe Libraries: Gems

```
# $ gem install hulk
require 'hulk'
```

```
require './path/to/file.rb'
```

http://bundler.io

Core Features

Mixins

Einfachvererbung
Ein*mixen* von Funktionalität

```
class Foo
  include Enumerable

  def each
    # implement each here
  end
end

# each => map, select, drop_while, group_by, find_all …
```

```
p (1..10).group_by {|i| i % 3 == 0}
# {false => [1, 2, 4, 5, 7, 8, 10],
#  true => [3, 6, 9]}
```

Core Features

Mixins

Einfachvererbung
Ein*mixen* von Funktionalität

```
1  class Foo
2    include Enumerable
3
4    def each
5      # implement each here
6    end
7  end
8
9  # each => map, select, drop_while, group_by, find_all …
```

```
1  p (1..10).group_by {|i| i % 3 == 0}
2  # {false => [1, 2, 4, 5, 7, 8, 10],
3  #  true => [3, 6, 9]}
```

Core Features

Mixins

Einfachvererbung
Ein*mixen* von Funktionalität

```
class Foo
  include Enumerable

  def each
    # implement each here
  end
end

# each => map, select, drop_while, group_by, find_all …
```

```
p (1..10).group_by {|i| i % 3 == 0}
# {false => [1, 2, 4, 5, 7, 8, 10],
#  true => [3, 6, 9]}
```

method_missing

```ruby
class Finder
  def method_missing(meth, *args, &block)
    puts "#{meth} was called with these arguments:"
    puts args
    puts 'a block was given' if block_given?
  end
end
```

```ruby
f = Finder.new
f.find_person(name: 'Johannes') {|person| puts person }
# => find_person was called with these arguments:
# => {:name=>"Johannes"}
# => a block was given
```

method_missing

```ruby
class Finder
  def method_missing(meth, *args, &block)
    puts "#{meth} was called with these arguments:"
    puts args
    puts 'a block was given' if block_given?
  end
end
```

```ruby
f = Finder.new
f.find_person(name: 'Johannes') {|person| puts person }
# => find_person was called with these arguments:
# => {:name=>"Johannes"}
# => a block was given
```

Section 5

Basics

Basics

```ruby
zahl = 7
text = 'Ich bin ein String'
text_interpoliert = "#{zahl * 3} ist eine Zahl"
here_doc =<<EOH
Ich bin ein Here-Document. #{zahl}
EOH

puts here_doc

symbol = :mode

KONSTANTE = 'ich bin ein fester Wert'
class IchBinEinBezeichner; end
```

Ein Schluck %-iges

```
%i[ ] # Non-interpolated Array of symbols, separated by whitespace
%I[ ] # Interpolated Array of symbols, separated by whitespace
%w[ ] # Non-interpolated Array of words, separated by whitespace
%W[ ] # Interpolated Array of words, separated by whitespace
```

```
%q[ ] # Non-interpolated String (except for \\ \[ and \])
%Q[ ] # Interpolated String (default)
```

```
%r[ ] # Interpolated Regexp
      # ()flags can appear after the closing delimiter)
```

Vorsicht

```
%x[ ] # Interpolated shell command
```

Ein Schluck %-iges

```
%i[ ] # Non-interpolated Array of symbols, separated by whitespace
%I[ ] # Interpolated Array of symbols, separated by whitespace
%w[ ] # Non-interpolated Array of words, separated by whitespace
%W[ ] # Interpolated Array of words, separated by whitespace
```

```
%q[ ] # Non-interpolated String (except for \\ \[ and \])
%Q[ ] # Interpolated String (default)
```

```
%r[ ] # Interpolated Regexp
      # ()flags can appear after the closing delimiter)
```

Vorsicht

```
%x[ ] # Interpolated shell command
```

Ein Schluck %-iges

```
%i[ ] # Non-interpolated Array of symbols, separated by whitespace
%I[ ] # Interpolated Array of symbols, separated by whitespace
%w[ ] # Non-interpolated Array of words, separated by whitespace
%W[ ] # Interpolated Array of words, separated by whitespace
```

```
%q[ ] # Non-interpolated String (except for \\ \[ and \])
%Q[ ] # Interpolated String (default)
```

```
%r[ ] # Interpolated Regexp
      # ()flags can appear after the closing delimiter)
```

Vorsicht

```
%x[ ] # Interpolated shell command
```

Ein Schluck %-iges

```
%i[ ] # Non-interpolated Array of symbols, separated by whitespace
%I[ ] # Interpolated Array of symbols, separated by whitespace
%w[ ] # Non-interpolated Array of words, separated by whitespace
%W[ ] # Interpolated Array of words, separated by whitespace
```

```
%q[ ] # Non-interpolated String (except for \\ \[ and \])
%Q[ ] # Interpolated String (default)
```

```
%r[ ] # Interpolated Regexp
      # ()flags can appear after the closing delimiter)
```

Vorsicht

```
%x[ ] # Interpolated shell command
```

Basics

Besondere Zuweisungen

```
a, b = 1, 2, 3
# a = 1   b = 2
```

```
a, b, c = 1, [2, 3]
# a = 1   b = [2, 3]    c = nil
```

```
a, b, c = 1, *[2, 3]
# a = 1   b = 2   c = 3
```

```
a, *b = 1, 2, 3
# a = 1   b = [2, 3]
```

Basics

Besondere Zuweisungen

```
a, b = 1, 2, 3
# a = 1   b = 2
```

```
a, b, c = 1, [2, 3]
# a = 1   b = [2, 3]    c = nil
```

```
a, b, c = 1, *[2, 3]
# a = 1   b = 2   c = 3
```

```
a, *b = 1, 2, 3
# a = 1   b = [2, 3]
```

Basics

Besondere Zuweisungen

```
a, b = 1, 2, 3
# a = 1   b = 2
```

```
a, b, c = 1, [2, 3]
# a = 1   b = [2, 3]    c = nil
```

```
a, b, c = 1, *[2, 3]
# a = 1   b = 2   c = 3
```

```
a, *b = 1, 2, 3
# a = 1   b = [2, 3]
```

