



Ruby

Eine elegante und ausdrucksvolle
Programmiersprache!

Informatik Workshop

WS 2021/22

Hochschule Mannheim



Die Ruby Grundlagen kennenlernen und verstehen!



Ruby Kernkonzepte nachvollziehen und anwenden!



Fortgeschrittene Ruby Features nutzen!



Den Ruby Horizont erweitern!



Mariia
Robota



Domenic
Gosein



Sebastian
Schäfer



Christian
Müller



Adrian
Petschenka



Timo
Scheuermann

Unit 2

Unit 1

Unit 3

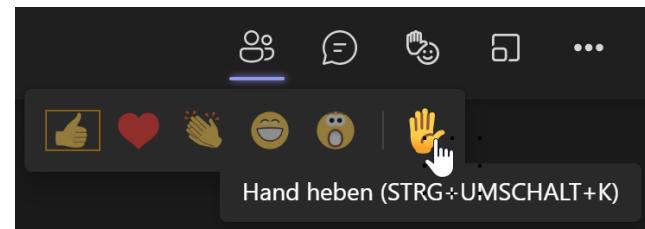
9:00 – 9:15 Uhr	Einführung
9:15 – 10:40 Uhr	Unit 1: Grundlagen
	Coffee Break
10:45 – 12:10 Uhr	Unit 2: Objektorientierung
	Lunch Break
12:55 – 13:55 Uhr	Unit 3: Weiterführendes
	Coffee Break
14:00 – 14:20 Uhr	Ausblick
	Coffee Break
14:25 – 14:45 Uhr	Feedback Session & Quiz Auswertung
	Schluss

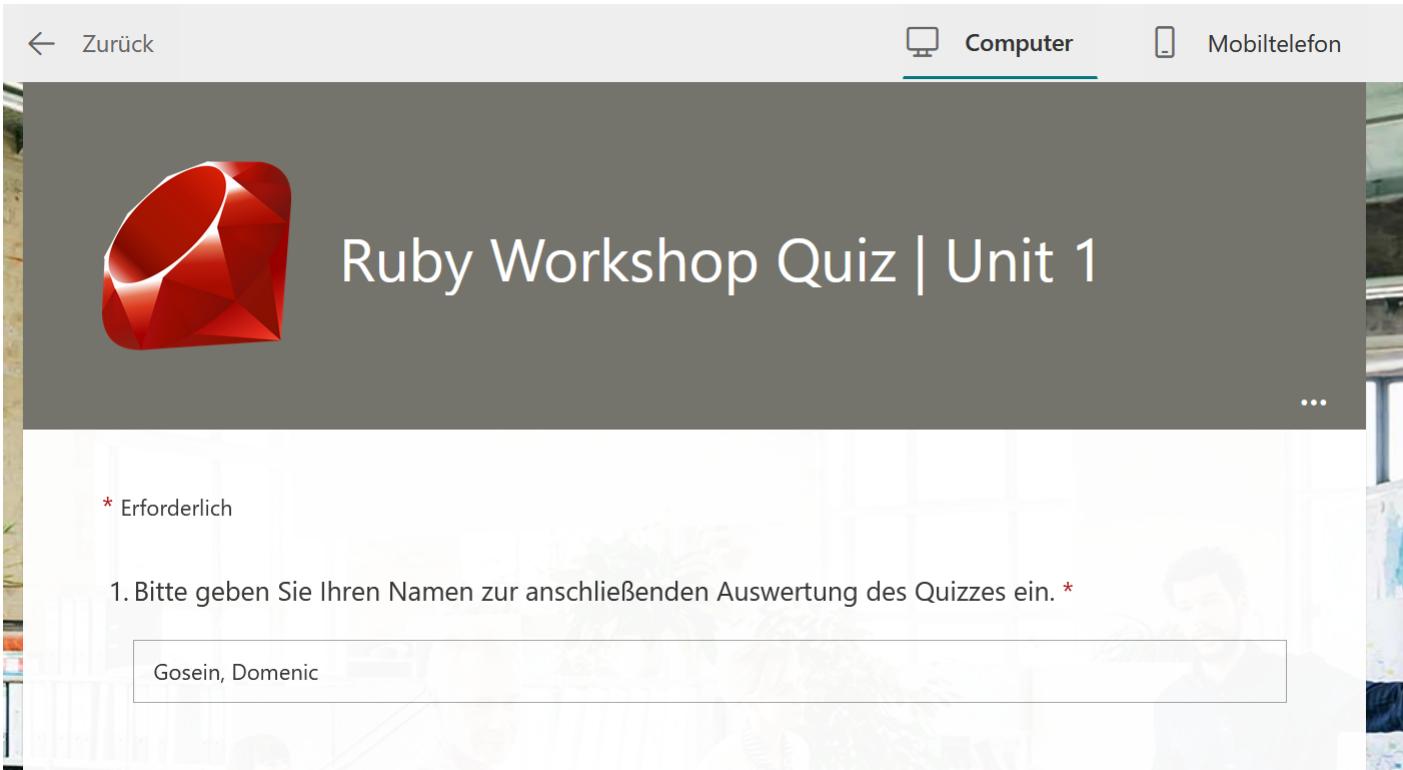
Der Workshop besteht aus:

- **Theorie** mit der Möglichkeit, interaktiv mitzuarbeiten und Dinge auszuprobieren
- **Übungen** zur Vertiefung der gezeigten Theorie in Gruppen
- **Reflexion** in Form von Quizzen nach jeder Einheit



Fragen sind jederzeit gerne erwünscht, wir bitten allerdings alle Teilnehmer, die „Hand heben“ Funktion in Teams zu verwenden!





The screenshot shows a quiz interface. At the top left is a back button labeled "Zurück". In the top right, there are two tabs: "Computer" (selected) and "Mobiltelefon". The main title "Ruby Workshop Quiz | Unit 1" is displayed prominently. On the left side of the form area, there is a red 3D Ruby logo icon. Below the title, a note says "* Erforderlich". The first question asks: "1. Bitte geben Sie Ihren Namen zur anschließenden Auswertung des Quizzes ein.*". A text input field contains the name "Gosein, Domenic".

→ Bitte immer denselben Namen in folgendem Format angeben:
Nachname, Vorname

Arbeitsumgebung



Benutzername	Teilnehmer
nbu01	Peter Knauber
nbu02	Markus Gumbel
nbu03	Alexis Dos Santos
nbu04	Jannis Seefeld
nbu05	Maximilian Kürschner
nbu06	Mike Sickmüller
nbu07	Pascal Egner
nbu08	Tuncay Yildiz
nbu09	Lukas Weinkoetz

Für Mitarbeit und Aufgaben steht
unser **iRuby Jupyter Notebook** zur
Verfügung:

<https://bit.ly/workshop-ruby>

Kennwort für alle Teilnehmer:

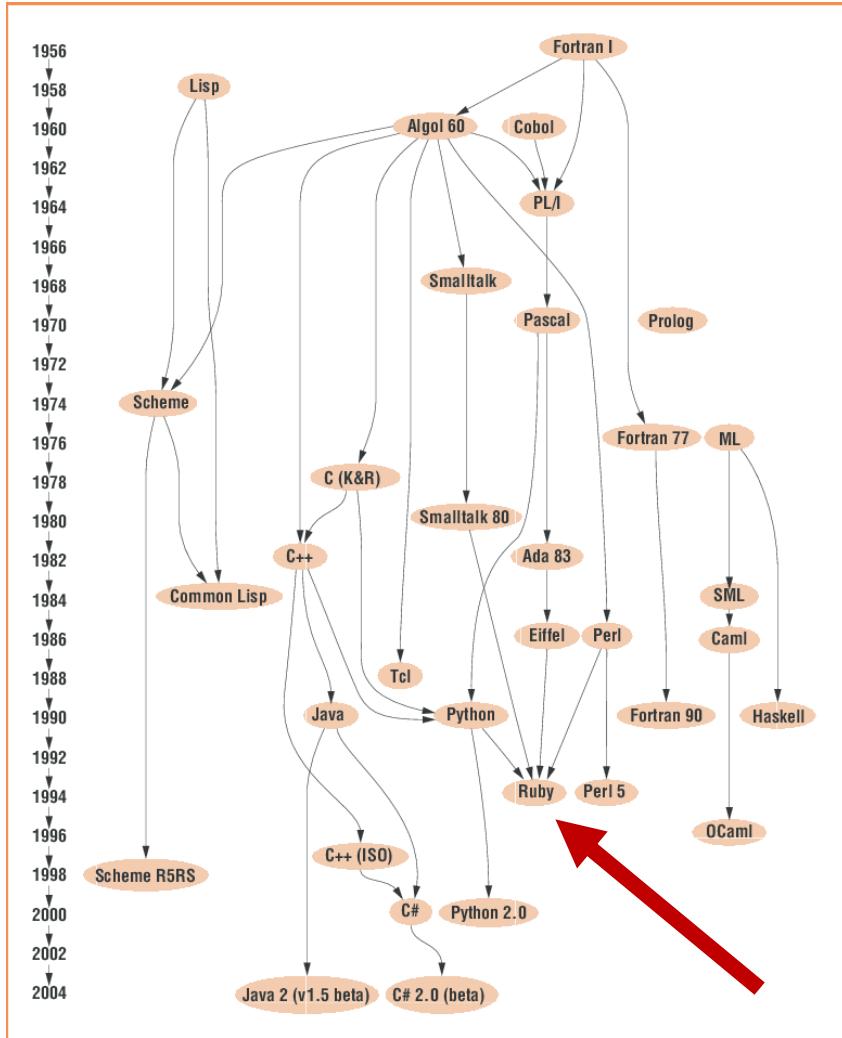
nbutest



- universell, objektorientiert und interpretiert
- ausdrucksvooll und elegant
- 1993 entworfen von Yukihiro Matsumoto >Matz<

**“Ruby wurde entworfen, um
Programmierer glücklich zu machen.”**

Wie ist es entstanden und wo kommt es vor?



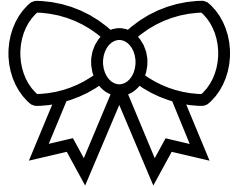
airbnb

Bloomberg

couchsurfing

shopify

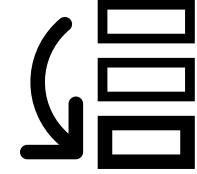
Was macht Ruby so interessant für uns?



- Jeder Wert ist ein Objekt
- Klammern sind optional
- Iteratoren und Blöcke statt Schleifen



- Verknüpfung von Codeblocks mit Methodenaufrufen
- Kontrollstrukturen wie Anweisungen sind in Ruby Ausdrücke
- Ruby-Operatoren sind als Methoden implementiert



- Klassen können diese Methoden definieren/umdefinieren (auch Kernklassen sind offen)
- Parallelle Wertzuweisung sind möglich
- Variablenpräfixe, Reguläre Ausdrücke und Bereiche engl. Ranges

1. Variablen und Datentypen
2. Arrays
3. Ranges
4. Operatoren
5. Fallunterscheidungen
6. Schleifen, Iteratoren und Enumeratorn
7. Blöcke
8. Steuerungsablauf (flow-of-control)
9. Ausnahmen (Exceptions)
10. Methoden



Nach dieser Einheit solltet ihr in der Lage sein, die **grundlegenden Abläufe in Ruby sowie dessen Syntax** zu verstehen!

Interactive Ruby Shell (IRB oder irb):

- ist ein REPL (Read Evaluate Print Loop) für die Programmierung in der objektorientierten Skriptsprache Ruby
- führt Ruby-Befehle mit sofortiger Reaktion aus (interaktiv)
- verfügt über eine Befehlshistorie
- gibt Rückgabewert aus, ohne *print* oder *puts* Befehle
- wurde von Keiju Ishitsuka entwickelt

- Alles ist ein Objekt - keine primitiven Datentypen
- **Nil** ist ein Objekt von **NilClass** - repräsentiert nichts
- Integer, Float, String, keinen Datentyp char
- Variablen sind ***immer*** Referenzen auf Objekte
- Parallele Zuweisung

Unveränderliche Werte werden als Konstanten bezeichnet,
z. B. der Wert von Pi (π)

- Konstanten beginnen mit einem Großbuchstaben
- Werte der Konstanten lassen sich verändern,
aber es gibt immer eine Warnung

- Ein Array ist eine Sammlung von Objekten
- Jedes Objekt hat eine Position, einen Index
- Initialisierung: `s = []` `s = Array.new`

`s[0] = 0` `s.push(1)` `s << 2`

Gängigste Methoden:

<code>push</code>	<code>pop</code>	<code>select</code>	<code>first</code>	<code>last</code>
<code>length</code>	<code>each</code>	<code>map</code>	<code>take</code>	<code>drop</code>
<code>delete</code>	<code>include?</code>	<code>uniq</code>	<code>join</code>	<code>delete_at</code>
<code>shift</code>	<code>unshift</code>	<code>reverse</code>	<code>concat</code>	

- Ranges sind ein eigener Datentyp in Ruby
- Beschreiben einen Bereich von Werten (Intervall)
 - a..b: Von a bis b **inklusive** b
 - a...b: Von a bis b **exklusive** b
- Vorteil von Ranges:
 - Ranges sind **speichersparend**: anstatt alle Werte zu speichern, müssen nur die Grenzen abgelegt werden
 - Mit der **include?**- Methode wird geprüft, ob ein Wert im Range liegt

defined?-Operator

Der Operator prüft, ob sein Argument definiert ist:

- nicht definiert → nil und somit false
- definiert → Beschreibung (z. B. local-variable) und somit true

Bemerkung:

Mit dem Operator kann man nicht prüfen, ob eine Variable nil enthält, z.B.:

```
k = 22  
defined? k
```

"local-variable"

```
defined? u=1
```

"assignment"

```
defined? puts
```

"method"

```
w = nil
```

```
defined? w
```

"local-variable"

```
w.nil?
```

true

Splat -Operator (*)

Um eine undefinierte Anzahl von Argumenten in Ruby zu behandeln, haben wir einen Parameter mit dem Splat-Operator (*)

Parameter mit dem Splat-Operator:

- nimmt nur die Argumente auf, für die keine anderen Parameter vorhanden sind
- ist optional
- gibt bei mehreren Argumenten ein Array zurück

```
a, b, c, d, e = *[1,2,3],*[4,5]
print a, " ", b, " ", c, " ", d, " ",e
1 2 3 4 5
```

```
a,* c = 1,2,3,4,5,6,7
puts a
print c
1
[2, 3, 4, 5, 6, 7]
```

```
*a, d = 1,2,3,4,5,6,7,8
puts d
print a
8
[1, 2, 3, 4, 5, 6, 7]
```

```
a,* b = 1
puts a
print b
1
[]
```

Der ternäre Operator ermöglicht es, dass ein Ausdruck eine kleine if/else-Anweisung enthält:

```
alter = 10
typ = alter < 18 ? "Kind" : "Erwachsener"
puts "Der Benutzer ist ein " + typ
```

Der Benutzer ist ein Kind

Vergleich für Sortierung (-1, 0, 1):

