# Inhaltsverzeichnis

# 1 Einleitung und How to use

Alle kleinen Zusammenfassungen und Code-snippets sind 1:1 unter der originalen Überschrift von Bogner zu finden.

**Der Inhalt der Dateien ist von mir teilweise ergänzt oder Teile komplett selbst geschrieben.**

**Dieses Dokument erhebt keinen Anspruch auf Korrektheit oder Vollständigkeit!**

Du kannst entweder über das Inhaltsverzeichnis suchen (klicki-bunti) oder das Stichwortverzeichnis nutzen, um z. B. die Seite zu finden auf der eine if-Anweisung erklärt ist.
Alle Links sind blau.

# 2 Ruby

## 2.1 Aufgaben (Code)

### 2.1.1 01_ start

a. Öffnen Sie eine irb - Session auf der Console
b. addieren Sie zwei Zahlen
c. weisen Sie der Variable tage die Anzahl der Tage pro Jahr zu
d. weisen Sie der Variable stunden die Anzahl der Stunden pro Tag zu
e. ermitteln Sie die Anzahl der Stunden in einem Jahr über die Multiplikation der Variablen
f. speichern Sie in der Variable str den String "Hello Ruby!"
g. Geben Sie den String in Großbuchstaben aus
h. schliessen Sie die irb - Session
i. öffnen Sie eine Textdatei, speichern Sie diese als hello.rb. notieren Sie eine Willkomensnachricht, die auf der Konsole ausgegeben werden soll
j. rufen Sie die Datei hello.rb auf der Konsole auf
k. speichern Sie die Willkommensnachricht in der Variable hello
l. geben sie die Willkommensnachricht drei mal hintereinander aus
m. Geben Sie die aktuelle Zeit aus

```
var = 2+2

puts var

tage = 365

stunden = 24

puts tage*stunden

"Hello Ruby!"

puts str.upcase

ruby hello.rb

hello = "Hallo Welt"

3. times do puts hello end

puts Time.now
```

### 2.1.2 02_ strings

a. Legen Sie eine neue Datei studio_ game.rb an
b. speichern Sie in der Variable name1 den Wert 'larry'
c. speichern Sie in der Variable health den Wert 60
d. geben Sie auf der Konsole: 'larry's health is 60' , unter Verwendung der Variablen aus (2 Versionen)
e. verdreifachen Sie den Wert von larrys health
f. reduzieren Sie die health durch eine Division durch 9. Geben Sie eine Variante mit float
und eine mit integer aus
g. erzeugen Sie folgende Ausgabe:

    Players:
    larry
    curly
    moe

h. speichern sie die Namen der Spieler in Variablen und geben Sie diese erneut wie abgebildet aus

```
name1 = "larry"
health = 60
name2 = "curly"
name3 = "moe"
puts "#{name1}'s health is #{health}
#{name1}'s health is #{health * 3}
#{name1}'s health is #{(health / 9.0).round(2)}
#{name1}'s health is #{health / 9}
Players:
\t#{name1}
\t#{name2}
\t#{name3}
"
allenamen = <<array1zu1
Players:
#{name1}
#{name2}
#{name3}
array1zu1
puts allenamen
```

### 2.1.3 03_ strings2

Erzeugen Sie folgende Ausgabe:
Larry has a health of 60.
CURLY has a health of 125.
**********Moe has a health of 100.**********
**********Moe has a health of 100.**********
Shemp..... 90 health
Shemp..... 90 health
Players:
    Larry
    Curly
    Moe
The game started on Thursday 08/02/2012 at 11:42AM

```
name1 = "Larry"
name2 = "curly"
name3 = "Moe"
name4 = "Shemp"
health = 5
allename = <<array1zu1
Players:
   #{name1}
   #{name2}
   #{name3}
array1zu1
var1 = "#{name3} has a health of #{health*20}"
var2 = "#{name4} has a health of #{health*18}"
var3 = "#{health*18} health"
puts"
#{name1} has a health of #{health*12}
#{name2.upcase} has a health of #{health*25}
#{var1.center(50, "*")}
#{var1.center(50, "*")}
#{name4}#{var3.rjust(30, ".")}
#{name4}#{var3.rjust(30, ".")}
#{allename}
This game started on #{Time.now.strftime("%A %D at %H:%M %p")}
```