Basics

Besondere Zuweisungen

```
a, b = 1, 2, 3
# a = 1   b = 2
```

```
a, b, c = 1, [2, 3]
# a = 1   b = [2, 3]    c = nil
```

```
a, b, c = 1, *[2, 3]
# a = 1   b = 2   c = 3
```

```
a, *b = 1, 2, 3
# a = 1   b = [2, 3]
```

Basics

Splatter!

```ruby
a = [1,2]
b = %w(drei vier)
c = %w(5 6)

cross = a.product b, c
cross.each { |t| p t }
# => [1, "drei", "5"]
# [1, "drei", "6"]
# [1, "vier", "5"]
# [1, "vier", "6"]
# [2, "drei", "5"]
# [2, "drei", "6"]
# [2, "vier", "5"]
# [2, "vier", "6"]
```

```ruby
dims = [a, b, c]
head, *tail = *dims
p head
# => [1, 2]
p tail
# => [["drei", "vier"],
#     ["5", "6"]]

cross = head.product *tail
cross.each { |t| p t }
# siehe links :)
```

```
combination
permutation
```

Basics

Splatter!

```
a = [1,2]
b = %w(drei vier)
c = %w(5 6)

cross = a.product b, c
cross.each { |t| p t }
# => [1, "drei", "5"]
# [1, "drei", "6"]
# [1, "vier", "5"]
# [1, "vier", "6"]
# [2, "drei", "5"]
# [2, "drei", "6"]
# [2, "vier", "5"]
# [2, "vier", "6"]
```

```
dims = [a, b, c]
head, *tail = *dims
p head
# => [1, 2]
p tail
# => [["drei", "vier"],
#     ["5", "6"]]

cross = head.product *tail
cross.each { |t| p t }
# siehe links :)
```

```
combination
permutation
```

Basics

Splatter!

```
a = [1,2]
b = %w(drei vier)
c = %w(5 6)

cross = a.product b, c
cross.each { |t| p t }
# => [1, "drei", "5"]
# [1, "drei", "6"]
# [1, "vier", "5"]
# [1, "vier", "6"]
# [2, "drei", "5"]
# [2, "drei", "6"]
# [2, "vier", "5"]
# [2, "vier", "6"]
```

```
dims = [a, b, c]
head, *tail = *dims
p head
# => [1, 2]
p tail
# => [["drei", "vier"],
#     ["5", "6"]]

cross = head.product *tail
cross.each { |t| p t }
# siehe links :)
```

```
combination
permutation
```

if

```ruby
if a == :foo
  # code
elsif a == :bar && b == '7'
  # code
else
  # code
end
```

```ruby
h = if 0
      'true'
    else
      'false'
    end
puts h
# => true
```

Vorsicht!

nil und *false* werden zu *false* ausgewertet.
0 wird zu *true* ausgewertet.

if

```ruby
if a == :foo
  # code
elsif a == :bar && b == '7'
  # code
else
  # code
end
```

```ruby
h = if 0
      'true'
    else
      'false'
    end
puts h
# => true
```

Vorsicht!

nil und *false* werden zu *false* ausgewertet.
0 wird zu *true* ausgewertet.

if

```ruby
if a == :foo
  # code
elsif a == :bar && b == '7'
  # code
else
  # code
end
```

```ruby
h = if 0
      'true'
    else
      'false'
    end
puts h
# => true
```

Vorsicht!

nil und *false* werden zu *false* ausgewertet.
0 wird zu *true* ausgewertet.

unless

```ruby
a = nil
unless a
  puts 'piep'
end
# => piep
```

```ruby
if (a.nil? || a == false)
  puts 'piep'
end

# if not a
# if ! a
```

unless

```ruby
a = nil
unless a
  puts 'piep'
end
# => piep
```

```ruby
if (a.nil? || a == false)
  puts 'piep'
end

# if not a
# if ! a
```

case

```ruby
a = 7

case a
  when 'sieben'
    # code
  when 1..8 then #code
  when [4, 7]
    # code
end
```

```ruby
a = 35
h = case a
      when 4
        'vier'
      when 30..35
        '[30, 35]'
        'neun'
      else
        18
    end
puts h
# => neun
```

case

```ruby
a = 7

case a
  when 'sieben'
    # code
  when 1..8 then #code
  when [4, 7]
    # code
end
```

```ruby
a = 35
h = case a
      when 4
        'vier'
      when 30..35
        '[30, 35]'
        'neun'
      else
        18
    end
puts h
# => neun
```

and & or

Kontrollfluss steuern

```ruby
raise 'STDIN ist leer' unless line = $stdin.gets
```

```ruby
line = $stdin.gets || raise 'STDIN ist leer'
# syntax error, unexpected tSTRING_BEG, expecting keyword_do or '{' or '(
# line = $stdin.gets || raise 'STDIN ist leer'
#                          ^
```

```ruby
line = $stdin.gets or raise 'STDIN ist leer'
```

Vorsicht!

```ruby
a = ( :x or :y and nil )
p a
# nil
```

```ruby
a = ( (:x or :y) and nil )
p a
#  nil
```

Avid Grimm devblog.avdi.org/2014/08/26/how-to-use-rubys-english-andor-operators-without-going-nuts/

and & or

Kontrollfluss steuern

```ruby
raise 'STDIN ist leer' unless line = $stdin.gets
```

```ruby
line = $stdin.gets || raise 'STDIN ist leer'
# syntax error, unexpected tSTRING_BEG, expecting keyword_do or '{' or '('
# line = $stdin.gets || raise 'STDIN ist leer'
#                           ^
```

```ruby
line = $stdin.gets or raise 'STDIN ist leer'
```

Vorsicht!

```ruby
a = ( :x or :y and nil )
p a
# nil
```

```ruby
a = ( (:x or :y) and nil )
p a
# nil
```

Avid Grimm devblog.avdi.org/2014/08/26/how-to-use-rubys-english-andor-operators-without-going-nuts/

and & or

Kontrollfluss steuern

```
raise 'STDIN ist leer' unless line = $stdin.gets
```

```
line = $stdin.gets || raise 'STDIN ist leer'
# syntax error, unexpected tSTRING_BEG, expecting keyword_do or '{' or '(
# line = $stdin.gets || raise 'STDIN ist leer'
#                           ^
```

```
line = $stdin.gets or raise 'STDIN ist leer'
```