-1, wenn a kleiner als b ist

0, wenn a und b gleich sind

1, wenn a größer als b ist

5<=>5

0

"A"<=>"B"

-1

5<=>4

1

5 .<=>4

1

&& - Operator

```
#a && b: => a wenn a false ist  
puts 1 > 2 && 3
```

false

```
#a && b: => b wenn b true ist  
puts "Hallo" && 3
```

3

```
puts "Hallo" && "Welt"
```

Welt

|| - Operator

```
#a || b: => a wenn a true ist
```

```
a = 2
```

```
puts a || 15 < 3
```

2

```
#a || b: => b wenn a false ist
```

```
w = 2; v = 3; d = "Hallo"
```

```
puts (w == v) || d
```

Hallo

eql?: Vergleicht Wert und Typ

```
a = 2; b = 2.0; c = 2.0
```

```
a.eql?(b)
```

```
false
```

```
c.eql?(b)
```

```
true
```

```
a == b
```

```
true
```

```
c == b
```

```
true
```

equal?: Check auf dieselben Objekte

```
a = "Hallo Welt"; b = "Hallo Welt"  
c = a
```

```
a.equal?(b)
```

```
false
```

```
a.equal?(c)
```

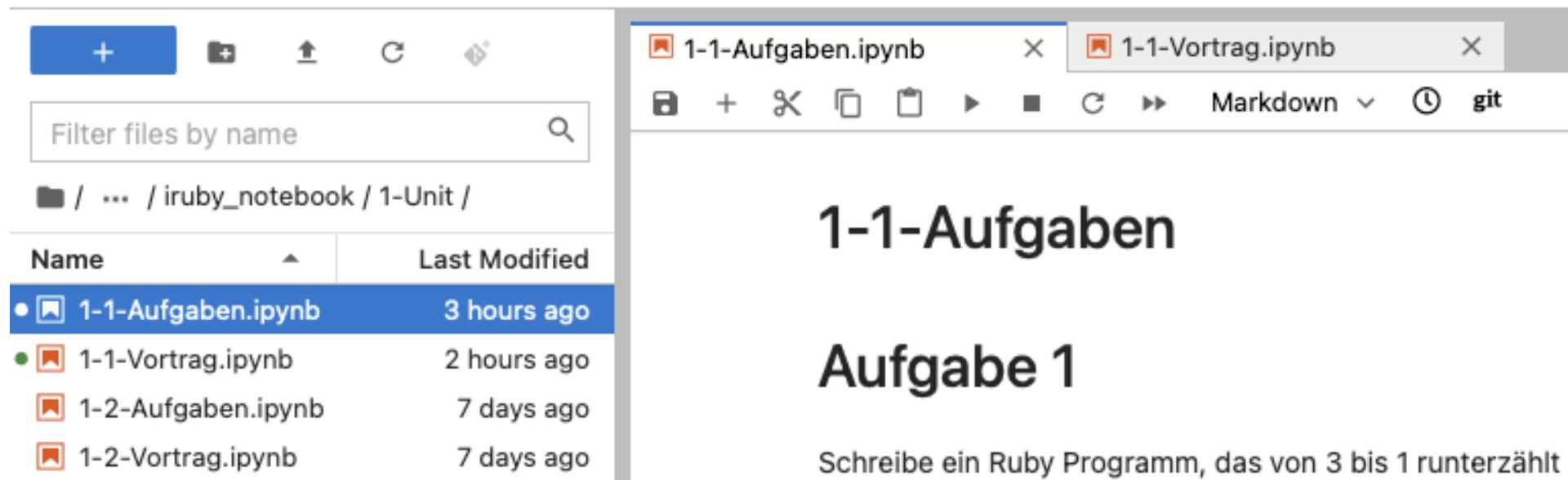
```
true
```

```
a == b
```

```
true
```

```
a == c
```

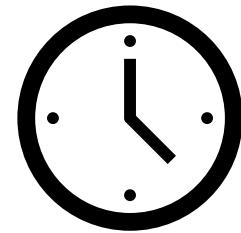
```
true
```



The screenshot shows a file manager interface with two tabs at the top: '1-1-Aufgaben.ipynb' and '1-1-Vortrag.ipynb'. The left pane displays a list of files in the directory '/ ... / iruby_notebook / 1-Unit /'. The files listed are:

Name	Last Modified
1-1-Aufgaben.ipynb	3 hours ago
1-1-Vortrag.ipynb	2 hours ago
1-2-Aufgaben.ipynb	7 days ago
1-2-Vortrag.ipynb	7 days ago

The right pane contains the content of the selected notebook, '1-1-Aufgaben.ipynb'. It features a large title '1-1-Aufgaben' and a section titled 'Aufgabe 1' with the instruction: 'Schreibe ein Ruby Programm, das von 3 bis 1 runterzählt'.



5 Minuten

- Immer **true**, außer: **false, nil**
- Trennung durch Zeilenumbruch, ; oder Schlüsselwort **then**
- Alles ist ein Ausdruck und hat einen Rückgabewert!
- Bei **case** anstatt **elseif** → **when**

```
# Eine Bedingung ist wahr, außer für: false, nil
x = ""
puts "Ein leerer String ist wahr!" if x
```

Ein leerer String ist wahr!

```
# Einfache elseif Fallunterscheidung
x = 3
if x == 1
  name = "eins"
elsif x == 2
  name = "zwei"
elsif x == 3 then name = "drei"
elsif x == 4; name = "vier"
else
  name = "viele"
end
```

"drei"

```
# Elegante Schreibweise der Fallunterscheidung
name = if x == 1 then "eins"
        elsif x == 2 then "zwei"
        elsif x == 3 then "drei"
        elsif x == 4 then "vier"
        else
          "viele"
end
```

"drei"

```
# Dasselbe Beispiel mit case
name = case
  when x == 1 then "eins"
  when x == 2 then "zwei"
  when x == 3 then "drei"
  when x == 4 then "vier"
  else
    "viele"
end
```

"drei"

- Modifier (Einzeiler): **Code if Ausdruck**
- **unless**, das Gegenteil von **if** ohne **elseif**

```
# Umgekehrte Schreibweise
"drei" if x == 3
```

"drei"

```
# Das Gegenteil von if: unless
alter = 17
unless alter >= 18
  puts "Nicht alt genug."
end
```

Nicht alt genug.

```
# unless mit else
unless alter >= 18
  puts "Nicht alt genug."
else
  puts "Alt genug."
end
```

Nicht alt genug.

```
# Umgekehrte unless Schreibweise
puts "Nicht alt genug." unless alter >= 18
```

Nicht alt genug.

- **while** und **until** als Modifier/Einzeiler

- **until** ist die Umkehrung von **while**

- Schleifen beenden mit **end**

- **for**-Schleife ruft **each** Methode auf

```
# while (Kopf-)Schleife Beispiel
max = 28
min = 30
while max < min do
    puts "Maximum noch nicht erreicht!"
    max += 1
end
```

Maximum noch nicht erreicht!
Maximum noch nicht erreicht!

```
# while (Fuß-)Schleife Beispiel
max = 28
min = 30
begin
    puts "Maximum noch nicht erreicht!"
    max += 1
end while max < min
```

Maximum noch nicht erreicht!
Maximum noch nicht erreicht!

```
# until Schleife Beispiel
a = [1,2,3]
puts a.pop until a.empty?
puts "#{a}\n"
```

3
2
1
[]

```
# for Schleife Beispiel
for i in 0..2
    puts "Wert der lokalen Variable: #{i}"
end
```

Wert der lokalen Variable: 0
Wert der lokalen Variable: 1
Wert der lokalen Variable: 2
0..2

- Einfachster Iterator: ***loop***
 - Unterbrechung mit ***return/break***
- Gängige Iteratoren: ***times, each, collect/map, upto*** und ***downto***
- Agieren mit dem darauffolgenden Codeblock

```
# Loop Beispiel
i = 0
loop do
  break if i > 2
  puts i
  i += 1
end
```

```
0
1
2
```

```
# each Beispiel
Sammlung = ["Eins", "Zwei", "Drei", "Los!"]
Sammlung.each do |element|
  puts element
end
```

```
Eins
Zwei
Drei
Los!
["Eins", "Zwei", "Drei", "Los!"]
```

```
# .each Beispiel mit einem Hash
hash = { :a=>1, :b=>2, :c=>3 }
hash.each do |key,value|
  puts "#{key} => #{value}"
end
```

```
a => 1
b => 2
c => 3
{:a=>1, :b=>2, :c=>3}
```

- Eigenständige Entitäten, die durch `{}` oder `do/end` begrenzt werden
- Blockparameter können mit `|` parametrisiert werden
- Variablen nur innerhalb sichtbar

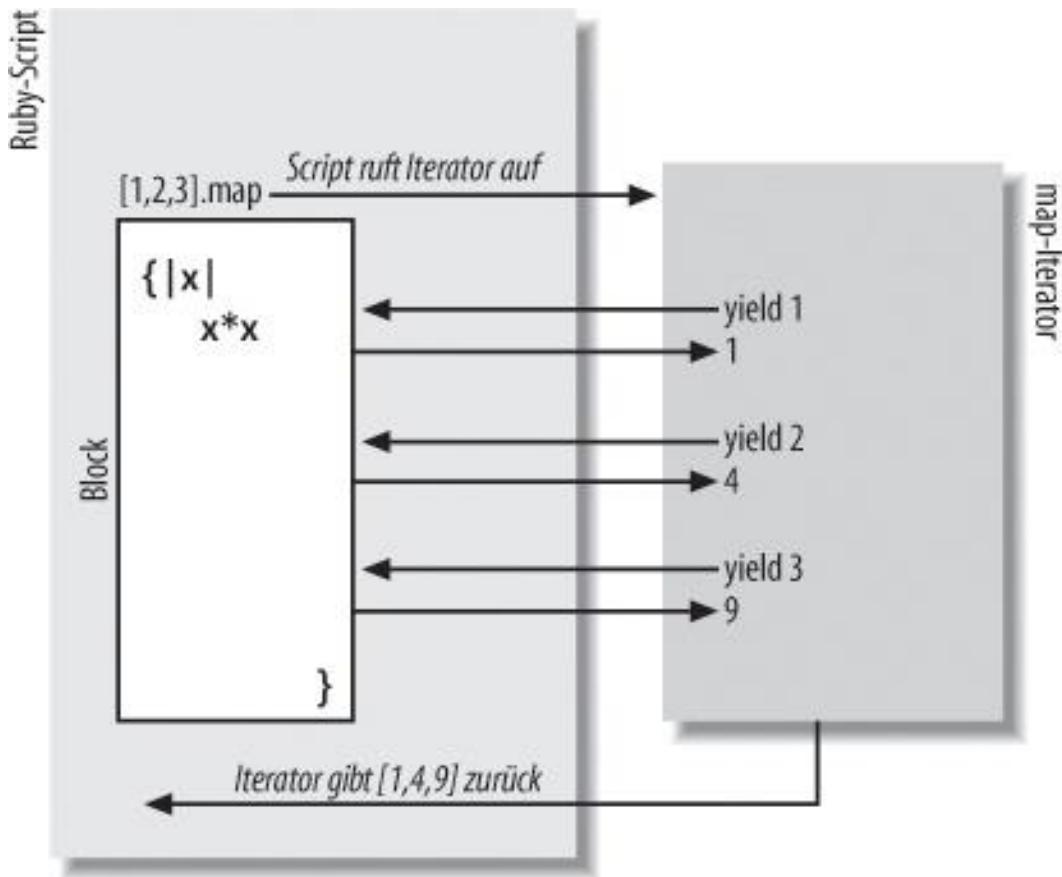
```
# Beispiel Parametrisierung mit upto {}
1.upto(3) {|x| puts x }
```

```
1
2
3
1
```

```
# Beispiel Parametrisierung mit upto do/end
1.upto(3) do |x|
  puts x
end
```

```
1
2
3
1
```

- Hinter diesen Methoden steckt ***yield***, eine komplexe Kontrollstruktur, die die Kontrolle vorübergehend an die Methode zurückgibt



- Spezielle Iteratoren Enumerable-Objekte

- Kontrolle über Ausführung/Objekt

- Iteratoren wie ***each*** liefern einen Enumerator zurück

- ***each_char*** oder ***each_with_index*** sind spezielle Enumeratoren

```
# Enumerator erzeugen und Elemente nacheinander ausgeben
```

```
a = [ 'a', 'b', 'c', 'd']
enum = a.to_enum
enum.next
```

"a"

```
# Enumerator mit Index
```

```
chars = "Enumerator".each_char
chars.with_index { |c, i| print "#{i}:#{c} \n" }
```

0:E
1:n
2:u
3:m
4:e
5:r
6:a
7:t
8:o
9:r

"Enumerator"

Steuerungsablauf (flow-of-control)



- **return** → Methode beenden/Wert zurückgeben

```
# return Beispiel
def two_copies(x)
  return nil if x == nil
  return x, x.dup
end
two_copies(5)
```

[5, 5]

- **break** → beenden

```
# break Beispiel
woerter = ["Anfang", "Mitte", "Ende"]
woerter.each do |w|
  break if w == "Mitte"
  puts w
end
```

Anfang

- **throw/catch** → Aus Schleife ausbrechen

```
# throw/catch Beispiel
catch (:done) do
  1.upto(100) do |i|
    1.upto(100) do |j|
      throw :done if j > 3
      puts j
    end
  end
end
```

1
2
3

- **next** → überspringen/nächste Iteration starten

```
# next Beispiel
a = [1, 2, 3]
a.each do |num|
  next if num < 2
  puts num
end
```

2
3
[1, 2, 3]

- **redo** → Schleife/Iterator am Anfang neu starten

```
# redo Beispiel
i = 0
while(i < 3)
  # Springt bei redo wieder hierhin
  print i
  i += 1
  redo if i == 3
end
```

0123

- **rescue** um Ausnahme zu fangen
- Erzeugen von Ausnahmeobjekten mit **raise**
- **raise** nimmt ein Argument als **message** entgegen

```
# rescue Beispiel durch Null teilen
begin
  x = 1/0
  rescue => ex
  puts "#{ex.class}: #{ex.message}"
end
```

```
ZeroDivisionError: divided by 0
```

```
# raise Beispiel durch Null teilen
def teilen(x, y)
  raise "Durch Null teilen geht nicht!" if x == 0 || y == 0
  x/y
end
teilen(2.5, 0)
```

```
RuntimeError: Durch Null teilen geht nicht!
(irb):2:in `teilen'
(irb):5:in `irb_binding'
```

- Methoden werden mit ***def*** definiert
 - es folgt **Methodename, Parameter und Methoden-Rumpf**
 - ***alias aka also_known_as***
- Mit ***method var, method()*** oder ***.method*** aufrufen
- Klammer nicht immer nötig
- Methodename mit ***!, ? oder =***

```
# Definition einer Methode mit Parameterstandardwert
def rm_prefix(str, len=5)
  str[len,str.length]
end
rm_prefix("2022 Ruby Workshop")
```