### 2.1.4 04_ methoden

a. definieren Sie eine Methode say_ hello mit dem Argument name. Das Ergebnis des Aufrufs ist dann die Ausgabe: 'Ich bin Ludwig'
b. rufen Sie die Methode mit verschiedenen namen auf
c. puts soll nun aus der Methode entfernt werden
d. ändern sie die Methode, so dass sie einen Standardparameter health entgegennimmt. Die Ausgabe lautet dann: 'Ich bin Ludwig mit einem Wert von 100'
e. definieren Sie eine Methode time, die in der Methode say_ hello aufgerufen wird und die Ausgabe folgendermaßen ergänzt: 'Ich bin Ludwig mit einem Wert von 100 um 18:25'

```ruby
def time
   Time.now.strftime("%H:%M")
end

def say_hello(name, health = 100)
   if not name
      name = 'Ludwig'
   end
   "Ich bin #{name} mit einem Wert von #{health} um #{time}"
end
puts say_hello(ARGV[0])
```

#### 2.1.5 05_ classes

a. erstellen Sie eine Klasse mit der Bezeichnung Player
b. erstellen Sie ein Objekt player1 der Klasse
c. erstellen Sie eine initialize - Methode, die die Parameter name und health hat und aus diesen Werten die Instanzvariablen @name und @health initialisiert
d. updaten Sie das Objekt player1 und lassen sich das Objekt anzeigen
e. setzen Sie für health einen Standardwert 100
f. legen Sie eine neues Objekt player2 an, das nur den Namen als Parameter hat
g. passen Sie die say_ hello - Methode aus der vorherigen Übung zur Ausgabe an,
sodass puts player1.say_ hello die Ausgabe ergibt
h. fügen Sie eine Instanzmethode blam und w00t hinzu, die den Wert für health um 10 hochsetzt, bzw. reduziert und ausgibt: Tom got blamed, bzw. w00ted.
i. die say_ hello - Methode soll aufgerufen werden, wenn nur das Objekt ausgegeben werden soll, also puts players erfolgt.

```ruby
class Player
   def initialize(name, health = 100)
      @name = name
      @health = health
   end
   def to_s
      "Ich bin #{@name} mit einem Wert von #{@health}"
   end
   #def name <- ist das gleiche wie oben das attr_... GETTER
   #  @name
   #end
   attr_reader :name
   #def name = (name) <- neu beschrieben der Variable... SETTER
   #  @name = name
   #end
   attr_writer :name
   # attr_accessor :name <- Kurzform für beides
   def blame(value = 10)
      i = 1
      healthp = @health
      while i < 11 do
         healthp = healthp + value
         puts "#{@name} besitzt #{healthp} Healthpoints"
         i = i +1
         sleep 1
      end
   end
   def w00t(value = 10)
      i = 1
      healthp = @health
      while i < 11 do
         healthp = healthp - value
         puts "#{@name} besitzt #{healthp} Healthpoints"
         i = i +1
         sleep 2
      end
   end
end
player1 = Player.new("Tom")
player2 = Player.new("Horst")
player3 = Player.new("Hans")

#puts player1.name

Players = [player1, player2, player3]

puts "Es sind #{Players.size} Spieler angemeldet:"

i = 1
Players.each do |spieler|
   puts "Spieler #{i} heißt #{spieler.name}"
   i += 1
end
```

### 2.1.6 06_ attributes

a. machen Sie aus der Instanzvariable @health ein lesbares Attribut
b. geben Sie die Eigenschaft health des Objektes player1 aus
c. die Eigenschaft name soll sowohl lesbar, als auch schreibbar sein
d. erzeugen Sie ein virtuelles Attribut score, das den Wert von health addiert mit der Buchstabenlänge des name -Attributes
e. ändern Sie die to_ s - Methode, zur Ausgabe des Score

```ruby
class Player
   attr_reader :health
   attr_writer :name
   attr_reader :name

   def initialize(name, health=100)
      @name=name
      @health=health

   end

   def say_hello
      puts "Ihr Name: #{@name} und health #{@health}"

   end

   def to_s
        length = @name.length
        @health += length
   end


end


player1 = Player.new("Kim",24)
player2 = Player.new("Christian")

puts player1.health
puts player1.name
player1.name = "Hans"
puts player1.name
puts player1.to_s
```