Vorsicht!

```
a = ( :x or :y and nil )
p a
# nil
```

```
a = ( (:x or :y) and nil )
p a
#  nil
```

Avid Grimm devblog.avdi.org/2014/08/26/how-to-use-rubys-english-andor-operators-without-going-nuts/

Schleifen

```ruby
while condition
  # code
end

code while condition
```

```ruby
until condition
  # code
end

code until condition
```

```ruby
for var, var1 in expression
  code
end
```

`break` terminate loop

`next` jump to next iteration

`redo` repeats current iteration

Schleifen

```ruby
while condition
  # code
end

code while condition
```

```ruby
until condition
  # code
end

code until condition
```

```ruby
for var, var1 in expression
  code
end
```

`break` terminate loop

`next` jump to next iteration

`redo` repeats current iteration

Schleifen

```ruby
while condition
  # code
end

code while condition
```

```ruby
until condition
  # code
end

code until condition
```

```ruby
for var, var1 in expression
  code
end
```

`break` terminate loop

`next` jump to next iteration

`redo` repeats current iteration

Schleifen

```ruby
while condition
  # code
end

code while condition
```

```ruby
until condition
  # code
end

code until condition
```

```ruby
for var, var1 in expression
  code
end
```

`break` terminate loop

`next` jump to next iteration

`redo` repeats current iteration

Strings

```ruby
s = 'hallo'
s[2..-1] = 'se'
puts s
# => hase
s['a'] = 'o'
puts s
# => hose
s[/e$/] = 'i'
puts s
# => hosi

puts 'drin' if s =~ /^h/
# => drin
```

Basics

Methodendefinition

```ruby
def bmi(weight, height)
  weight / height ** 2 # implicit return
end

bmi 53.4, 1.62
```

```ruby
def bmi(weight:, height:)
  weight / height ** 2
end

bmi height: 1.62, weight: 53.3
```

Basics

Methodendefinition

```ruby
def bmi(weight, height)
  weight / height ** 2 # implicit return
end

bmi 53.4, 1.62
```

```ruby
def bmi(weight:, height:)
  weight / height ** 2
end

bmi height: 1.62, weight: 53.3
```

Basics

Methodendefinition - Default Argumente

```ruby
def foo(argument = 'guten ')
  print argument
end

def bar(keyword_argument: 'Tag')
  puts keyword_argument
end

foo
bar
# => guten Tag
```

Basics

Methodendefinition - Splat

```ruby
def foo(a, *args, opts)
  p a
  p args
  p opts
end

foo 7, 1, 2, 3, :force => true, :verbose => false
# 7
# [1, 2, 3]
# {:force=>true, :verbose=>false}
```

Basics

Methoden - Namenskonventionen

```ruby
def i_change_self!
end

def is_hash?
end
```

```ruby
a = [3, 2, 1]
b = a.sort
p b
# [1, 2, 3]
p a
# [3, 2, 1]

a.sort!
p a
# [1, 2, 3]
```

Basics

Methoden - Namenskonventionen

```ruby
def i_change_self!
end

def is_hash?
end
```

```ruby
a = [3, 2, 1]
b = a.sort
p b
# [1, 2, 3]
p a
# [3, 2, 1]

a.sort!
p a
# [1, 2, 3]
```

Array

```ruby
liste = ['ich', 7]
# ['ich', 7]
liste << 'horst'

p liste.include?('horst')
# true
```

```ruby
[[1,2,3], %w(vier fünf)]
```

Array

```ruby
liste = ['ich', 7]
# ['ich', 7]
liste << 'horst'

p liste.include?('horst')
# true
```

```ruby
[[1,2,3], %w(vier fünf)]
```

Array

Defaultwerte

```ruby
liste = Array.new(3, [])
p liste
# [[], [], []]

liste.first << :test
p liste
# [[:test], [:test], [:test]]

liste = Array.new(3) {|idx| idx + 3}
p liste
# [3, 4, 5]

liste = Array.new(3) {|_| []}
```

Hash

```ruby
h = {:herbst => 'campus',
     zahl: 7,  # sugar for :zahl => 7
     'customer_id' => 7}
h[7] = 'sieben'
p h
# {:herbst=>"campus", :zahl=>7, "customer_id"=>7, 7=>"sieben"}
```

```ruby
data = [['a', 1], [:b, 2]]
h = Hash[data]
p h
# {"a"=>1, :b=>2}

data = ['a', 1, :b, 2]
h = Hash[*data]
p h
# {"a"=>1, :b=>2}
```

Hash

```ruby
h = {:herbst => 'campus',
     zahl: 7,  # sugar for :zahl => 7
     'customer_id' => 7}
h[7] = 'sieben'
p h
# {:herbst=>"campus", :zahl=>7, "customer_id"=>7, 7=>"sieben"}
```

```ruby
data = [['a', 1], [:b, 2]]
h = Hash[data]
p h
# {"a"=>1, :b=>2}

data = ['a', 1, :b, 2]
h = Hash[*data]
p h
# {"a"=>1, :b=>2}
```

Hash

Defaultwerte

```ruby
h = Hash.new('herbstcampus')
puts h[:mathema]
# => herbstcampus

h = Hash.new([])
h[:teilnehmer] << 'Franz'
p h[:organisatoren]
# ["Franz"]
```

```ruby
h = Hash.new do |hash, key|
  hash[key] = []
end
h[:teilnehmer] << 'Franz'
p h[:organisatoren]
# []
```

Hash

Defaultwerte

```ruby
h = Hash.new('herbstcampus')
puts h[:mathema]
# => herbstcampus

h = Hash.new([])
h[:teilnehmer] << 'Franz'
p h[:organisatoren]
# ["Franz"]
```

```ruby
h = Hash.new do |hash, key|
  hash[key] = []
end
h[:teilnehmer] << 'Franz'
p h[:organisatoren]
# []
```

Hash

Memoization

```ruby
def fib(idx)
  return 0 if idx == 0
  return 1 if idx == 1

  fib(idx - 1) + fib(idx - 2)
end

puts fib(7)
# => 13
```

```ruby
fibs = Hash.new do |hash, key|
  hash[key] = hash[key - 1] + hash[key - 2]
end.update({0 => 0, 1 => 1})

puts fibs[70]
# =>  190392490709135
```