"Ruby Workshop"

```
# Methodenalias festlegen
alias rmp rm_prefix
rmp("2022 Ruby Workshop")
```

"Ruby Workshop"

```
# Ausgeben mit puts
message = "Ich bins"
puts message
```

Ich bins

```
# Trennung mit einem Punkt
message.length
```

8

```
# Trennung mit einem Punkt und Übergabe eines Arguments
Math.sqrt(2)
```

1.4142135623730951

```
# Array Inhalt überprüfen
a.empty?
```

false

- Singleton Methoden, die nur für ein Objekt definiert wurden
- Splat * für mehrere Argumente
- Hash für Parameter/Argument Zuordnung

```
# Singleton-Methode für nur ein Objekt
obj = "Warning"
def obj.printwarn
  puts self
end
obj.printwarn
```

```
# Übergeben und sortieren mehrerer Werte
def min(first, *rest)
  min = first
  rest.each { |x| min = x if x < min }
  min
end

min(10,5,6,7,8,2,1,9)

array = [10,5,6,7,8,2,1,9]
min(*array) # Übergabe des Arrays als einziger Parameter nur mit * möglich!
```

```
# Argumente via Hash übergeben, somit spielt die Reihenfolge keine Rolle!
def zimmer(args)
  zimmer = args[:zimmer_pro_stockwerk] || 10
  stockwerke = args[:stockwerke] || 1
  anzahl = zimmer*stockwerke
end

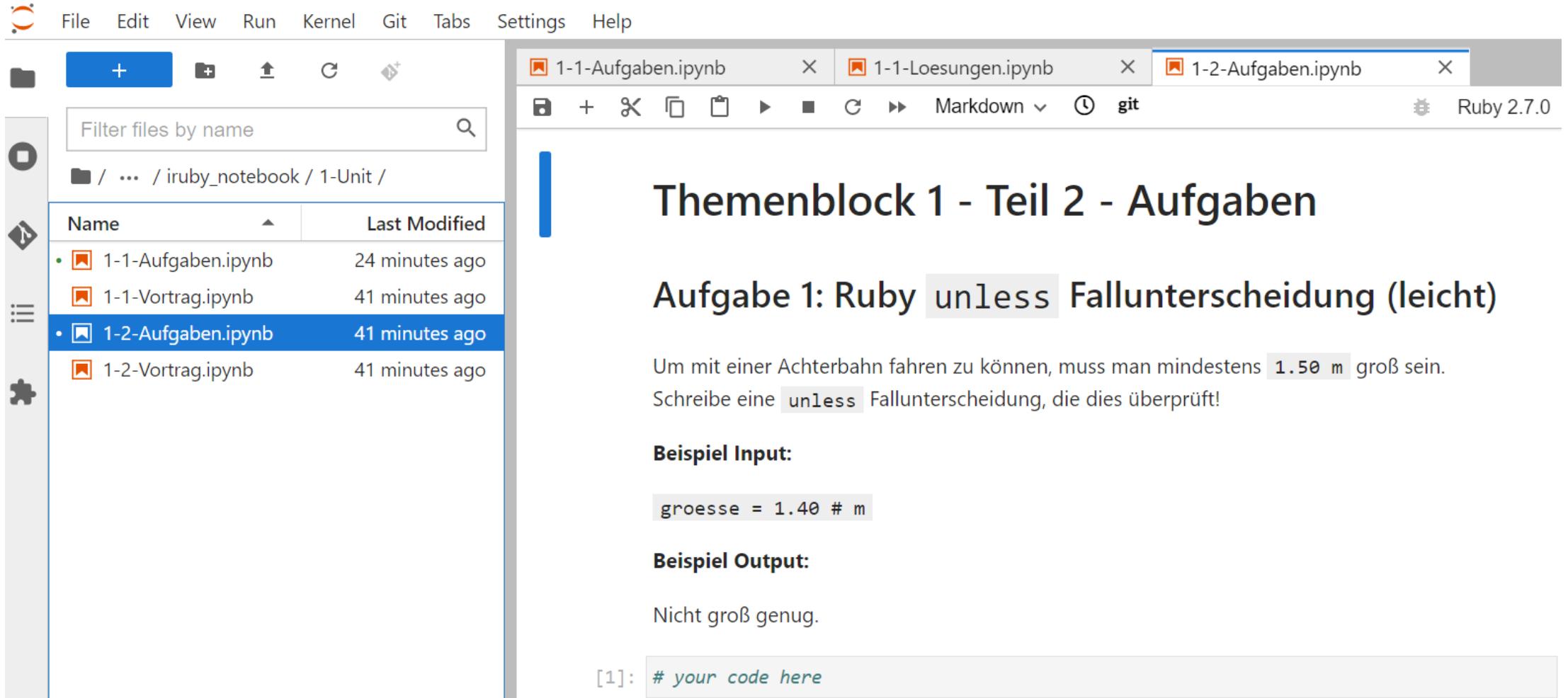
hotel_hash = { :zimmer_pro_stockwerk=>20, :stockwerke=>3 }

zimmer(hotel_hash)
# zimmer({:zimmer_pro_stockwerk=>20, :stockwerke=>3})
```

EXCEPTIONS = [FooException, BarException]

```
begin
  a = rand
  if a > 0.5
    raise FooException
  else
    raise BarException
  end
rescue *EXCEPTIONS
  puts "rescued!"
end
```

```
Exception
NoMemoryError
ScriptError
LoadError
NotImplementedError
SyntaxError
SecurityError
SignalException
Interrupt
StandardError -- default for rescue
ArgumentError
UncaughtThrowError
EncodingError
FiberError
IOError
EOFError
IndexError
KeyError
StopIteration
LocalJumpError
NameError
NoMethodError
RangeError
FloatDomainError
RegexpError
RuntimeError -- default for raise
SystemCallError
Errno::*
ThreadError
TypeError
ZeroDivisionError
SystemExit
SystemStackError
```



The screenshot shows a Jupyter Notebook interface. On the left, a file browser displays a directory structure under '/.../iruby_notebook/1-Unit/'. The list includes:

Name	Last Modified
1-1-Aufgaben.ipynb	24 minutes ago
1-1-Vortrag.ipynb	41 minutes ago
1-2-Aufgaben.ipynb	41 minutes ago
1-2-Vortrag.ipynb	41 minutes ago

The tab bar at the top shows three open notebooks: '1-1-Aufgaben.ipynb', '1-1-Loesungen.ipynb', and '1-2-Aufgaben.ipynb'. The status bar indicates 'Ruby 2.7.0'.

Themenblock 1 - Teil 2 - Aufgaben

Aufgabe 1: Ruby unless Fallunterscheidung (leicht)

Um mit einer Achterbahn fahren zu können, muss man mindestens `1.50 m` groß sein.
Schreibe eine `unless` Fallunterscheidung, die dies überprüft!

Beispiel Input:

```
groesse = 1.40 # m
```

Beispiel Output:

Nicht groß genug.

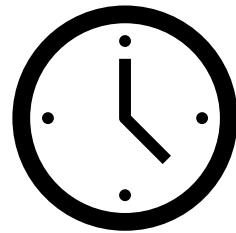
[1]: `# your code here`

QUIZ

Erreichbar unter:

<https://forms.office.com/r/YgfD5CfTQk>





5 Minuten

1. Einführung
2. Klassen & Objekte
3. Instanz Methoden
4. Verkapselung
5. Getter & Setter
6. Klassen Methoden & Variablen
7. Vererbung
8. Mehrfachvererbung



Nach erfolgreichem Abschließen dieses Kapitels solltet ihr in der Lage sein, die **Grundlagen der Objektorientierung und Vererbung in Ruby** anzuwenden.

Vorteile von Objektorientierung

- Probleme in überschaubare Teile kapselbar
 - Konzepte können überdacht werden
 - Modellierung des Programms
- Testen wird vereinfacht
- Verständnis wird erleichtert

→ Vereinfacht Konstruktion robuster Anwendungen

Klassen

- Baupläne, aus denen Objekte erstellt werden
- Abstrakte Vorlagen für Objekte
- enthalten Vorlage für Verhaltensweisen (Methoden) und Daten (Variablen)

Objekte

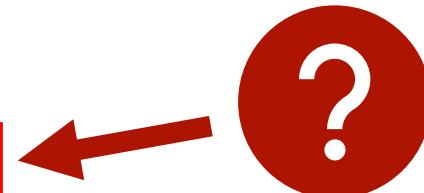
- Instanzen einer Klasse
- Verfügt über die Attribute, die einer Klasse zugeordnet sind

Notation

- Schlüsselwort "**class**" gefolgt von dem Namen der Klasse
- Erster Buchstabe des Klassennamens wird großgeschrieben

Beispiel

```
class User
  def initialize(name)
    @name = name
  end
end
```



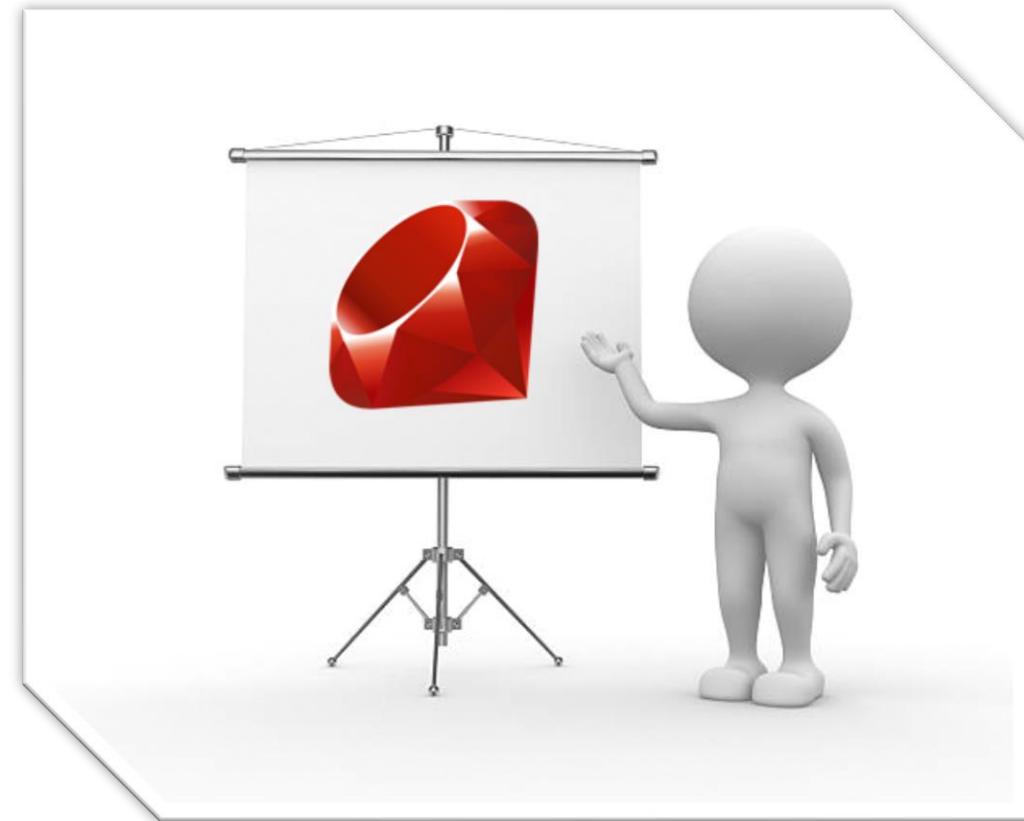
Initialize Methode

- spezielle Methode in Ruby innerhalb der Klassendefinition
- In anderen Sprachen als "Konstruktor" bezeichnet
 - automatisch bei "Instanziierung" aufgerufen
- **Zweck:** Bestimmter Zustand (z. B. Variablen) für jedes instanzierte Objekt initialisieren



Instanz Methode ist automatisch private, auch wenn sie nicht explizit im private Bereich aufgeführt wird!

L I V E



3. Übungsaufgabe – Klassen & Methoden



The screenshot shows a Jupyter Notebook interface. The left sidebar displays a file tree with the following structure:

- / ... / iruby_notebook / 2-Unit /
- Name Last Modified
- 2-1 Aufgaben.ipynb an hour ago
- 2-1-Vortrag.ipynb an hour ago
- 2-2-Aufgaben.ipynb an hour ago
- 2-2-Vortrag.ipynb an hour ago
- vererbung.png

The main notebook area has a tab titled "2-1 Aufgaben.ipynb". The toolbar includes icons for file operations like new, open, save, and run, along with Markdown and git integration. The text content of the notebook is as follows:

Aufgabe 1 - Klassen & Methoden

Erstellen Sie eine Klasse "Car" mit den Attributen Modell, PS und Farbe.

Implementieren Sie innerhalb der Klasse "Car" die Methode "details", welche das Modell, die Farbe und die PS des Autos nach folgendem Schema in der Konsole ausgeben soll.

"Bei dem Auto handelt es sich um einen <model> mit <ps> PS in der Farbe <colour>."

Erstellen Sie zu diesem Zweck eine Instanz der Klasse Car und rufen Sie die Methode Details auf.

```
[1]: # your code here
```

3. Übungsaufgabe – Klassen & Methoden



#Mögliche Lösung:

```
class Car

  #initialize Methode (=Konstruktor)
  def initialize(model, ps, colour)
    @model = model
    @ps = ps
    @colour = colour
  end

  def details
    puts "Bei dem Auto handelt es sich um einen #{@model} mit #{@ps} PS in der Farbe #{@colour}."
  end
end

#Instanz der Klasse Car
car1 = Car.new("Audi A3", 105, "Blau")

puts car1.details
```

Bei dem Auto handelt es sich um einen Audi A3 mit 105 PS in der Farbe Blau.