### 2.1.7 06_ uebung_ bank

Einrichten einer Kundenklasse.
Eine Klasse customer soll eingerichtet werden.
Beim Anlegen eines Customer-Objektes muss als notwendiger Parameter eine Kontonummer (Instanzvariable bank_ account) eingegeben werden.
Optional kann auch der Nachname des Kunden (lastname) eingegeben werden.
Erstellen Sie SET und GET - Methoden für die Eigenschaften (Instanzvariablen) firstname, street, city. z. B. heisst die Set-Methode fuer street: set_ street(pstreet)
Eine Methode print_ address soll beim Aufruf "puts obj.print_ address' eine Ausgabe wie folgt erzeugen:

Konto:.....9999
Name:.....Theo Lodgz
Strasse:.....Hermelinweg 7
Ort:.....22159 Hamburg

Legen Sie das Objekt obj an, rufen Sie die entsprechenden Get-Methoden für die Dateneingabe auf, erzeugen Sie die Kontrollausgabe.

```ruby
class Customer

   def initialize (bacc,lname ="")
      @bank_account = bacc
      @lastname = lname
   end

   def set_first(fname = " ")
      @firstname = fname
   end
   def set_last(lname = "")
      @lastname = lname
   end

   def set_street(str = " ")
      @street = str
   end

   def set_city(city = " ")
      @city = city
   end

   def print_address
      "Kunde:
   Konto:#{@bank_account.rjust(20, '.')}
   Name:#{@lastname.rjust(20, '.')}
   Vorname:#{@firstname.rjust(20, '.')}
   Strasse:#{@street.rjust(20, '.')}
   Ort:#{@city.rjust(20, '.')}"
   end

end

customer1 = Customer.new("18")
customer2 = Customer.new("5678","Horst")

customer2.set_first("Marvin")

customer2.set_street("Gasse 1")

customer2.set_city("23456 Dorf")

puts customer2.print_address
```

### 2.1.8 06_ uebung_ calculator

Erstellen Sie eine Klasse Calculator mit einem Konstruktor, der zwei Zahlen entgegennimmt und in Instanzvariablen speichert.
Eine Methode add, und eine Methode subtract führt eine Addition und eine Substraktion durch und gibt das Ergebnis zurueck.
Legen Sie ein Objekt an und wenden Sie die Methoden add und subtract an.

```ruby
class Calculator
   attr_accessor :zahl1, :zahl2
   def initialize(zahlA, zahlB)
      @zahl1 = zahlA
      @zahl2 = zahlB
   end
   def substract
      @ergS = @zahl1 - @zahl2
      puts "Das Ergebnis ist #{@ergS}"
   end
   def add
      @ergA = @zahl1 + @zahl2
      puts "Das Ergebnis ist #{@ergA}"
   end
   def multipl
      @ergM = @zahl1 * @zahl2
      puts "Das Ergebnis ist #{@ergM}"
   end
   def divid
      if @zahl1 == 0 || @zahl2 == 0
         puts "Error"
      else
         @ergD = @zahl1 / @zahl2
         puts "Das Ergebnis ist #{@ergD}"
      end
   end
end
obj = Calculator.new(5, 2)
obj.zahl1 = 8
obj.divid
```

### 2.1.9 07_ array

a. legen Sie zur Übung auf der irb ein Array jahreszeiten mit den Elementen: Sommer, Herbst, Winter an; sowohl in der Langform, als auch der Kurzform
b. lassen Sie ausgeben wieviele Elemente das Array enthält, welche Jahreszeit die zweite im Array ist
c. fügen Sie das Element fruehjahr hinzu
d. entfernen Sie das Element wieder aus dem Array
e. ermitteln Sie mit Hilfe von ri, wie join Ihnen bei einer folgendermaßen formatierten Ausgabe behilflich sein kann: Sommer und Herbst und Winter
f. notieren Sie den Befehl
g. ermitteln Sie den Befehl mit dem Sie die Positionen der Elemente des Arrays nach einem Zufallsprinzip anordnen können
h. packen Sie Ihre drei Player aus dem Projekt in ein Array
i. Geben Sie Player mit einer Iteration über das Array aus. Beginnen Sie mit einer Zeile, in der sie die Anzahl der Mitspieler notieren

```
#a
jahreszeiten = ["Sommer", "Herbst", "Winter"]
jahreszeiten = %w[Sommer Herbst Winter]
#b
jahreszeiten.size
#c
jahreszeiten.push("Frühjahr")
#d
jahreszeiten.pop
#e/f
jahreszeiten.join("und")
#g
jahreszeiten.shuffle
#h
players = ["Player1", "Player2", "Player3"]
#i
#siehe 05.loesung
```