Hash

Memoization

```ruby
def fib(idx)
  return 0 if idx == 0
  return 1 if idx == 1

  fib(idx - 1) + fib(idx - 2)
end

puts fib(7)
# => 13
```

```ruby
fibs = Hash.new do |hash, key|
  hash[key] = hash[key - 1] + hash[key - 2]
end.update({0 => 0, 1 => 1})

puts fibs[70]
# =>  190392490709135
```

Hash

Nested Defaultwerte

```ruby
# h[:x][:y][:z]
h = Hash.new do |hash, key|
  hash[key] = Hash.new {|h,k|
    h[k] = {}
  }
end

p h[:x][:y][:z] = '7'
# {:x=>{:y=>{:z=>7}}}
```

```ruby
h = Hash.new do |hash, key|
  hash[key] = Hash.new(&hash.default_proc)
end

p h[:x][:y][:z] = 7
# {:x=>{:y=>{:z=>7}}}
```

Hash

Nested Defaultwerte

```ruby
# h[:x][:y][:z]
h = Hash.new do |hash, key|
  hash[key] = Hash.new {|h,k|
    h[k] = {}
  }
end

p h[:x][:y][:z] = '7'
# {:x=>{:y=>{:z=>7}}}
```

```ruby
h = Hash.new do |hash, key|
  hash[key] = Hash.new(&hash.default_proc)
end

p h[:x][:y][:z] = 7
# {:x=>{:y=>{:z=>7}}}
```

Struct

```ruby
Struct.new('Pair', :lhs, :rhs)
# Struct::Pair

paar = Struct::Pair.new(7, 'mathema')
p paar
# <struct Struct::Pair lhs=7, rhs="mathema">

paar = Struct::Pair.new(7)
p paar
# <struct Struct::Pair lhs=7, rhs=nil>
```

```ruby
puts paar.lhs
puts paar[:lhs]
# => 7
```

```ruby
p Struct::Pair.members
p paar.members
# => [:lhs, :rhs]
p paar.values
# => [7, "mathema"]
```

Struct

```ruby
Struct.new('Pair', :lhs, :rhs)
# Struct::Pair

paar = Struct::Pair.new(7, 'mathema')
p paar
# <struct Struct::Pair lhs=7, rhs="mathema">

paar = Struct::Pair.new(7)
p paar
# <struct Struct::Pair lhs=7, rhs=nil>
```

```ruby
puts paar.lhs
puts paar[:lhs]
# => 7
```

```ruby
p Struct::Pair.members
p paar.members
# => [:lhs, :rhs]
p paar.values
# => [7, "mathema"]
```

Struct

```ruby
Struct.new('Pair', :lhs, :rhs)
# Struct::Pair

paar = Struct::Pair.new(7, 'mathema')
p paar
# <struct Struct::Pair lhs=7, rhs="mathema">

paar = Struct::Pair.new(7)
p paar
# <struct Struct::Pair lhs=7, rhs=nil>
```

```ruby
puts paar.lhs
puts paar[:lhs]
# => 7
```

```ruby
p Struct::Pair.members
p paar.members
# => [:lhs, :rhs]
p paar.values
# => [7, "mathema"]
```

Struct

```ruby
Pair = Struct.new(:lhs, :rhs)
# Pair

paar = Pair.new(7, 'mathema')
p paar
# <struct Pair lhs=7, rhs="mathema">

paar = Pair[8,9]
p paar
# <struct Pair lhs=8, rhs=9>
```

Struct

```ruby
Pair = Struct.new(:lhs, :rhs) do
  def to_s
    "(#{lhs}, #{rhs})"
  end
end
puts Pair['franz']
# => (franz, )
```

Struct

```ruby
pair = Struct.new(:lhs, :rhs)
p pair
# <Class:0x0000010109a2c0>

h = pair[0b1101, 0xcafebabe]
p h
# #<struct lhs=13, rhs=3405691582>
```

Wozu zur Hölle...?

generische Erzeugung
von Structs
mit gleichem Namen
zur Laufzeit

Struct

```ruby
pair = Struct.new(:lhs, :rhs)
p pair
# <Class:0x0000010109a2c0>

h = pair[0b1101, 0xcafebabe]
p h
# #<struct lhs=13, rhs=3405691582>
```

Wozu zur Hölle…?

generische Erzeugung
von Structs
mit gleichem Namen
zur Laufzeit

Struct

Wozu zur Hölle…?

let me explain!

```ruby
def db_read(table)
  case table
    when :people
      %w[Franz Fritz Horst].each_with_index.map {|name, idx|
        {:id => idx+1, :name => name.capitalize,
         :email => "#{name}@example.com"}
      }
    when :values then 1.upto(4).map {|i|  {id: i, value: i**2} }
    else
      raise ArgumentError.new("Table #{table} not known!")
  end
end
```

```ruby
p db_read(:people)
# [{:id=>1, :name=>"Franz", :email=>"Franz@example.com"},
#  {:id=>2, :name=>"Fritz", :email=>"Fritz@example.com"},
#  {:id=>3, :name=>"Horst", :email=>"Horst@example.com"}]
```

Struct

Wozu zur Hölle...?

let me explain!

```ruby
def db_read(table)
  case table
    when :people
      %w[Franz Fritz Horst].each_with_index.map {|name, idx|
        {:id => idx+1, :name => name.capitalize,
         :email => "#{name}@example.com"}
      }
    when :values then 1.upto(4).map {|i|  {id: i, value: i**2} }
    else
      raise ArgumentError.new("Table #{table} not known!")
  end
end
```

```ruby
p db_read(:people)
# [{:id=>1, :name=>"Franz", :email=>"Franz@example.com"},
#  {:id=>2, :name=>"Fritz", :email=>"Fritz@example.com"},
#  {:id=>3, :name=>"Horst", :email=>"Horst@example.com"}]
```

Struct

Wozu zur Hölle…?