4. Verkapselung



Konzept auch in vielen anderen Programmiersprachen verankert

Häufig jedoch nur **private & public**

Ruby fügt noch die Verkapselung **protected** hinzu

- Public Methods
 - Standard Level von Methoden in Ruby Klassen
 - Variablen sind standardmäßig private, Methode zum Aufruf ist jedoch public
- Private Methods
 - Zugriff nur von der Klasse selbst möglich
 - Keyword **private** vor der Methode
- Protected Methods
 - Mittelweg zwischen public & private
 - Keyword **protected** vor der Methode

Beispiel: Visibility Rules

1. Erstellen einer Klasse mit drei Methoden:
 - Public
 - Private
 - Protected
2. Erstellen eines Objektes und Aufruf der Methoden

```
# 1.  
class Student  
  def public_method  
    puts "Das ist eine public Methode"  
  end  
  protected  
  def protected_method  
    puts "Das ist eine protected Methode"  
  end  
  private  
  def private_method  
    puts "Das ist eine private Methode"  
  end  
end  
  
#2.  
Stud = Student.new  
Stud.public_method;  
Stud.protected_method;  
Stud.private_method;
```

4. Verkapselung



```
# 1.  
class Student  
  def public_method  
    puts "Das ist eine public Methode"  
  end  
  protected  
  def protected_method  
    puts "Das ist eine protected Methode"  
  end  
  private  
  def private_method  
    puts "Das ist eine private Methode"  
  end  
end  
  
#2.  
Stud = Student.new  
Stud.public_method;  
Stud.protected_method;  
Stud.private_method;
```

Das ist eine public Methode

```
NoMethodError: protected method `protected_method'  
called for #<#<Class:0x0000563c57447520>::Student:0  
x0000563c572e8fa8>  
Did you mean? protected_methods  
(irb):18:in `irb_binding'
```

Beispiel: Visibility Rules

3. Betrachten den bisherigen Code
4. Fügen weitere Methode hinzu, welche die anderen Methoden aufruft
5. Erstellen eines weiteren Objektes und Aufruf der neuen Methode

```
#3.  
class Student  
  #4.  
  def call_each  
    public_method  
    protected_method  
    private_method  
  end  
  def public_method  
    puts "Das ist eine public Methode"  
  end  
  protected  
  def protected_method  
    puts "Das ist eine protected Methode"  
  end  
  private  
  def private_method  
    puts "Das ist eine private Methode"  
  end  
end  
  
#5.  
t = Student.new;  
t.call_each;
```

4. Verkapselung



```
#3.  
class Student  
#4.  
  def call_each  
    public_method  
    protected_method  
    private_method  
  end  
  def public_method  
    puts "Das ist eine public Methode"  
  end  
protected  
  def protected_method  
    puts "Das ist eine protected Methode"  
  end  
private  
  def private_method  
    puts "Das ist eine private Methode"  
  end  
end  
  
#5.  
t = Student.new;  
t.call_each:
```

Das ist eine public Methode
Das ist eine protected Methode
Das ist eine private Methode

Beispiel: Visibility Rules

6. Betrachten den bisherigen Code
7. Anpassen der call_each Methode:
Hinzufügen des Schlüsselwortes self

```
#6.  
class Student  
  def call_each  
    #7.  
    self.public_method  
    self.protected_method  
    self.private_method  
  end  
  def public_method  
    puts "Das ist eine public Methode"  
  end  
  protected  
  def protected_method  
    puts "Das ist eine protected Methode"  
  end  
  private  
  def private_method  
    puts "Das ist eine private Methode"  
  end  
end  
  
t = Student.new;  
t.call_each;
```

4. Verkapselung



```
#6.  
class Student  
  def call_each  
    #7.  
    self.public_method  
    self.protected_method  
    self.private_method  
  end  
  def public_method  
    puts "Das ist eine public Methode"  
  end  
  protected  
  def protected_method  
    puts "Das ist eine protected Methode"  
  end  
  private  
  def private_method  
    puts "Das ist eine private Methode"  
  end  
end  
  
t = Student.new;  
t.call_each;
```

```
Das ist eine public Methode  
Das ist eine protected Methode  
main.rb:7:in `call_each': private method `private_method' called for #<Student:0x005582f6ec6278> (NoMethodError)  
Did you mean?  private_methods  
              from main.rb:23:in `<main>'
```

5. Getter & Setter



- Variablen von Objekten holen und setzen
- Ruby bietet verschiedene Möglichkeiten
- = verwenden um zu setzen
- attr_accessor als Schlüsselwort

```
class Student
  def initialize(name)
    @name = name
  end
  def name
    @name
  end
  def name=(new_name)
    @name = new_name
  end
end

s = Student.new("Heinz")
puts(s.name)

s.name = "Franz"
print(s.name)
```

Heinz
Franz

attr_accessor als Schlüsselwort

Für Accessor-Methoden einer Instanz

- Sowohl get als auch set
- Mehrere getter & setter können gleichzeitig erstellt werden → Symbole/Attribute auflisten
- attr_reader – für nur get
- attr_writer – für nur set

5. Getter & Setter



```
class Student
    attr_accessor :name, :matrikelnummer, :age

    def initialize(name)
        @name = name
    end
end

s = Student.new("Bernd")
s.name = "Lisa"
s.matrikelnummer = 12345
s.age = 20

puts(s.name, s.matrikelnummer, s.age)
```

5. Getter & Setter



```
class Student
    attr_accessor :name, :matrikelnummer, :age

    def initialize(name)
        @name = name
    end

end

s = Student.new("Bernd")
s.name = "Lisa"
s.matrikelnummer = 12345
s.age = 20

puts(s.name, s.matrikelnummer, s.age)
```

```
Lisa
12345
20
```

5. Getter & Setter



```
class Teacher
  attr_reader :name
  attr_writer :subject

  def initialize(name)
    @name = name
  end
end

t = Teacher.new("Albert")

#reader/get möglich
puts(t.name)

#writer erstellt, set möglich, get nicht möglich
t.subject = "Fach"
puts(t.subject)

#kein writer, nur reader, set nicht möglich
t.name = "Heinz"
puts(t.name)
```

5. Getter & Setter



```
class Teacher
  attr_reader :name
  attr_writer :subject

  def initialize(name)
    @name = name
  end
end

t = Teacher.new("Albert")

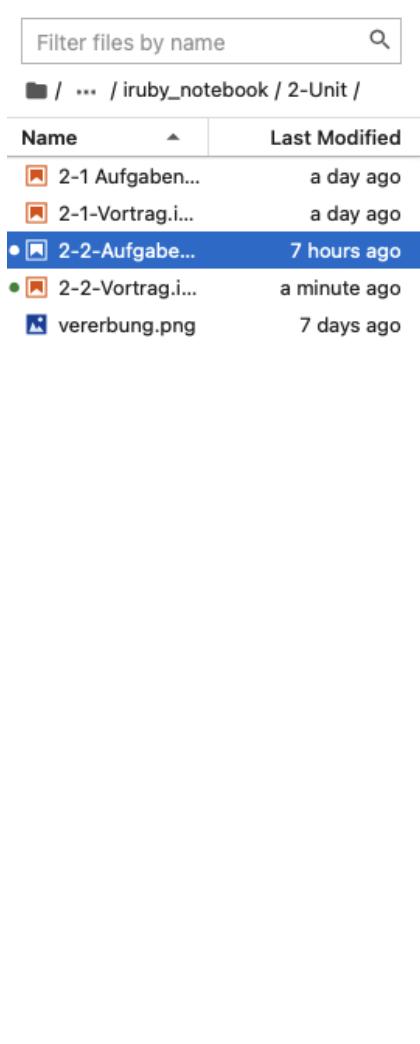
#reader/get möglich
puts(t.name)

#writer erstellt, set möglich, get nicht möglich
t.subject = "Fach"
puts(t.subject)

#kein writer, nur reader, set nicht möglich
t.name = "Heinz"
puts(t.name)
```

Albert

```
NoMethodError: undefined method `subject' for #<
#<Class:0x0000555a0111edc8>::Teacher:0x0000555a0
071ccf8>
Did you mean?  subject=
(irb):16:in `irb_binding'
```



Filter files by name

File browser interface showing the following files:

Name	Last Modified
2-1 Aufgaben...	a day ago
2-1-Vortrag.i...	a day ago
2-2-Aufgabe...	7 hours ago
2-2-Vortrag.i...	a minute ago
vererbung.png	7 days ago

2-2-Aufgaben

Übung Getter & Setter

Mit dieser Übung vertiefst du deine Kenntnisse zu Getter & Setter in Ruby.

1. Erstelle insgesamt 3 Klassen: User, Company und Department.
2. Füge folgende attr_accessor und Initialisierungen hinzu:
 - User: name, address, department - name, address, department
 - Company: name, url, department - name, url als URI.parse(url)
 - Department: name, users - name
3. Erstelle eine Company mit dem Konstruktor, z. B. mit dem Name "HS Mannheim" und der url "www.hs-mannheim.de"
4. Erstelle zwei Abteilungen mit unterschiedlichen, beliebigem Name.
5. Erstelle mind. 2 User. Bei der Erstellung des Users trage eine der zuvor erstellten Abteilungen ein.
6. Ordne nun der Company die erstellten Abteilungen zu.

Nun wollen wir den Use Case betrachten, dass ein Unternehmen eine Liste über alle User erhalten möchte. Das Unternehmen hat bereits Zugriff auf die Departments.

7. Erstelle eine Methode, die über alle User je Department durch iteriert.

Hierbei ergibt sich folgendes Problem. Wir erhalten ein Array mit Arrays. Ruby liefert hierfür die .flatten Methode. Diese Methode erstellt aus allen Items ein Ein-dimensionales Array.

8. Füge dahier .flatten der Methode hinzu.

Die Department Klasse besitzt einen accessor für :users. Dies kann jedoch nil sein und dementsprechend bei der Ausgabe einen Fehler verursachen. Um diesen Fehler abzufangen, passen wir die Initialisierung an, indem wir bei Objekt Generierung ein leeres Users-Array erstellen.

5. Getter & Setter



```
#1..2.
class User
attr_accessor :name, :address, :department

def initialize(name, address, department)
  @name = name
  @address = address
  @department = department
#9.
  @department.users << self
end
end

class Company
attr_accessor :name, :url, :departments

def initialize(name, url)
  @name = name
  @url = URI.parse(url)
end
#7. & 8.(.flatten hinzugefügt)
def users
  departments.map(&:users).flatten
end
end
```

```
#8. @users = []
class Department
attr_accessor :name, :users

def initialize(name)
  @name = name
  @users = []
end
end

#3.
c = Company.new("HS Mannheim", "www.hs-mannheim.de")

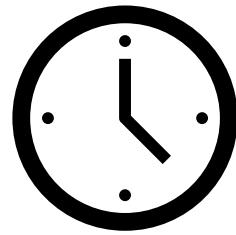
#4.
sales = Department.new("Sales")
engineering = Department.new("Engineering")

#5.
bob = User.new("Bob Smith", "1 Main Street", sales)
mary = User.new("Mary Jane", "10 Another Street", engineering)

#6.
c.departments = [sales, engineering]

#10
print c.users
```

```
[#<#<Class:0x00005592fdc31870>::User:0x00005592fdade3b0 @name="Bob Smith", @address="1 Main Street", @department=#<#<Class:0x00005592fdc31870>::Department:0x00005592fdade568 @name="Sales", @users=[#<#<Class:0x00005592fdc31870>::User:0x00005592fdade3b0 ...]>>, #<#<Class:0x00005592fdc31870>::User:0x00005592fdade2c0 @name="Mary Jane", @address="10 Another Street", @department=#<#<Class:0x00005592fdc31870>::Department:0x00005592fdade4a0 @name="Engineering", @users=[#<#<Class:0x00005592fdc31870>::User:0x00005592fdade2c0 ...]>>]
```



5 Minuten

6. Klassen Methoden & Variablen



Variablen & Methoden, die auf Klassenebene arbeiten

Klassenmethoden und –variablen:

- Unveränderliche Daten/Methoden
- Gehören zu Klassen
- Können auf alle Instanzen der Klasse angewandt werden

Klassenvariablen beginnen mit @@

Häufigste Klassenmethode: .new Konstruktor

Zwei unterschiedliche Schreibweisen für Klassenmethoden

- Methode 1:
Definieren der Methode mit
`def self.method`
- Methode 2:
Definieren der Methode(n)
mit `class << self`
- Auswahl ist abhängig von der
Anzahl zu definierender
Klassenmethoden

```
class Example
  ##Methode 1
  def self.hello(str)
    puts "Hello #{ str }!"
  end

  ##Methode 2
  class << self
    def hello2(str)
      puts "And hello #{ str }!"
    end
    def hello3
      puts "Mehrere Klassenmethoden sind möglich"
    end
  end
end

Example.hello("World")
Example.hello2("IWS Teilnehmer")
Example.hello3
```

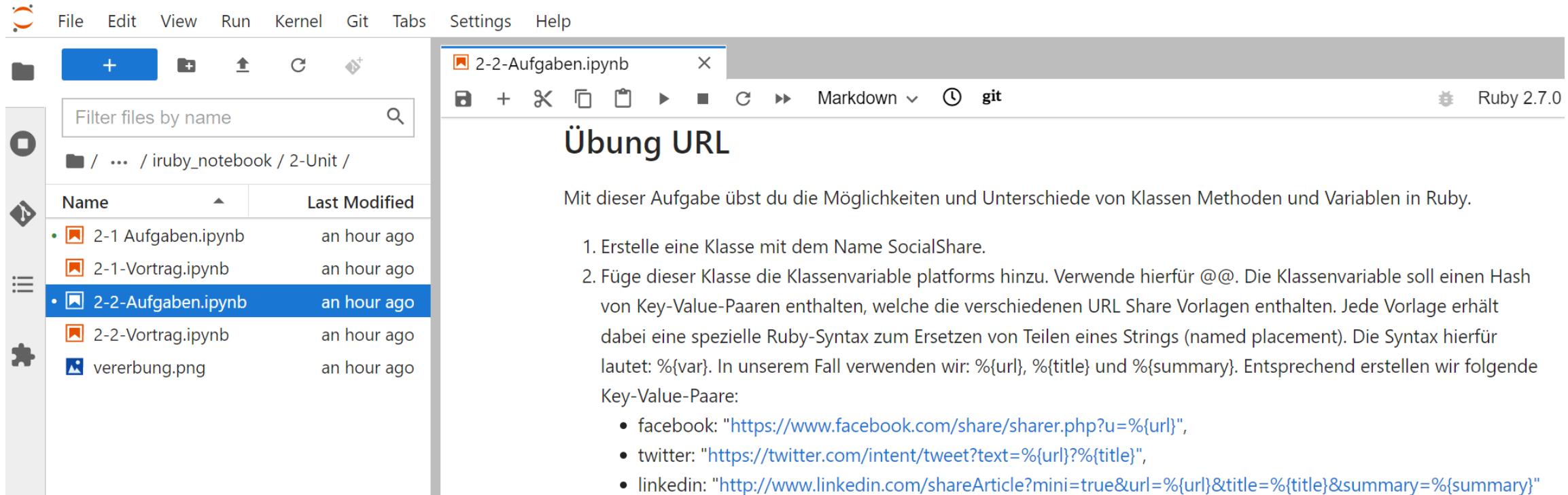
- Methode 1:
Definieren der Methode mit
`def self.method`
- Methode 2:
Definieren der Methode(n)
mit `class << self`
- Auswahl ist abhängig von der
Anzahl zu definierender
Klassenmethoden

```
class Example
  ##Methode 1
  def self.hello(str)
    puts "Hello #{ str }!"
  end

  ##Methode 2
  class << self
    def hello2(str)
      puts "And hello #{ str }!"
    end
    def hello3
      puts "Mehrere Klassenmethoden sind möglich"
    end
  end
end

Example.hello("World")
Example.hello2("IWS Teilnehmer")
Example.hello3

Hello World!
And hello IWS Teilnehmer!
Mehrere Klassenmethoden sind möglich
```



The screenshot shows a Jupyter Notebook interface. On the left, there is a file browser with a sidebar containing icons for file operations like new file, upload, and refresh. A search bar labeled "Filter files by name" is present. The main area displays a list of files in the directory "/ ... / iruby_notebook / 2-Unit /". The file "2-2-Aufgaben.ipynb" is selected and highlighted in blue. The list includes:

Name	Last Modified
2-1 Aufgaben.ipynb	an hour ago
2-1-Vortrag.ipynb	an hour ago
2-2-Aufgaben.ipynb	an hour ago
2-2-Vortrag.ipynb	an hour ago
vererbung.png	an hour ago

The notebook cell on the right is titled "2-2-Aufgaben.ipynb" and contains the following text:

Übung URL

Mit dieser Aufgabe übst du die Möglichkeiten und Unterschiede von Klassen Methoden und Variablen in Ruby.