### 2.1.10 08_ objects_ interacting

a. erstellen Sie eine Klasse Game mit einem Konstruktor, der eine Instanzvariable title anlegt. Zugleich soll ein leeres Array mit dem Namen players erstellt werden.
b. die Klasse Game bekommt eine Methode add_ player der ein Spieler hinzugefügt wird
c. eine Methode play erzeugt folgenden Ausdruck:

There are 3 players in Knuckleheads:
I'm Moe with a health of 100 and a score of 103.
I'm Larry with a health of 60 and a score of 65.
I'm Curly with a health of 125 and a score of 130.
Moe got blammed!
Moe got w00ted!
Moe got w00ted!
I'm Moe with a health of 120 and a score of 123.
Larry got blammed!
Larry got w00ted!
Larry got w00ted!
I'm Larry with a health of 80 and a score of 85.
Curly got blammed!
Curly got w00ted!
Curly got w00ted!
I'm Curly with a health of 145 and a score of 150.

```
class Game
     def initialize(title)
        @title = title
        @Player = []
     end
     def add_player(player)
        @Player << player

     end
     def play
        puts "There are #{@Player.size} Players in #{@title}\n"

           @Player.each do |player|
              puts player.say_hello
           end
           @Player.each do |player|
              puts player.blam
              puts player.woot
              puts player.woot
              puts player.say_hello
           end
     end
end
class Player
   def initialize(name, health=100)
      @name = name
      @health = health
   end
   def say_hello
      "I'm #{@name} with a health of #{@health} and a score of #{score}"
   end
   def blam
         @health -= 10
           if @health <= 0
```

```
            @health = 0
            "#{@name} got blamed"
          else
            "#{@name} got blamed"
      end
   end
   def woot
      @health += 10
         "#{@name} got wooted"
   end
   def score
      @health + @name.length
   end
end

player1 = Player.new("Hans",24)
player2 = Player.new("Max",12)
player3 = Player.new("Moritz",42)

players = [player1, player2, player3]

Game1 = Game.new("Knuckleheads")
players.each do |player|
   Game1.add_player(player)
end
#players.each {|player| Game1.add_player(player)}
Game1.play
```