let me explain!

```ruby
def create_struct_for_fields(fields)
  Struct.new(*fields) do
    def to_s
      each_pair.map { |field, value|
        "#{field}: #{value}"
      }.join(', ')
    end
  end
end
```

```ruby
row = create_struct_for_fields([:id, :name, :email])
p row.members # => [:id, :name, :email]
h = row[8, 'Johannes', 'mail@johannesheld.net']
puts h
# id: 8, name: Johannes, email: mail@johannesheld.net
```

Struct

Wozu zur Hölle…?

let me explain!

```ruby
def create_struct_for_fields(fields)
  Struct.new(*fields) do
    def to_s
      each_pair.map { |field, value|
        "#{field}: #{value}"
      }.join(', ')
    end
  end
end
```

```ruby
row = create_struct_for_fields([:id, :name, :email])
p row.members # => [:id, :name, :email]
h = row[8, 'Johannes', 'mail@johannesheld.net']
puts h
# id: 8, name: Johannes, email: mail@johannesheld.net
```

Struct

Wozu zur Hölle…? let me explain!

```
1  fields = %i(id name email)
2  row = create_struct_for_fields fields
3  db_row = {:id => 8, :name => 'Johannes',
4            :email => 'mail@johannesheld.net'}
5
6  h = db_row_2_my_row(db_row, row)
7  puts h
8  # => id: 8, name: Johannes, email: mail@johannesheld.net
```

Struct

Wozu zur Hölle…?

let me explain!

Umwandeln einer *db_row* in eine *row*

```
1  def db_row_2_my_row(db_row, row)
2    row.new(*db_row.values_at(*row.members))
3  end
```

```
1  row.members
2  # [:id, :name, :email]
```

```
1  db_row.values_at(:id, :name, :email)
2  # [8, 'Johannes', 'mail@johannesheld.net']
```

```
1  row.new(8, 'Johannes', 'mail@johannesheld.net')
2  # neue Instanz des Structs row
```

Struct

Wozu zur Hölle…?

let me explain!

Umwandeln einer *db_row* in eine *row*

```
1  def db_row_2_my_row(db_row, row)
2    row.new(*db_row.values_at(*row.members))
3  end
```

```
1  row.members
2  # [:id, :name, :email]
```

```
1  db_row.values_at(:id, :name, :email)
2  # [8, 'Johannes', 'mail@johannesheld.net']
```

```
1  row.new(8, 'Johannes', 'mail@johannesheld.net')
2  # neue Instanz des Structs row
```

Struct

Wozu zur Hölle…?

let me explain!

Umwandeln einer *db_row* in eine *row*

```ruby
def db_row_2_my_row(db_row, row)
  row.new(*db_row.values_at(*row.members))
end
```

```ruby
row.members
# [:id, :name, :email]
```

```ruby
db_row.values_at(:id, :name, :email)
# [8, 'Johannes', 'mail@johannesheld.net']
```

```ruby
row.new(8, 'Johannes', 'mail@johannesheld.net')
# neue Instanz des Structs row
```

Struct

Wozu zur Hölle…?

let me explain!

Umwandeln einer *db_row* in eine *row*

```ruby
def db_row_2_my_row(db_row, row)
  row.new(*db_row.values_at(*row.members))
end
```

```ruby
row.members
# [:id, :name, :email]
```

```ruby
db_row.values_at(:id, :name, :email)
# [8, 'Johannes', 'mail@johannesheld.net']
```

```ruby
row.new(8, 'Johannes', 'mail@johannesheld.net')
# neue Instanz des Structs row
```

Struct

Wozu zur Hölle…?

let me explain!

```ruby
def puts_db_data
  {people: [:id, :name, :email],
   values: [:id, :value]}.each do |table, fields|

    row = create_struct_for_fields(fields)

    rows = db_read(table).map {|db_row|
      db_row_2_my_row(db_row, row)
    }

    rows.each &method(:puts)  # rows.each {|row| puts row }

  end
end
```

Struct

Wozu zur Hölle…?

let me explain!

puts_db_data

```
id: 1, name: Franz, email: Franz@example.com
id: 2, name: Fritz, email: Fritz@example.com
id: 3, name: Horst, email: Horst@example.com
id: 1, value: 1
id: 2, value: 4
id: 3, value: 9
id: 4, value: 16
```

Section 6

Class

Klassen

Konstruktor und Methoden

```ruby
class Customer
  def initialize(name = 'Horst')
    @name = name
  end

  def greet
    "Angenehm, #{@name}"
  end
end

h = Customer.new
puts h.greet
# => Angenehm, Horst
```

Klassen

Instanzvariablen

```ruby
class Customer
  attr_reader :name
  # def name
  #   @name
  # end

  attr_writer :name
  # def name=(value)
  #   @name = value
  # end

  attr_accessor :name
end
```

Klassen

Klassenvariablen

```ruby
class Customer
  @@count = 0

  def initialize
    @@count += 1
  end

  def self.count
    @@count
  end
end

2.times { Customer.new }
puts Customer.count
# => 2
```

Klassen

Memoization mit ||=

```ruby
class Test
  def teure_berechnung
    puts 'das ist teuer'
    42
  end
  def a
    @a ||= teure_berechnung
  end
end
```

```ruby
t = Test.new
p t.a
# => das ist teuer
# 42
p t.a
# 42
```

```ruby
class Test
  def teure_berechnung
    puts 'das ist teuer'
    42
  end
  def a
    unless @a
      @a = teure_berechnung
    end
    @a
  end
end
```

Klassen

Memoization mit ||=

```ruby
class Test
  def teure_berechnung
    puts 'das ist teuer'
    42
  end
  def a
    @a ||= teure_berechnung
  end
end
```

```ruby
t = Test.new
p t.a
# => das ist teuer
# 42
p t.a
# 42
```

```ruby
class Test
  def teure_berechnung
    puts 'das ist teuer'
    42
  end
  def a
    unless @a
      @a = teure_berechnung
    end
    @a
  end
end
```