1. Erstelle eine Klasse mit dem Name SocialShare.
2. Füge dieser Klasse die Klassenvariable platforms hinzu. Verwende hierfür @@. Die Klassenvariable soll einen Hash von Key-Value-Paaren enthalten, welche die verschiedenen URL Share Vorlagen enthalten. Jede Vorlage erhält dabei eine spezielle Ruby-Syntax zum Ersetzen von Teilen eines Strings (named placement). Die Syntax hierfür lautet: %{var}. In unserem Fall verwenden wir: %{url}, %{title} und %{summary}. Entsprechend erstellen wir folgende Key-Value-Paare:
 - facebook: "<https://www.facebook.com/share/sharer.php?u=%{url}>",
 - twitter: "<https://twitter.com/intent/tweet?text=%{url}?%{title}>",
 - linkedin: "<http://www.linkedin.com/shareArticle?mini=true&url=%{url}&title=%{title}&summary=%{summary}>"

6. Klassen Methoden & Variablen



Ruby

```
#1.
class SocialShare
#2.
@@platforms = {
  facebook: "https://www.facebook.com/share/sharer.php?u=%{url}",
  twitter: "https://twitter.com/intent/tweet?text=%{url}/%{title}",
  linkedin: "http://www.linkedin.com/shareArticle?mini=true&url=%{url}&title=%{title}&summary=%{summary}"
}

#3.
def initialize(platform, title, url, summary= "")
  @platform = platform.to_sym

  @share_url = url

  @title = title

  @summary = summary
end

#5.
def url
  @url ||= generate_url
end
```

```
#4.
def generate_url
  @@platforms[@platform] % {title: @title, url: @share_url, summary: @summary}
end

#6.
def self.url(platform, title, url, summary = "")
  new(platform, title, url, summary).url
end

#9.
def self.get_links_for_all(title, url, summary)
  @@platforms.keys.map do |platform|
    url(platform, title, url, summary)
  end
end

#7.
platform = "facebook"

title = "InformatikWorkshop"
url = "www.hs-ma.de"
summary = "IWS"

#8.
puts SocialShare.url(platform, title, url, summary)

#10.
puts SocialShare.get_links_for_all(title, url, summary)
```

<https://www.facebook.com/share/sharer.php?u=www.hs-ma.de>

<https://www.facebook.com/share/sharer.php?u=www.hs-ma.de>

<https://twitter.com/intent/tweet?text=www.hs-ma.de/InformatikWorkshop>

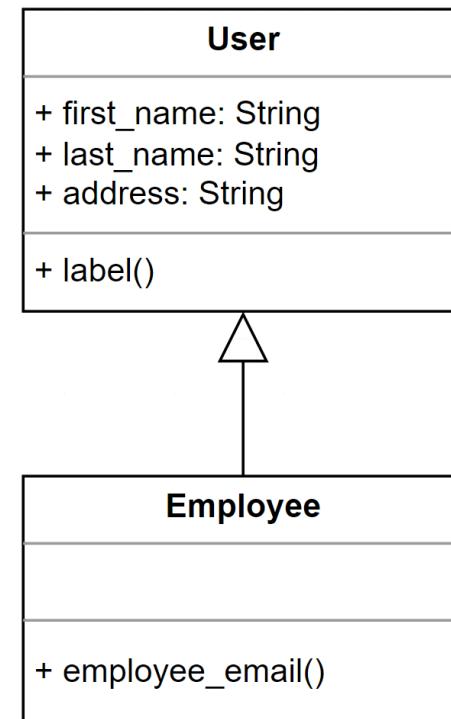
<http://www.linkedin.com/shareArticle?mini=true&url=www.hs-ma.de&title=InformatikWorkshop&summary=IWS>

Grundlagen

- Klassen als Vorlage für andere Klassen
→ Subklassen

Beispiel

- Gemeinsame Eigenschaften eines Users
 - Vorname
 - Nachname
 - Adresse
- Subklasse:
 - Angestellter



7. Vererbung

Beispiel:

```
# Superklasse
class User

attr_accessor :first_name, :last_name, :address

def initialize(first_name, last_name, address)
  @first_name = first_name
  @last_name = last_name
  @address = address
end

def label
  "#{first_name} #{last_name}"
end
```

```
# Subklasse
class Employee < User

def employee_email
  # first.last_name@example.com
  "#{@first_name}.#{@last_name[0]}@example.com"
end

e = Employee.new("Bob", "Smith", "1 Main street")
puts e.label
puts e.employee_email
Bob Smith
Bob.S@example.com
```

Super

- Anpassen von Verhalten und Variablen
- "*Super*" aufrufen der übergeordneten Methode
- 3 Möglichkeiten "*Super*" in Ruby aufzurufen
 - ohne Argumente aufrufen
 - mit Argumenten anrufen
 - mit dem Splat-Operator (*) aufrufen

Beispiel – 3. Super mit dem Splat-Operator

```
class User

  attr_accessor :name, :address

  def initialize(name, address)
    puts "In User#initialize"
    @name = name
    @address = address
  end

end
```

In User#initialize

```
#<#<Class:0x000055edd70dc5d0>::Employee:0x000055edd69b8448 @address="125 Main Street", @on_payroll=true, @name="Bob">
```

```
class Employee < User

  attr_accessor :on_payroll

  def initialize(*)
    @on_payroll = true
    super
  end

end
```

```
e2 = Employee.new("Bob", "125 Main Street")
```

Grundlagen

- Eine Klasse kann von mehr als einer Basisklasse erben
- Ruby unterstützt keine Mehrfachvererbung!
 - Lösung: Wiederverwendbarkeit von Code mittels Modulen

Module

- Code umschließen, um ihn mit anderen Codeabschnitten zu teilen
- Module können dabei in anderen Code
 - Eingefügt
 - Erweitert
 - Vorangestellt werden

Module != Klassen

- Module können nicht instanziert werden!
- Module können auf Variablen & Methoden von Klassen zugreifen
- Klassen können auf Variablen & Methoden von Modulen zugreifen

Eingefügte Module

- Code bündeln um ihn in anderem Code wiederzuverwenden
- Schlüsselwort: "**include**"

8. Mehrfachvererbung

Beispiel – Eingefügte Module

```
module Address

attr_accessor :address_line1, :address_line2, :city,
:state, :postal_code, :country

def mailing_label
  label = []
  label << @address_line1
  label << @address_line2
  label << "#{@city}, #{@state} #{@postal_code}"
  label << @country
  label.join("\n")
end

end

class User_multiple_inheritance
  include Address
end

class Building
  include Address
end
```

```
u = User_multiple_inheritance.new
b = Building.new
u.address_line1 = "123 Main Street"
b.address_line1 = "987 Broadway"

puts "User-Instanz:"
puts u.instance_variable_get("@address_line1")
puts "Objekt ID: #{u.instance_variable_get("@address_line1").object_id}"

puts "Building-Instanz:"
puts b.instance_variable_get("@address_line1")
puts "Objekt ID: #{b.instance_variable_get("@address_line1").object_id}"

User-Instanz:
123 Main Street
Objekt ID: 49260

Building-Instanz:
987 Broadway
Objekt ID: 49280
```

8. Übungsaufgabe – Vererbung



File Edit View Run Kernel Git Tabs Settings Help

2-1 Aufgaben.ipynb x

Filter files by name

/ ... / iruby_notebook / 2-Unit /

Name	Last Modified
2-1 Aufgaben.ipynb	an hour ago
2-1-Vortrag.ipynb	an hour ago
2-2-Aufgaben.ipynb	an hour ago
2-2-Vortrag.ipynb	an hour ago
vererbung.png	an hour ago

Aufgabe 2 - Vererbung

Erstellen Sie eine Klasse "Plant" und eine Klasse "Tree".

Die Klasse Plant kann mit einem Namen initialisiert werden, der über eine get-Methode ausgelesen werden kann. Zusätzlich dazu besitzt die Klasse Plant eine Methode "naming", die den Namen der Instanz als String in folgender Form zurückgibt.

"Die Pflanze trägt den Namen <name>."

Die Klasse Tree erbt den Namen der Klasse Plant und besitzt zudem ein weiteres Attribut "has_needles", welches boolesch angeben soll, ob ein Baum Nadelblätter besitzt oder nicht.

Erweitern Sie die Klasse Tree durch die Methode "grow", achten Sie dabei auf die Möglichkeit denselben Code auch in anderen Klassen zu verwenden, die gegebenenfalls unterschiedliche Elternklassen haben. Die Methode soll dabei die Höhe der Pflanze nach folgendem Schema berechnen und zurückgeben:

Höhe = Wachstumsrate_proJahr + Höhe

Werte der Variablen:

- Wachstumsrate_proJahr = 1m
- Höhe = 0m

Abschließend erstellen Sie bitte eine Instanz der Klasse Tree, geben dessen Namen mit der dafür vorgesehenen Methode naming aus und lassen ihn zweimal wachsen. Die Höhe des Baums geben Sie anschließend bitte in der Konsole aus.

[2]: # your code here

8. Übungsaufgabe – Vererbung



#Mögliche Lösung:

```
#Modul Growable zur Wiederverwendung der Methode grow innerhalb anderer Klassen
module Growable
  def grow
    @height ||= 0
    @growth_rate_pa ||= 1

    @height += @growth_rate_pa
    self
  end
end

class Plant
  #Get-Methode
  attr_reader :name

  #initialize Methode (=Konstruktor)
  def initialize(name)
    @name = name
  end

  def naming
    puts "Die Pflanze trägt den Namen #{name}."
  end
end
```

#Tree erbt durch die Zuordnung "<" von der Klasse Plant

```
class Tree < Plant
  #Get-Methode
  attr_reader :height

  #initialize Methode (=Konstruktor)
  def initialize(name, has_needles)
    super(name)
    @has_needles = has_needles
  end

  #Inkludierung des Moudls Growable zur Nutzung der grow-Methode
  include Growable
end

#Instanz der Klasse Tree
t1 = Tree.new("Tanne", true)

#Zugriff auf die Methode naming der Klasse Plant durch
#die Vererbung dieser an die Klasse Tree
t1.naming

#Zugriff auf die grow-Methode des Moduls Growable
t1.grow
t1.grow

puts "Die Höhe des Baumes beträgt: #{t1.height}m."
```

Die Pflanze trägt den Namen Tanne.

Die Höhe des Baumes beträgt: 2m.

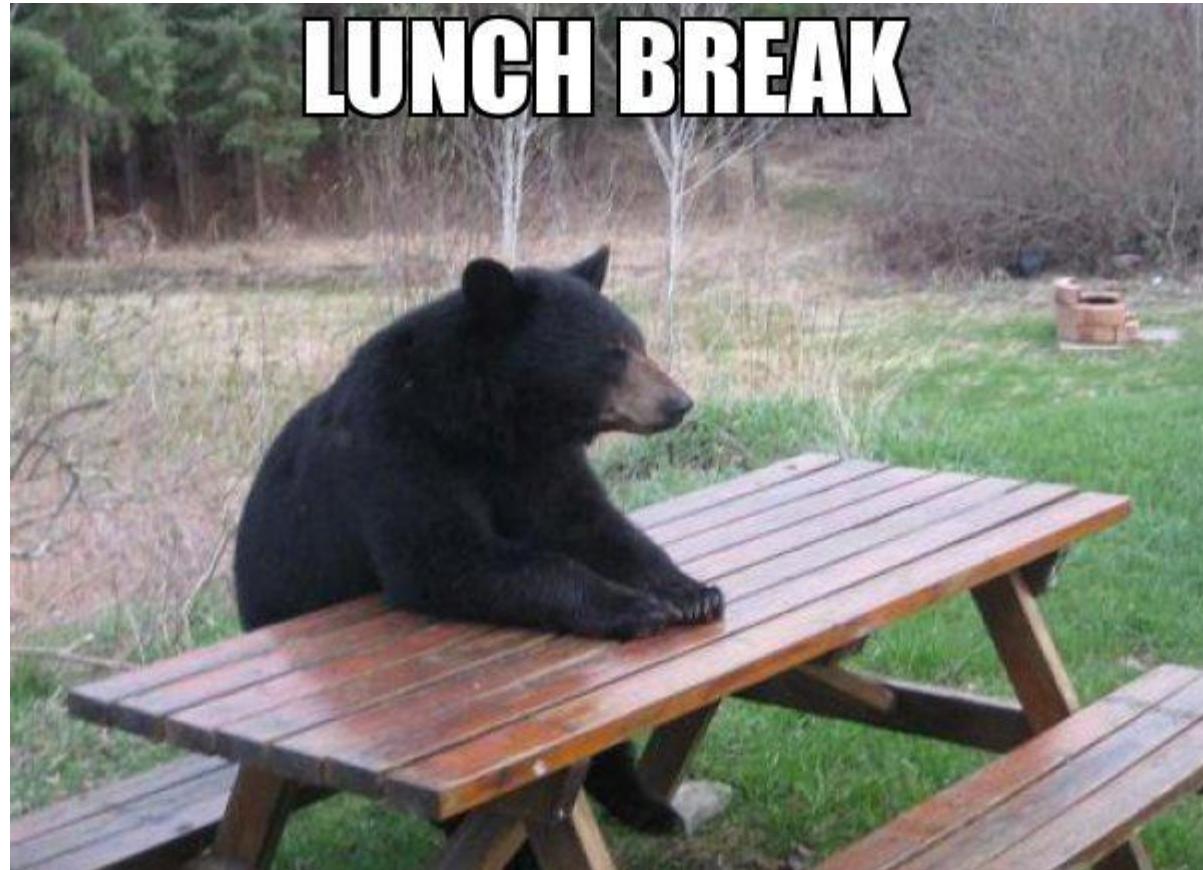
QUIZ

Erreichbar unter:

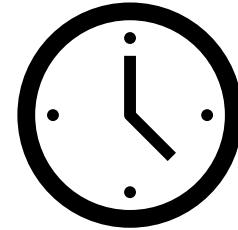
<https://forms.office.com/r/AbLmbTE58Q>



<i>jetzt</i>	Lunch Break
13:25 – 14:25 Uhr	Unit 3: Weiterführendes
	Coffee Break
14:30 – 14:50 Uhr	Ausblick
14:50 – 15:00 Uhr	Feedback Session & Quiz Auswertung
	Schluss



LUNCH BREAK



45 Minuten

1. Files (input / output)
2. Funktionale Programmierung
3. Lamdas
4. Yield
5. Ruby on Rails
6. Gems
7. Letzte Chance für Fragen



Nach erfolgreichem Abschließen dieses Kapitels, könnt ihr **mit Dateien umgehen und funktional programmieren.**

Dateimodi

r nur lesen

r+ lesen und schreiben

w nur schreiben

w+ schreiben und lesen

a nur schreiben (Zeiger am Ende)

a+ lesen und schreiben (Zeiger am Ende)

b binärer Dateimodus

```
File.open("dateiname.txt", "modus")
```

```
file = File.open("dateiname.txt")
```

```
file = File.open("dateiname.txt", "modus")
```

```
file.closed?
```

```
file.close
```

```
File.rename("alter_name.txt", "neuer_name.txt")  
File.delete("dateiname.txt")
```

```
File.exists?("dateiname.txt")  
File.file?("dateiname.txt")  
File.directory?("dateiname.txt")  
File.readable?("dateiname.txt")  
File.writeable?("dateiname.txt")  
File.executable?("dateiname.txt")
```

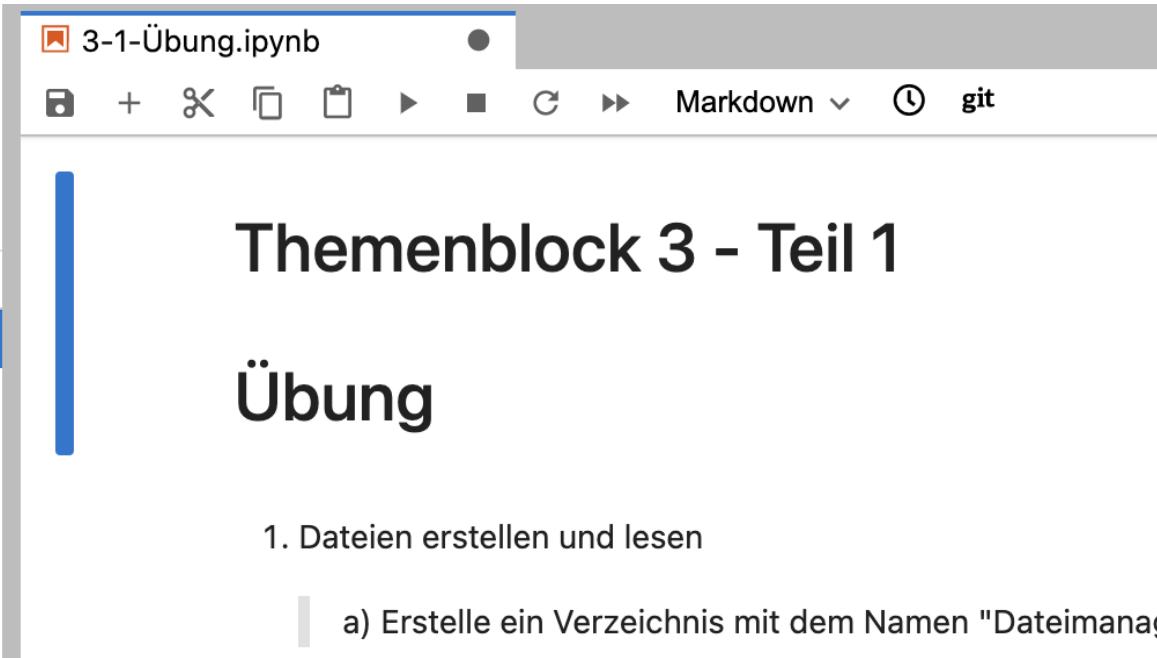
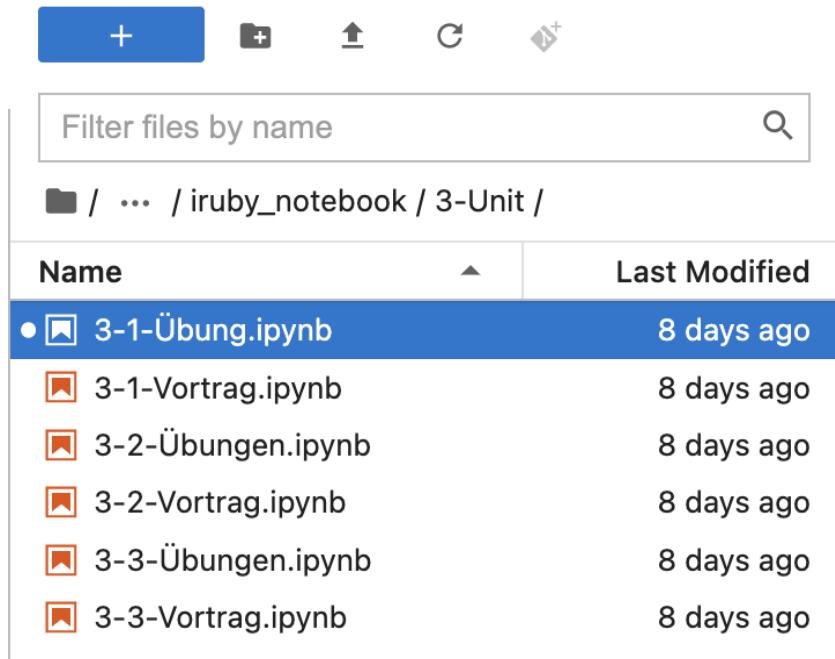
```
File.size("dateiname.txt")  
File.zero?("dateiname.txt")  
File.ftype("dateiname.txt")  
File.ctime("dateiname.txt")  
File.mtime("dateiname.txt")  
File.atime("dateiname.txt")
```

Lesen

```
file_data = file.read  
file_data = file.readline  
file_data = file.readlines  
  
file_data = File.read("dateiname.txt")  
file_data = File.read("dateiname.txt").split  
  
File.foreach("dateiname.txt") { |line|  
    puts line  
}
```

Schreiben

```
file.puts("Das ist eine Testzeile")  
  
file.rewind
```



Themenblock 3 - Teil 1

Übung

1. Dateien erstellen und lesen
 - a) Erstelle ein Verzeichnis mit dem Namen "Dateimanag...

Dateien in Ruby – Lösungen



```
folder_name = "Dateimanagement";
file_name = folder_name + "/First_File.txt";

# a)
if !Dir.exists?(folder_name)
  Dir.mkdir(folder_name)
end

# b)
if !File.exists?(file_name)
  File.open(file_name, "w") { |file|
    # c)
    puts !file.closed?

    # d)
    file.puts("Das ist Zeile 1")
    file.puts("Und das ist Zeile 2")
    file.puts("Ich bin die letzte Zeile, also 3")
  }
end

# e)
File.foreach(file_name) { |line|
  puts line
}
puts File.open(file_name, "r").read

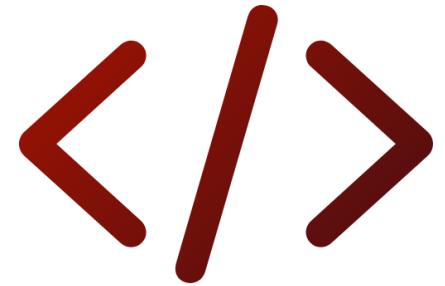
# f)
puts File.exists?(file_name)
puts File.ftype(file_name)
puts File.ftype(folder_name)
puts File.size(file_name)
```

- Was ist Funktionale Programmierung (FP)?
- Vorteile
- FP in Ruby
 - Dont's und Do's
 - Blöcke
- Übungen zu FP
- Fazit

- Programmierparadigma
- Vermeidung von Zuständen und veränderlichen Daten

Einfach

- Förderung von Code **ohne** Seiteneffekte und **ohne** Änderung des Wertes von Variablen
- Gegensatz dazu: Imperative Programmierung



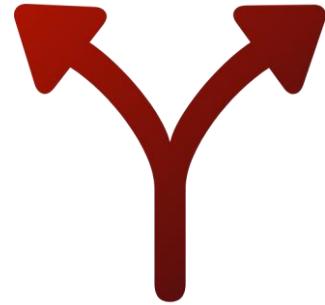
Cleaner Code



Kein Verfolgen von
Zustandsänderungen



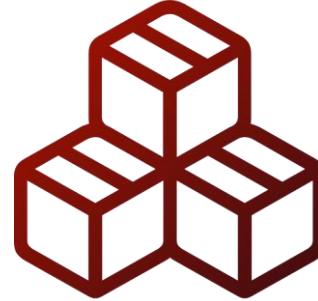
Referentielle
Transparenz



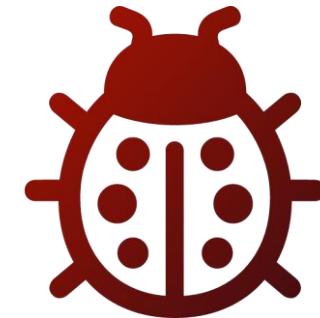
Parallelisierung



Memoization



Modularisierung



Debugging

Und was hat das mit Ruby zu tun?

- Ruby ist keine funktionale Programmiersprache
- FP ist ein Stil
- FP sollte immer dann verwendet werden, wenn es die Qualität des Codes erhöht

Variablen updaten

```
name = "Kaiser Wilhelm"
```

```
new_name = "Kaiser Wilhelm II"
```

Neue Elemente in Arrays hinzufügen

```
indexes = [1, 2, 3]
indexes << 4
# => [1, 2, 3, 4]
```

```
indexes = [1, 2, 3]
all_indexes = indexes + [4]
# => [1, 2, 3, 4]
```

Hashes updaten

```
hash = { :a => 1, :b => 2 }
hash[:c] = 3
# => {:a=>1, :b=>2, :c=>3}
```

```
hash = { :a => 1, :b => 2 }
new_hash = hash.merge(:c => 3)
# => {:a=>1, :b=>2, :c=>3}
```

Ersetzungsmethoden verwenden

```
string = "hallo"  
string.gsub!(/l/, 'z')  
# => "hazzo"
```

```
string = "hallo"  
new_string = string.gsub(/l/, 'z')  
# => "hazzo"
```

Werte zusammenführen

```
output = []
output << 1
output << 2 if i_have_to_add_two
output << 3
```

```
output = [1, (2 if i_have_to_add_two), 3].compact
```

Wenn eine Sprache funktional verwendet werden soll,
benötigt man Funktionen einer höheren Ordnung

- Ruby ist in dieser Hinsicht besonders
- Ein Block ist ein anonymes Stück Code
 - kann weitergeben und
 - nach Belieben ausgeführt werden

```
dogs = []
["milu", "gustav"].each { |name|
  dogs << name.upcase
}
dogs # => ["MILU", "GUSTAV"]
```

```
dogs = ["milu", "gustav"].map { |name|
  name.upcase
}
# => ["MILU", "GUSTAV"]
```

```
dogs = []
["milu", "gustav"].each { |name|
  if name.size == 4
    dogs << name
  end
}
dogs # => ["milu"]
```

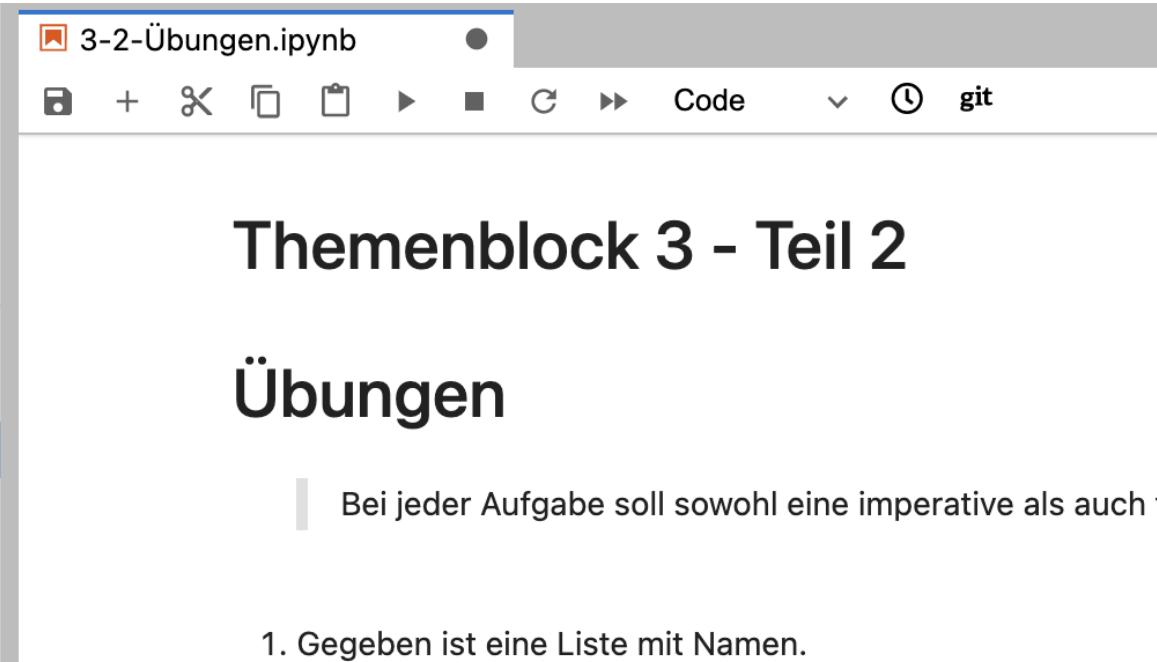
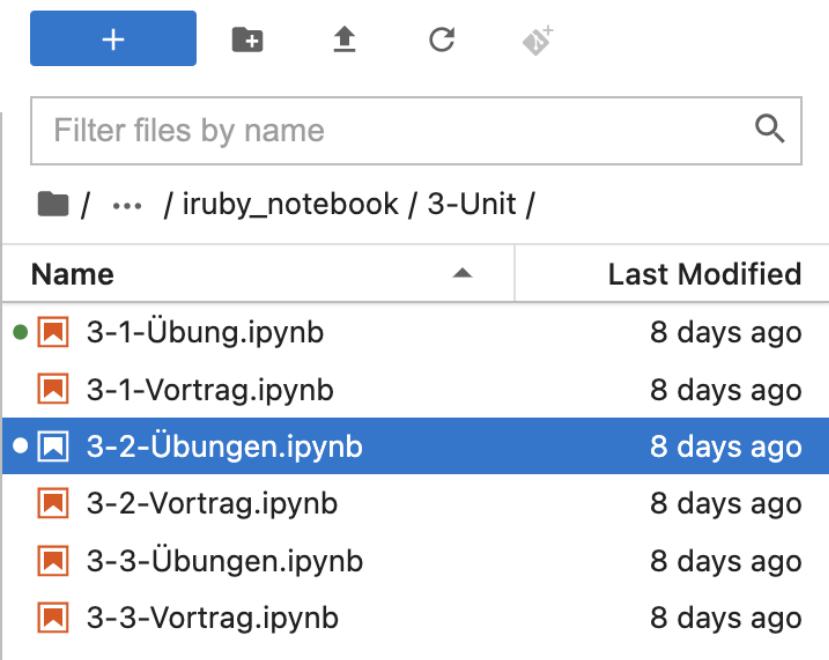
```
dogs = ["milu", "gustav"].select { |name|
  name.size == 4
}
# => ["milu"]
```

```
length = 0
["milu", "gustav"].each { |dog_name|
  length += dog_name.length
}
length # => 10
```

```
length = ["milu", "gustav"]
.inject(0) { |accumulator, dog_name|
  accumulator + dog_name.length
}
# => 10
```

```
length = ["milu", "gustav"]
.inject(0) { |accumulator, dog_name|
  accumulator + dog_name.length
}
# => 10
```

```
length = ["milu", "gustav"]
.map(&:length)
.inject(0, :+)
# => 10
```