Klassen

Memoization mit ||=

```ruby
class Test
  def teure_berechnung
    puts 'das ist teuer'
    42
  end
  def a
    @a ||= teure_berechnung
  end
end
```

```ruby
t = Test.new
p t.a
# => das ist teuer
# 42
p t.a
# 42
```

```ruby
class Test
  def teure_berechnung
    puts 'das ist teuer'
    42
  end
  def a
    unless @a
      @a = teure_berechnung
    end
    @a
  end
end
```

||= und &&=

```ruby
a = nil
a ||= 7
puts a
# => 7

a = false
a ||= 7
puts a
#=> 7
```

```ruby
a = nil
a &&= 7
p a
# nil

a = 'wertig'
a &&= 7
p a
# 7
```

||= und &&=

```ruby
a = nil
a ||= 7
puts a
# => 7

a = false
a ||= 7
puts a
#=> 7
```

```ruby
a = nil
a &&= 7
p a
# nil

a = 'wertig'
a &&= 7
p a
# 7
```

Klassen

Private

```
1  class Customer
2    def tell_secret
3      secret
4    end
5    private
6    def secret
7      puts 'geheim'
8    end
9  end
10
11 h = Customer.new
12 h.secret  # private method 'secret' called …
13 h.tell_secret # => geheim
```

Klassen

Einfachvererbung

```ruby
class KeyAccount < Customer
  def initialize(name, value)
    super(name)
    @value = value
  end

  def greet
    ::greet + ":#{@value}"
  end
end

jo = KeyAccount.new 'Johannes', 7
puts jo.greet
# => Angenehm, Johannes:7
```

Klassen

Einfachvererbung

```ruby
class Customer
  private
  def secret
    puts 'geheim'
  end
end

class KeyAccount < Customer
  def tell_secret
    secret
  end
end

h = KeyAccount.new
h.secret  # private method 'secret' called …
puts h.tell_secret # => geheim
```

Klassen

Instanzmethoden

```
1  class Customer
2    # wie bisher
3  end
4  jo = Customer.new 'Johannes'
5
6  def jo.nick
7    'hehejo'
8  end
9
10 puts jo.nick
11 # => hehejo
```

Singleton

```ruby
require 'Singleton'
class MyLog
  include Singleton
  attr_reader :log

  def initialize
    @log = []
    @level = :info
  end

  def <<(msg)
    @log << "[#{@level}] #{msg}"
    self
  end

  def *(level); @level = level; self end
end
```

```ruby
l = MyLog.instance
l << 'test'
l * :warn << 'alert'
p l.log
# ["[info] test", "[warn] alert"]
```

Singleton

```ruby
require 'Singleton'
class MyLog
  include Singleton
  attr_reader :log

  def initialize
    @log = []
    @level = :info
  end

  def <<(msg)
    @log << "[#{@level}] #{msg}"
    self
  end

  def *(level); @level = level; self end
end
```

```ruby
l = MyLog.instance
l << 'test'
l * :warn << 'alert'
p l.log
# ["[info] test", "[warn] alert"]
```

Section 7

Module

Module

```ruby
module Mathema
  class HerbstCampus
    DATE = '01.–04.09.2014'
  end
end

puts Mathema::HerbstCampus::DATE
# => 01.–04.09.2014
```

Module

Mixin

```
module Foo
  def foo
    puts 'herbstcampus'
  end
end
```

include vs. extend

```
class Include
  include Foo
end
Include.new.foo
```

```
class Extend
  extend Foo
end
Extend.foo
```

Module

Mixin

include vs. extend

```ruby
module Foo
  def foo
    puts 'herbstcampus'
  end
end
```

```ruby
class Include
  include Foo
end
Include.new.foo
```

```ruby
class Extend
  extend Foo
end
Extend.foo
```

Module

Mixin

```
module Foo
  def foo
    puts 'herbstcampus'
  end
end
```

include vs. extend

```
class Include
  include Foo
end
Include.new.foo
```

```
class Extend
  extend Foo
end
Extend.foo
```

Module

Mixin

```ruby
module JoMath
  def pi; 3.14 end
end
puts JoMath.pi
# NoMethodError: undefined method
# ‘pi’ for JoMath:Module
```

```ruby
class Bar
  include JoMath
  def say_pi; pi end
end

b = Bar.new
puts b.say_pi
# => 3.14
puts b.pi
# => 3.14
```

extend self

```ruby
module JoMath
  def pi; 3.14 end
  module_function :pi
end
puts JoMath.pi
# => 3.14
```

```ruby
class Bar
  include JoMath
  def say_pi; pi end
end

b = Bar.new
puts b.say_pi
# => 3.14
puts b.pi
# private method ‘pi’ called
```

Module

Mixin

```ruby
module JoMath
  def pi; 3.14 end
end
puts JoMath.pi
# NoMethodError: undefined method
# ‘pi’ for JoMath:Module
```

```ruby
class Bar
  include JoMath
  def say_pi; pi end
end

b = Bar.new
puts b.say_pi
# => 3.14
puts b.pi
# => 3.14
```

extend self

```ruby
module JoMath
  def pi; 3.14 end
  module_function :pi
end
puts JoMath.pi
# => 3.14
```

```ruby
class Bar
  include JoMath
  def say_pi; pi end
end

b = Bar.new
puts b.say_pi
# => 3.14
puts b.pi
# private method ‘pi’ called
```

Module

Mixin

```
1  module JoMath
2    def pi; 3.14 end
3  end
4  puts JoMath.pi
5  # NoMethodError: undefined method
6  # ‘pi’ for JoMath:Module
```

```
1  class Bar
2    include JoMath
3    def say_pi; pi end
4  end
5
6  b = Bar.new
7  puts b.say_pi
8  # => 3.14
9  puts b.pi
10 # => 3.14
```

extend self

```
1  module JoMath
2    def pi; 3.14 end
3    module_function :pi
4  end
5  puts JoMath.pi
6  # => 3.14
```

```
1  class Bar
2    include JoMath
3    def say_pi; pi end
4  end
5
6  b = Bar.new
7  puts b.say_pi
8  # => 3.14
9  puts b.pi
10 # private method ‘pi’ called
```

Module

Mixin

```
1  module JoMath
2    def pi; 3.14 end
3  end
4  puts JoMath.pi
5  # NoMethodError: undefined method
6  # ‘pi’ for JoMath:Module
```

```
1  class Bar
2    include JoMath
3    def say_pi; pi end
4  end
5
6  b = Bar.new
7  puts b.say_pi
8  # => 3.14
9  puts b.pi
10 # => 3.14
```

extend self

```
1  module JoMath
2    def pi; 3.14 end
3    module_function :pi
4  end
5  puts JoMath.pi
6  # => 3.14
```

```
1  class Bar
2    include JoMath
3    def say_pi; pi end
4  end
5
6  b = Bar.new
7  puts b.say_pi
8  # => 3.14
9  puts b.pi
10 # private method ‘pi’ called
```

Module

Mixin

Hooks

```
1  module Foo
2    def self.extended(obj)
3      puts "I'm extending #{obj}"
4    end
5
6    def self.included(obj)
7      puts "I'm included in #{obj}"
8    end
9  end
```

Module

Mixin

Hooks

```ruby
module Foo
  def self.extended(obj)
    obj.send :prepend, Initializer
  end

  module Initializer
    def initialize(*args)
      super
      puts 'Foo#Initializer'
    end
  end
end
```

```ruby
class Bar
  extend Foo
end
Bar.new
# => Foo#Initializer
```

Module

Mixin

```
module Foo
  def self.extended(obj)
    obj.send :prepend, Initializer
  end

  module Initializer
    def initialize(*args)
      super
      puts 'Foo#Initializer'
    end
  end
end
```

Hooks

```
class Bar
  extend Foo
end
Bar.new
# => Foo#Initializer
```

Section 8

Exceptions

Exceptions

2 Arten

- raise => rescue
- throw => catch

Exceptions

raise => rescue

```ruby
def raiser
  raise ArgumentError.new 'bomb'
end
```

```ruby
def vorsichtig
  begin
    raiser
  rescue Exception => err
    p err
  end
end
vorsichtig
# <ArgumentError: bomb>
```

```ruby
def vorsichtig
  raiser
rescue Exception => err
  p err
end
```

Exceptions

raise => rescue

```ruby
def raiser
  raise ArgumentError.new 'bomb'
end
```

```ruby
def vorsichtig
  begin
    raiser
  rescue Exception => err
    p err
  end
end
vorsichtig
# <ArgumentError: bomb>
```

```ruby
def vorsichtig
  raiser
rescue Exception => err
  p err
end
```

Exceptions

raise => rescue

```ruby
def raiser
  raise ArgumentError.new 'bomb'
end
```

```ruby
def vorsichtig
  begin
    raiser
  rescue Exception => err
    p err
  end
end
vorsichtig
# <ArgumentError: bomb>
```

```ruby
def vorsichtig
  raiser
rescue Exception => err
  p err
end
```

Exceptions

raise => rescue

```ruby
def raiser
  raise ArgumentError.new 'bomb'
end
```

```ruby
def vorsichtig
  begin
    raiser
  rescue Exception => err
    p err
  end
end
vorsichtig
# <ArgumentError: bomb>
```

```ruby
def vorsichtig
  raiser
rescue Exception => err
  p err
end
```

Exceptions

```
1  begin
2    # do something
3  rescue
4    # handle exception
5    retry # let's try again
6  else
7    # do this if no exception was raised
8  ensure
9    # do this whether or not an exception was raised
10 end
```

Exceptions

throw => catch

```ruby
def thrower(throw_it = false)
  throw(:ergebnis, 42) if throw_it
end
```

```ruby
z = catch :ergebnis do
  thrower :throw_it
  13
end
puts z
# => 42
```

```ruby
z = catch :ergebnis do
  thrower
  13
end
puts z
# => 13
```

Exceptions

throw => catch

```ruby
def thrower(throw_it = false)
  throw(:ergebnis, 42) if throw_it
end
```

```ruby
z = catch :ergebnis do
  thrower :throw_it
  13
end
puts z
# => 42
```

```ruby
z = catch :ergebnis do
  thrower
  13
end
puts z
# => 13
```

Exceptions

throw => catch

```ruby
def thrower(throw_it = false)
  throw(:ergebnis, 42) if throw_it
end
```

```ruby
z = catch :ergebnis do
  thrower :throw_it
  13
end
puts z
# => 42
```

```ruby
z = catch :ergebnis do
  thrower
  13
end
puts z
# => 13
```

Beispiel für throw => catch

Gegeben 3-stufiger Hash

```
#familien[hausnr][etage][wohnung] => Familie
familien = {20 => {1 => {1 => 'Held',  2 => 'Müller'},
                   2 => {1 => 'Mayer', 2 => 'Meyer'}},
             22 => {1 => {1 => 'Maier', 2 => 'Huber'},
                   2 => {1 => 'Mayr',  2 => 'Meyr'}}}
```

Gesucht: Hausnummer, Etage, Wohnung für Familie Held

Beispiel für throw => catch

Gegeben 3-stufiger Hash

```
#familien[hausnr][etage][wohnung] => Familie
familien = {20 => {1 => {1 => 'Held',  2 => 'Müller'},
                   2 => {1 => 'Mayer', 2 => 'Meyer'}},
             22 => {1 => {1 => 'Maier', 2 => 'Huber'},
                   2 => {1 => 'Mayr',  2 => 'Meyr'}}}
```

Gesucht: Hausnummer, Etage, Wohnung für Familie Held

Beispiel für throw => catch

```
found = false
keys = []
familien.each {|hausnr, etagen|
  p etagen
  etagen.each {|etage, wohnungen|
    print "\t"; p wohnungen
    wohnungen.each {|wohnung, name|
      print "\t\t"; p name
      if name == 'Held'
        found = true
        keys.unshift wohnung
        break
      end
    }
    if found
      keys.unshift etage
      break
    end
  }
  if found
    keys.unshift hausnr
    break
  end
}
# {1=>{1=>"Held", 2=>"Müller"}, 2=>{1=>"Mayer", 2=>"Meyer"}}
#   {1=>"Held", 2=>"Müller"}
#     "Held"
p keys
# [20, 1, 1]
```