Themenblock 3 - Teil 2

Übungen

Bei jeder Aufgabe soll sowohl eine imperative als auch f

1. Gegeben ist eine Liste mit Namen.

Funktionale Programmierung – Lösungen



Ruby

```
# a) Array sollte die Länge 5 haben und mit Rupert beginnen
filtered = []
for n in names
  if n.split(" ").length > 2
    filtered << n
  end
end

# b)
leng = filtered.length - 1
for i in 0...leng
  for j in 0...(leng - i)
    compare = filtered[j].split(" ")[1] <=> filtered[j+1].split(" ")[1]
    if compare == 1 then
      tmp = filtered[j];
      filtered[j] = filtered[j+1];
      filtered[j+1] = tmp
    end
  end
end
```

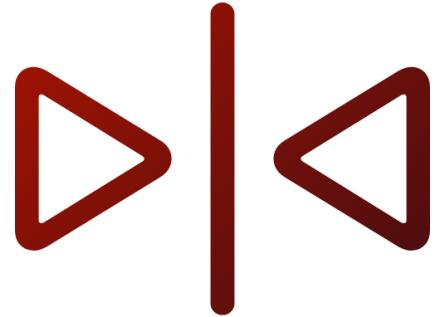
```
# a) Array sollte die Länge 5 haben und
#     mit Rupert beginnen
names.select { |n|
  n.split(" ").length > 2
}
# b)
.sort { |prev, curr|
  prev.split(" ")[1] <=> curr.split(" ")[1]
}
```

Imperativ 275

```
n, num_elements, sum = 1, 0, 0
while num_elements < 10
  if n**2 % 5 == 0
    sum += n
    num_elements += 1
  end
  n += 1
end
```

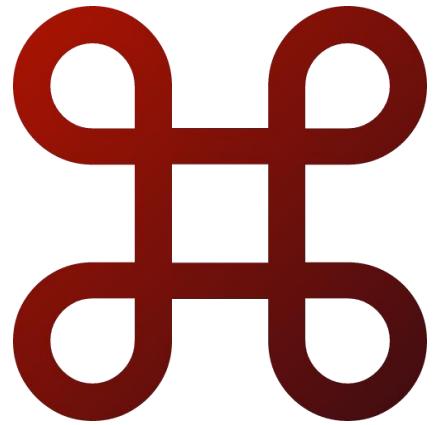
Funktional 275

```
(1...999).select { |x| x**2 % 5 == 0 }
  .take(10)
  .inject(:+)
```



Kompaktheit

weniger Code
Verkettung



Abstraktion

Code wird versteckt
Reduktion der Komplexität

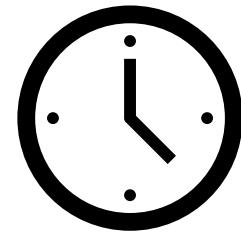


Deklarativ

Was soll getan werden,
nicht wie

Funktionale Programmierung
ist so, als würde man einem Mathematiker sein
Problem beschreiben.

Imperative Programmierung
ist so, als würde man jemandem
Anweisungen geben.



5 Minuten



Nach erfolgreichem Abschließen dieses Kapitels, solltet ihr in der Lage sein die **Lambdas und Yields** in Ruby anzuwenden.

- Definiert Methoden, auch in der Nähe des Aufrufs
- Keyword "lambda" kann statt "->" verwendet werden
- Kann mit {} oder do...end geschrieben werden
- Kann in Variablen gespeichert werden

```
say_something1 = lambda do
  puts "This is a lambda"
end
say_something1.call

say_something2 = -> { puts "This is a lambda" }
say_something2.call
```

- Kann Argumente haben
- Lambda wird mit "call" aufgerufen
- Alternativ:
 - Lambda.()
 - Lambda[]
 - Lambda.==
- Return beendet Lambda und gibt Wert zurück
- Standardwert, wenn kein Argument übergeben wird

```
times_two = ->(x)
{ x * 2 }
times_two.call(10)
```

```
write_name = -> lambda { |name="default"| puts name}
write_name.call
write_name.call("Workshop")
```

- Sehen sehr ähnlich aus
- Lambda benötigt die exakte Anzahl an Argumenten
- Proc überprüft nicht die Parameter
 - Setzt Variable auf **nil**, wenn nicht gesetzt
 - Ignorieren überflüssige Argumente
- Return beendet auch Funktion, die das Proc aufruft
 - Code nach dem Proc Aufruf wird nicht mehr aufgerufen
- Hat immer den aktuellsten Wert der Variablen (Referenz)

```
my_lambda = lambda { |name| puts "lambda says hello " + name.to_s }
my_lambda.call(`to everyone!`)

my_proc = Proc.new { |name| puts "proc says hello " + name.to_s }
my_lambda.call()
my_lambda.call("to everyone!")
my_lambda.call("to you!", "to me!")
```

- ActiveRecord scopes

- Dienen als Filter
- Können in hintereinander aufgerufen werden

```
scope :published, -> { where(published: true) }
```

- Callbacks

- Kapselung von Verhalten, Success und Failure

- Dynamic mapping

- Als Parameter für die Collection.Map Funktion

- A faux hash

- Können dynamisch als Sie ein Hash-ähnliches schreibgeschütztes Objekt erstellt werden
 - Überprüfung des Keys
 - Bearbeitung des Wertes

- Performance

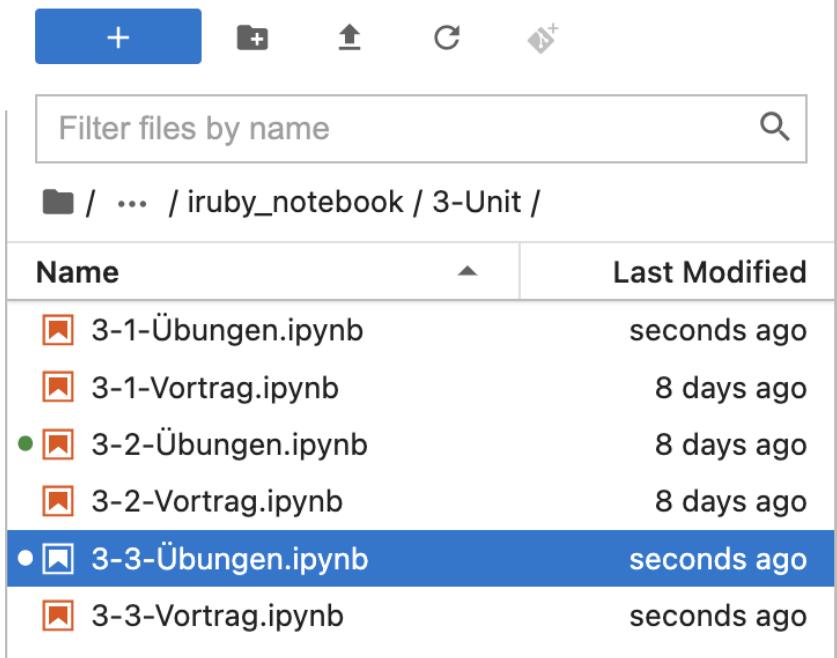
- Lambdas sind circa 30% langsamer als normale Methodenaufrufe

Lambda - A faux hash



```
def build_lambda(restricted_values)
  my_hash = {}
  my_lambda = lambda do |key|
    if restricted_values.include? key
      return "400"
    else
      return "200"
    end
  end
  my_lambda
end

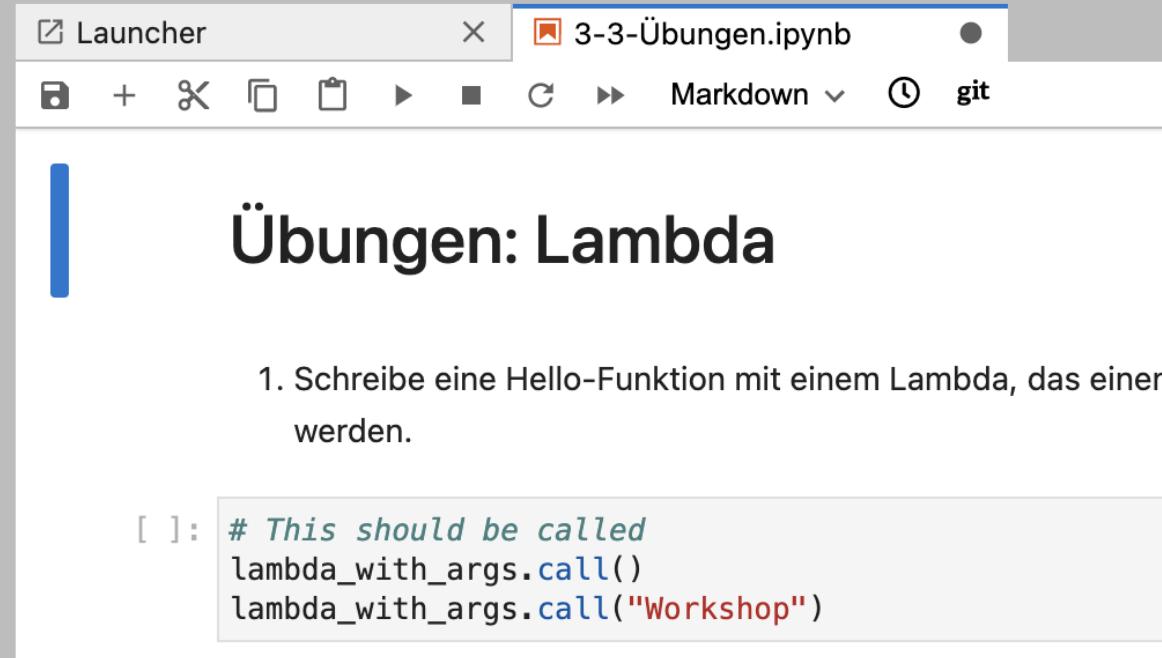
code_check = build_lambda(["401", "404"])
puts code_check["404"]      # => 400
puts code_check["201"]      # => 200
```



A screenshot of a file manager interface showing a list of Jupyter notebooks in a directory. The directory path is / ... / iruby_notebook / 3-Unit /. The list includes:

Name	Last Modified
3-1-Übungen.ipynb	seconds ago
3-1-Vortrag.ipynb	8 days ago
3-2-Übungen.ipynb	8 days ago
3-2-Vortrag.ipynb	8 days ago
3-3-Übungen.ipynb	seconds ago
3-3-Vortrag.ipynb	seconds ago

The notebook 3-3-Übungen.ipynb is currently selected, highlighted with a blue background.



The Jupyter Notebook window title is 3-3-Übungen.ipynb. The content area displays the following text:

Übungen: Lambda

1. Schreibe eine Hello-Funktion mit einem Lambda, das einen werden.

```
[ ]: # This should be called
lambda_with_args.call()
lambda_with_args.call("Workshop")
```

Aufgabe: 1

```
lambda_with_args = lambda do |s = "World"|  
    puts "Hello " + s  
end  
  
lambda_with_args.call()  
lambda_with_args.call("Workshop")
```

Aufgabe: 2

```
add_10 = lambda { |num| num + 10 }  
multiply_2 = lambda { |num| num * 2 }  
  
def using_lambda_with_functions(my_lambda, number)  
    puts my_lambda.call(number)  
end  
  
using_lambda_with_functions(add_10, 10)  
using_lambda_with_functions(multiply_2, 20)
```

Aufgabe: 3

```
def build_lambda()  
    my_lambda = lambda do |key|  
        if key == "200"  
            return "OK"  
        else  
            return "N/A"  
        end  
    end  
    my_lambda  
end  
  
code_check = build_lambda()  
puts code_check["200"]  
puts code_check["404"]
```

- Ruby Keyword
- Ruft Code-Blöcke auf
- Der Code innerhalb des Blocks wird durch `yield` aufgerufen ausgeführt
- `yield` kann mehrfach aufgerufen werden
 - Ruft Block mehrfach auf
- Kann Argumente übergeben
 - Argument wird im Block eingefügt

```
def print_twice
  yield
  yield
end

print_twice { puts "Hello!" }
```

```
def one_two_three
  yield 1
  yield 2
  yield 3
end

one_two_three { |number| puts number * 10 }
```

- Existenz des Code-Block kann überprüft werden

```
def check_block
  return "No block given" unless block_given?
  yield
end
```

- Kann auch direkt beim Aufruf von yield gemacht werden

```
def optional_block
  yield if block_given?
end
```

Yield – Explicit Block



- Ruby Keyword wie yield
- Ruft Code-Blöcke auf
- Speichert Code-Block in Variable "block"
 - Wird als Parameter hinter dem Funktionsnamen Definiert
 - Parameter hat ein vorangestelltes &

```
def explicit_block(&block)
  block.call
end

explicit_block { puts "Explicit block called" }
```

```
class Array
  def my_map
    return self.dup unless block_given? # Überprüfung des Block
    ary = [] # temporäres Array

    # Iteration über alle Elemente der Elternklasse
    self.each do |elem|
      ary << yield(elem) # Push Element in Array
    end

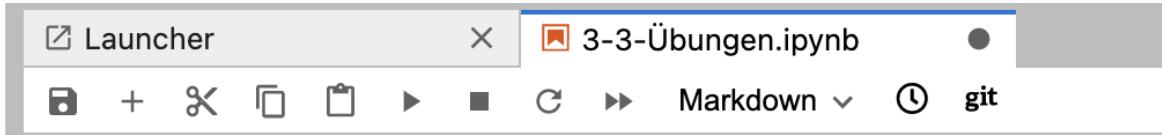
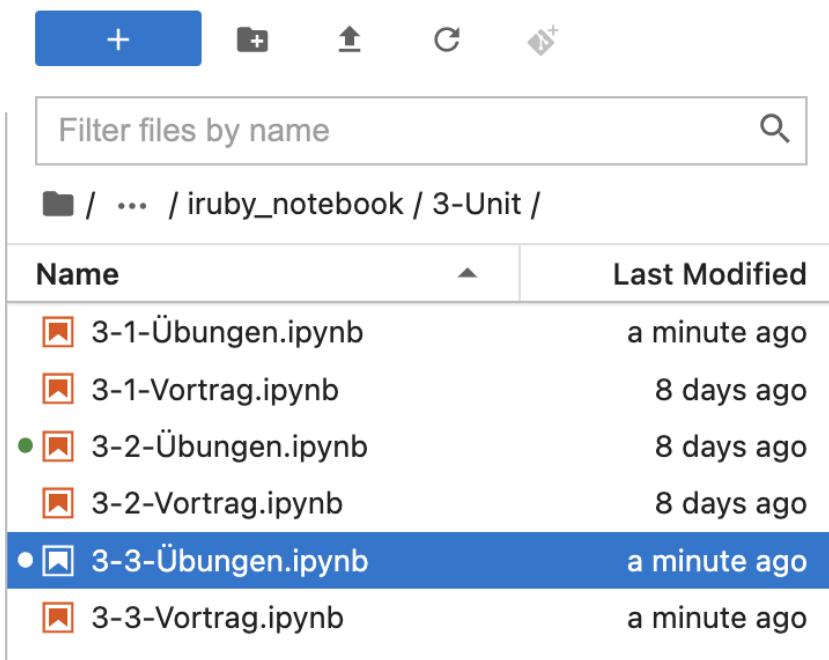
    ary # return Array
  end
end

array.my_map { |n| n + 2 }
```