Beispiel für throw => catch

```ruby
keys = catch :found do
  familien.each {|hausnr, etagen|
    p etagen
    etagen.each {|etage, wohnungen|
      print "\t"; p wohnungen
      wohnungen.each {|wohnung, name|
        print "\t\t"; p name
        throw(:found, [hausnr, etage, wohnung]) if name == 'Held'
      }
    }
  }
  []
end
# {1=>{1=>"Held", 2=>"Müller"}, 2=>{1=>"Mayer", 2=>"Meyer"}}
#   {1=>"Held", 2=>"Müller"}
#     "Held"
p keys
# [20, 1, 1]
```

Section 9

Proc & λ

Was ist eigentlich ein Block?

```ruby
def test(name)
  yield name
end

p test('egon') {|n|
  n.capitalize
}
# "Egon"
```

Was ist eigentlich ein Block?

```ruby
def test(name, &block)
  yield name
end

p test('egon') {|n| n.capitalize}
# "Egon"
```

```ruby
f = Proc.new {|n| n.capitalize}
p test('egon', &f)
# "Egon"
```

Was ist eigentlich ein Block?

```ruby
def test(name, &block)
  yield name
end

p test('egon') {|n| n.capitalize}
# "Egon"
```

```ruby
f = Proc.new {|n| n.capitalize}
p test('egon', &f)
# "Egon"
```

Was ist eigentlich ein Block?

Exkurs

```ruby
def cap(n)
  n.capitalize
end

p test('egon', &method(:cap))
# "Egon"
```

```ruby
p test('egon', &:capitalize)
# "Egon"
```

Was ist eigentlich ein Block?

Exkurs

```ruby
def cap(n)
  n.capitalize
end

p test('egon', &method(:cap))
# "Egon"
```

```ruby
p test('egon', &:capitalize)
# "Egon"
```

Was ist eigentlich ein Block?

```ruby
f = Proc.new {|n| n.capitalize}

p f.call('egon')
# "Egon"
p f.call('ernie', 'bert')
# "Ernie"
```

```ruby
def test(name, func)
  func[name] # func.call(name)
end

p test('egon', f)
# "Egon"
```

Was ist eigentlich ein Block?

```ruby
f = Proc.new {|n| n.capitalize}

p f.call('egon')
# "Egon"
p f.call('ernie', 'bert')
# "Ernie"
```

```ruby
def test(name, func)
  func[name] # func.call(name)
end

p test('egon', f)
# "Egon"
```

Was ist eigentlich ein Block?

```ruby
f = Proc.new {|lhs, rhs|
  return "#{lhs} has no friend" unless rhs
  "#{lhs} & #{rhs}"
}
```

```ruby
p f.call('ernie', 'bert')
# "ernie & bert"
```

```ruby
p f.call('egon')
# unexpected return (LocalJumpError)
```

Was ist eigentlich ein Block?

```
f = Proc.new {|lhs, rhs|
  return "#{lhs} has no friend" unless rhs
  "#{lhs} & #{rhs}"
}
```

```
p f.call('ernie', 'bert')
# "ernie & bert"
```

```
p f.call('egon')
# unexpected return (LocalJumpError)
```

Was ist eigentlich ein Block?

```ruby
f = Proc.new {|lhs, rhs|
  unless rhs
    "#{lhs} has no friend"
  else
    "#{lhs} & #{rhs}"
  end
}

p f.call('egon')
# "egon has no friend"
```

Enter λ

```
l = ->(lhs, rhs) {
  return "#{lhs} has no friend" unless rhs
  "#{lhs} & #{rhs}"
}
p l.call('ernie', 'bert')
# "ernie & bert"
```

```
p l.call('egon')
# wrong number of arguments (1 for 2) (ArgumentError)
```

```
l = lambda {|wort| wort.upcase}
p l['foo']
# "FOO"
```

Enter λ

```ruby
l = ->(lhs, rhs) {
  return "#{lhs} has no friend" unless rhs
  "#{lhs} & #{rhs}"
}
p l.call('ernie', 'bert')
# "ernie & bert"
```

```ruby
p l.call('egon')
# wrong number of arguments (1 for 2) (ArgumentError)
```

```ruby
l = lambda {|wort| wort.upcase}
p l['foo']
# "FOO"
```

Enter λ

```
l = ->(lhs, rhs) {
  return "#{lhs} has no friend" unless rhs
  "#{lhs} & #{rhs}"
}
p l.call('ernie', 'bert')
# "ernie & bert"
```

```
p l.call('egon')
# wrong number of arguments (1 for 2) (ArgumentError)
```

```
l = lambda {|wort| wort.upcase}
p l['foo']
# "FOO"
```

Closure

```ruby
def build_closure
  a = []
  f = ->() {a.size}
  a << 8
  f
end

func = build_closure
```

```ruby
p func[]
# 1
```

Closure

```ruby
def build_closure
  a = []
  f = ->() {a.size}
  a << 8
  f
end

func = build_closure
```

```ruby
p func[]
# 1
```

Proc vs. λ

Proc

- wie *glued code*
- return nicht möglich
- prüft Anzahl der Argumente *nicht*

λ

- wie anonyme Funktion
- return möglich
- prüft Anzahl der Argumente

Proc vs. λ

Proc

- wie *glued code*
- return nicht möglich
- prüft Anzahl der Argumente *nicht*

λ

- wie anonyme Funktion
- return möglich
- prüft Anzahl der Argumente

und noch viel mehr…

Create DSL with Ruby

`http://www.artima.com/rubycs/articles/ruby_as_dsl.html`

JRuby

`http://www.jruby.org`

Quellen und nützliche Links

- http://www.ruby-lang.org
- http://blog.arkency.com/2014/07/hidden-features-of-ruby-you-may-dont-know-about
- http://www.zenspider.com/Languages/Ruby/QuickRef.html
- http://juixe.com/techknow/index.php/2006/06/15/mixins-in-ruby
- http://www.rubyinside.com/21-ruby-tricks-902.html
- http://www.artima.com/rubycs/articles/ruby_as_dsl.html
- http://www.devalot.com/articles/2008/09/ruby-singleton
- http://www.vitarara.org/cms/ruby_metaprogamming_declaratively_adding_methods_to_a_class
- http://matt.aimonetti.net/posts/2012/07/30/ruby-class-module-mixins
- http://graysoftinc.com
- https://www.ruby-toolbox.com
- http://mislav.uniqpath.com/poignant-guide