- Keyword `yield_self`
- Wird benutzt wenn verkettete Aufrufe benötigt werden
- Gibt Ergebnis zurück und nicht das eigentliche Objekt

```
n_squared = -> (n) { n ** 2 }

2.yield_self(&n_squared).yield_self(&n_squared)
# 16
```



Übungen: Yields

1. Schreibe ein Funktion, die ein 2 mal einen yield Aufruf macht

```
[ ]: # This should be called  
call_twice {puts "This is a block"}
```

```
[ ]: # your code here
```

```
def call_twice
  puts "In the call_twice method"
  yield
  puts "Again from the call_twice method"
  yield
end

call_twice {puts "This is a block"}
```

Aufgabe: 1

```
def yield_with_return_value
  hello_world = yield
  puts hello_world
end

yield_with_return_value { "Hello World!" } # => Hello World!
```

Aufgabe: 2

```
def calc
  yield 2*3
  puts "In the method calc"
  yield 100
end

calc { |i| puts "calc #{i}"}
```

Aufgabe: 3

```
def optional_block
  yield if block_given?
end

optional_block
optional_block { puts 'optional block' }
```

Aufgabe: 4

```
def arithmetic(a, b)
  yield(a, b)
end

puts "The sum of the two numbers is #[arithmetic(8, 2) { |a, b| a + b }]"
```

Aufgabe: 5



Nach erfolgreichem Abschließen dieses Kapitels, solltet ihr ein grobes Verständnis von der Programmierung mit **Ruby on Rails** haben.

- Entwicklungsframework
 - aktuelle Version: v7.0.2
 - geschrieben in Ruby
- Vereinfachung der Web-Entwicklung
 - The MVC Architecture
- Produktivitätssteigerung
 - weniger Konfigurationsaufwand
 - weniger Programmieraufwand durch Standards
- Die Rails-Philosophie beinhaltet zwei wesentliche Leitprinzipien:
 - Wiederhole dich nicht
 - Konvention über Konfiguration
- Empfohlene IDE: RubyMine von JetBrains
 - selber Aufbau wie IntelliJ

- Installation von Rails: "**gem install rails**"
 - Ruby muss installiert sein
 - Windows: Ausführen von rubyinstaller.exe von rubyinstaller.org
 - Linux: "**sudo apt install ruby**"
- Update Rails: "**gem install rails**"
- Überprüfung der Installation: "**rails –version**"
- Erstellung einer Anwendung: "**rails new Shop**"
 - Hinzufügen von **gem 'tzinfo-data'** im Gemfile
 - Installieren des bundles mit "**bundle install**"
 - Starten der Anwendung: "**ruby shop/bin/rails server**"
 - Testen der Anwendung: <http://localhost:3000>

- Hinzufügen einer Route in "config/routes.rb":

```
Rails.application.routes.draw do
  root "articles#index"
  get "/articles", to: "articles#index"
end
```

- Erstellen eines Controllers:
 - View in "app/views/articles/index.html.erb"

ruby bin/rails generate controller Articles index --skip-routes

- Erstellung von Objekten:
 - name und description sind Attribute von Artikel
 - Erstellt auch DB Migrations und Tests
 - ID (auto-increment) wird automatisch miterstellt

ruby bin/rails generate model Article name:string description:text

- Migration der Datenbank:

ruby bin/rails db:migrate

- Generierung von Scaffolds:

- Angabe von Parametern, wie bei Klassen
- Generiert
 - Model und Tabelle
 - Benutzeroberflächen für alle CRUD Befehle
 - Tests
 - Controller
 - Template für Helper-Klasse
- Konfiguriert Pfade

**ruby bin/rails generate scaffold User
name:string**

```
invoke  active_record
create  db/migrate/20130924151154_create_users.rb
create  app/models/user.rb
invoke  test_unit
create  test/models/user_test.rb
create  test/fixtures/users.yml
invoke  resource_route
       route  resources :users
invoke  scaffold_controller
create  app/controllers/users_controller.rb
invoke  erb
create  app/views/users
create  app/views/users/index.html.erb
create  app/views/users/edit.html.erb
create  app/views/users/show.html.erb
create  app/views/users/new.html.erb
create  app/views/users/_form.html.erb
invoke  test_unit
create  test/controllers/users_controller_test.rb
invoke  helper
create  app/helpers/users_helper.rb
invoke  jbuilder
create  app/views/users/index.json.jbuilder
create  app/views/users/show.json.jbuilder
invoke  test_unit
create  test/application_system_test_case.rb
create  test/system/users_test.rb
```

- Benutzeroberflächen als html.erb
 - HTML Embedded RuBy
- Ruby code in Tags
 - `<% ... %>` führt Code aus
 - `<%= ... %>` führt Code aus und zeigt es an
 - `link_to` erstellt einen Hyperlink
 - `new_user_path` wird automatisch auf die new.html.erb im user folder gemappt

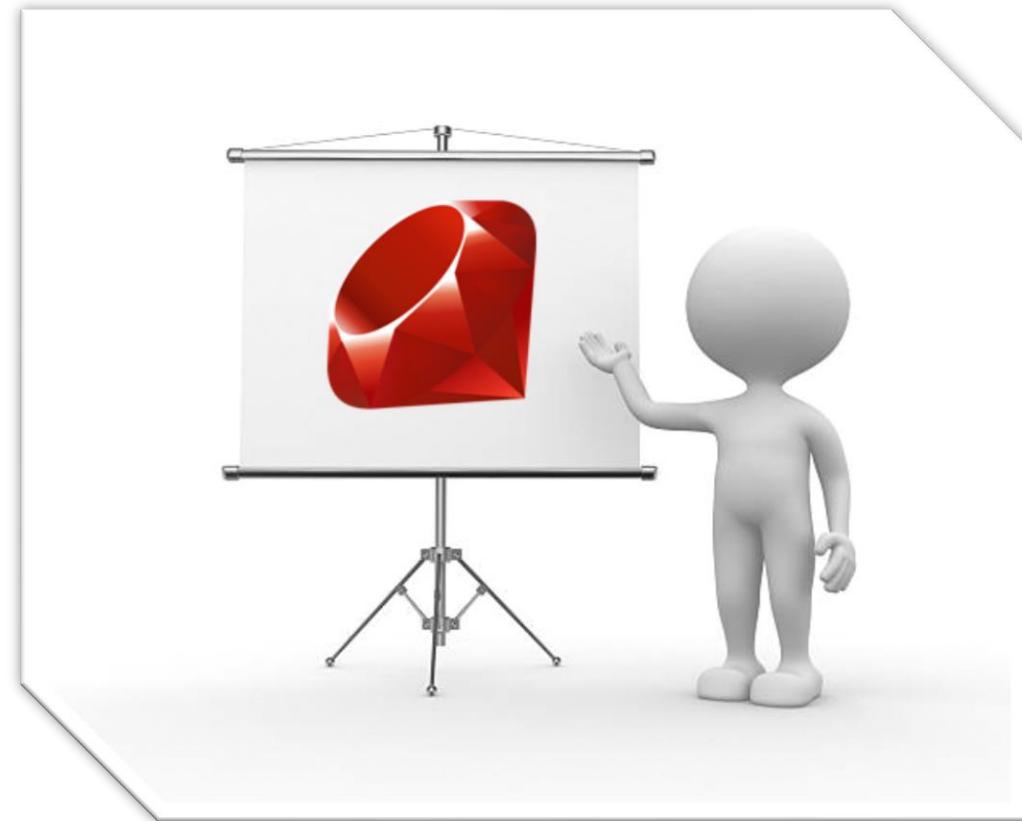
```
<p style="color: green"><%= notice %></p>

<h1>Users</h1>

<div id="users">
  <% @users.each do |user| %>
    <%= render user %>
    <p>
      <%= link_to "Show this user", user %>
    </p>
  <% end %>
</div>

<%= link_to "New user", new_user_path %>
```

L I V E



QUIZ

Erreichbar unter:

<https://forms.office.com/r/RU1U1d3vXq>

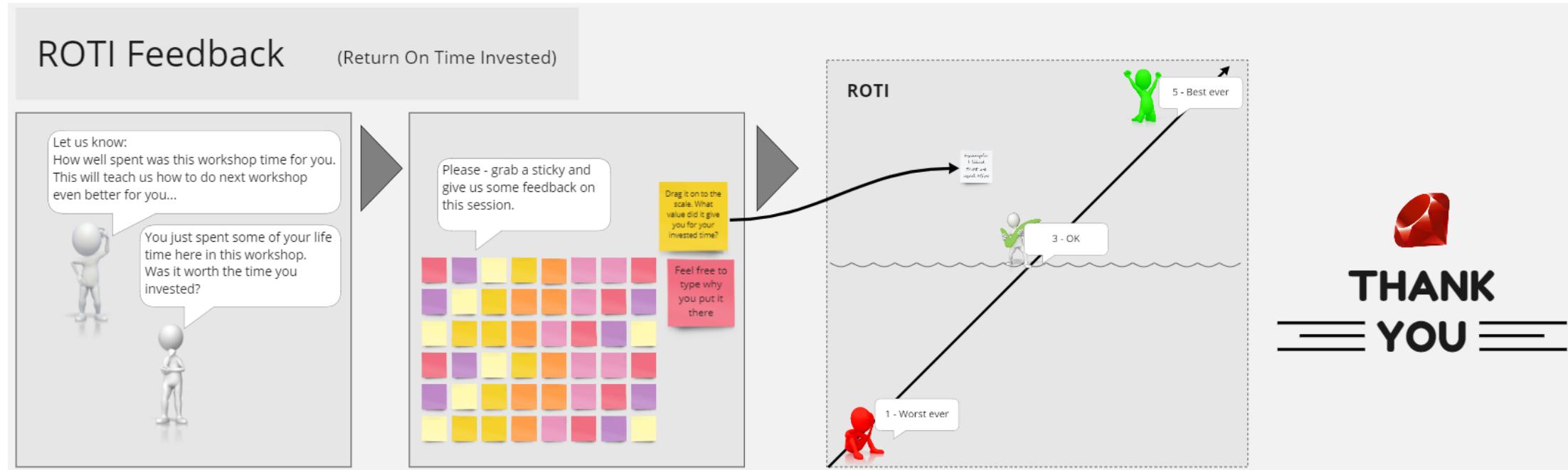


Jetzt ist euer Feedback gefragt!

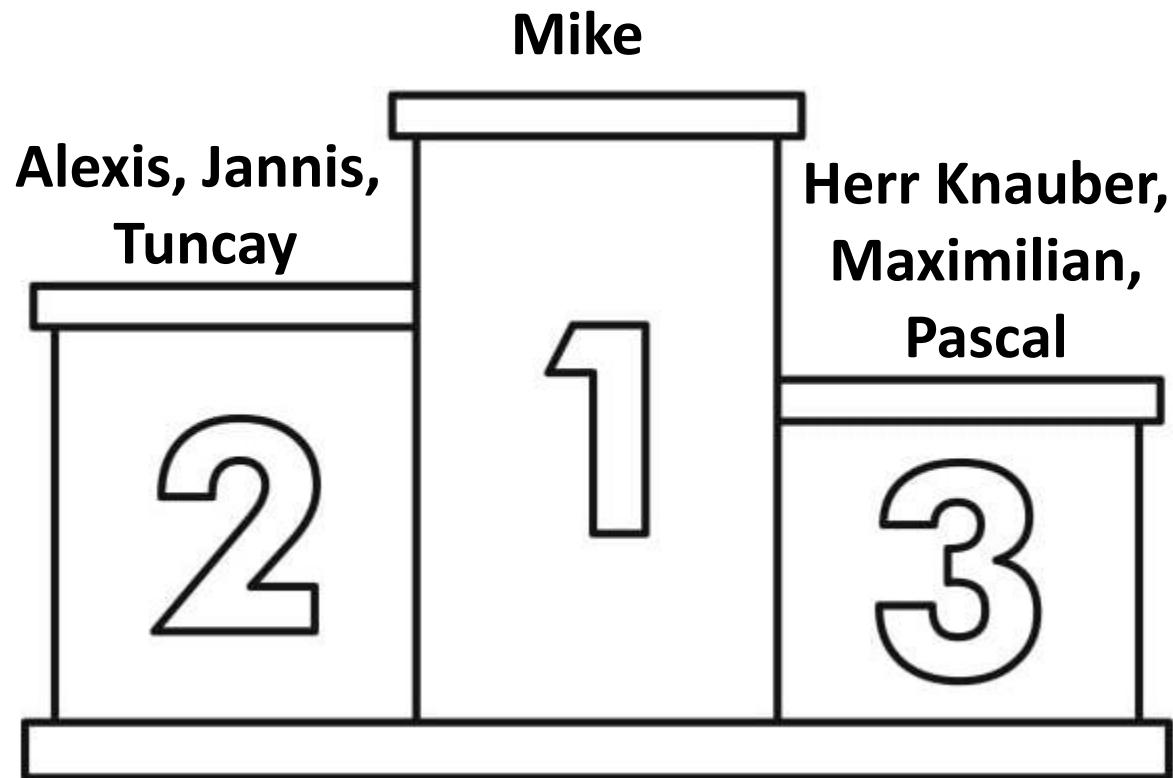
Link zum Miro Board: <https://bit.ly/3HZG1LN>

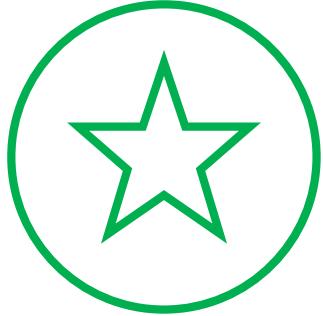
Zeit zum Ausfüllen: **5 Min.**

Diskussion: **5 Min.**



And the Winner is...





4.1

The screenshot shows a GitHub repository page. At the top, it displays the repository name 'otiofrui / rubyworkshop' and a 'Public' button. Below this are navigation links for 'Code', 'Issues', 'Pull requests', and a profile icon. Underneath, there's a summary showing 'main' branch, '5 branches', '0 tags', and two recent commits: 'Sebastian-Schaefer Add files via upload' and 'iruby_notebook'. A hand cursor icon is overlaid on the commit list, pointing towards the second commit.

<https://github.com/otiofrui/rubyworkshop>