#### 2.1.11 08_ uebung_ bank

1. Klasse Kunde
1a. Legen Sie eine Eigenschaft name und adresse in der Klasse Kunde an. Für die Eigenschaft name gibt es einen Lese- und Schreibzugriff, für die Eigenschaft adresse einen Schreibzugriff.
b. Die Eigenschaft name enthält den Vor- und Zunamen des Kunden. Diese Eigenschaft wird beim Anlegen des Objektes übergeben. Schreiben Sie den entsprechenden Konstruktor.
c. Erzeugen Sie ein Objekt mit dem Namen 'kunde1' und dem Initialwert: 'Theo Sommer'
d. Ändern Sie die Eigenschaft 'name' des Objektes in: 'Theo Sonnenschein'
e. Schreiben Sie den Befehl, der den Namen auf der Konsole ausgibt.
f. Fügen Sie der Klasse Kunde die Eigenschaft gehalt hinzu, die sowohl gelesen als auch geschrieben werden kann. Schreiben Sie den Befehl auf, der das Gehalt des Objektes als Konsolenbefehl auf 2000 setzt.
g. Die Eigenschaft adresse bekommt folgenden String übergeben: Hermelinweg 11, 22159 Hamburg
h. Schreiben Sie eine Getter-Methode' für die Klasse Kunde, für die Eigenschaft adresse die folgende Ausgabe produziert: 'Theo Sonnenschein, Hermelinweg 11, 22159 Hamburg.' und geben sie das auf der Konsole aus.
i. Zusatz optional: Sichern Sie, dass im String Theo Sonnenschein sowohl Theo, als auch Sonnenschein jeweils mit einem Grossbuchstaben beginnen
2. Klasse Kredit
a. Legen Sie eine Klasse Kredit an, mit der Eigenschaft kunde (ohne Setter und Getter) und einem Konstruktor, der die Eigenschaft kunde füllt. Erzeugen Sie ein Objekt kredit1, dem Sie das bereits existierende Objekt kunde1 übergeben.
b. Legen Sie eine Getter-Methode für die Eigenschaft kunde an, die folgende Ausgabe erzeugt: 'Theo Sonnenschein, Hermelinweg 11, 22159 Hamburg. Jahresgehalt: 24000 Euro.'
c. Fügen Sie eine Eigenschaft kredit hinzu, auf die lesend und schreibend zugegriffen werden kann. Speichern Sie für das bestehende Objekt in der Eigenschaft kredit den Wert 5000.

```ruby
class Kunde

   attr_accessor :name, :gehalt
   attr_writer :adresse


      def initialize(name)
         @name = name
      end


      def adresse
         "#{@name}, #{@adresse}."
      end
end
class Kredit

   attr_accessor :kredit

   def initialize(kunde)
      @kunde = kunde
   end

   def kunde
      puts "#{@kunde.adresse} Jahresgehalt: #{@kunde.gehalt * 12}"
   end
end

kunde1 = Kunde.new("Theo Sommer")
kunde1.name = "Theo Sonnenschein"
#puts kunde1.name
kunde1.gehalt = 2000
kunde1.adresse = "Hermelinweg 11, 22159 Hamburg"

#puts kunde1.adresse

kredit1 = Kredit.new(kunde1)
#kredit1.kunde
kredit1.kredit = "5000"


#methode um alle teile eines Strings großzuschrieben
str = "a b c"
str_array = str.split
ret = ""
str_array.each do |teilstring|
   ret = ret + " " + teilstring.capitalize
end
# ret.strip
```

#### 2.1.12 09_ bloecke

1. Legen Sie ein Array an mit den Zahlen 15 bis 20 an.
a. geben Sie diese Zahlen in einem Block aus
b. geben Sie die Zahlen mit dem Index etwa so aus: Index: 0 - Wert: 15
c. erzeugen Sie in einem Block einen Hash, wobei Sie die Arraywerte als Index nutzen und als Wert eine Zufallszahl zwischen 100 und 200 speichern.
d. Geben Sie diesen Hash anschließen etwa so aus: Index: 15 - Wert: 121
e. Ergänzen Sie die Ausgabe. Wenn der Wert größer als 150 ist, dann sieht die Ausgabe aus: Index: 15 - Wert: 151. WOW
f. Durchlaufen Sie das Array und speichern Sie die Quadratzahlen in einem Array
g. ! Durchlaufen Sie das Array und speichern Sie die Quadratzahlen in einem Hash mit der Zahl als Schlüssel
Zusatz Zahlen im Array und im Hash:
a. Legen Sie ein Array an mit 10 Primzahlen. Die Ausgabe des Arrays erzeugt:
1 ist einstellig 3 ist einstellig ... 11 ist zweistellig ... b. Legen Sie zur Übung einen Hash personen an, mit den keys vorname, nachname, stadt und den Werten Theo, Sommer, Lodz. Machen Sie eine Ausgabe, die 'vorname: Theo + neue Zeile' ausgibt
c. Fortsetzung Aufgabe a:
Speichern Sie beim Durchlaufen des Arrays die Werte in einem Hash mit der Bezeichnung prims, sodass die jeweilige Primzahl den Schlüssel bildet und der Wert den Text 'ist einstellig', 'ist zweistellig'
d. Sortieren Sie den Hash, sodass die Ausgabe folgendermaßen möglich wird:
Einstellig: 1,3,5 ..
Zweistellig: 11, 13, ..
e. Fortsetzung Aufgabe b:
Speichern Sie den Hash in dem Array adressen und legen Sie zwei weitere Hashes personen mit den gleichen Keys und anderen Werten an. Durchlaufen Sie das Array, sodass folgende Ausgabe erfolgt.
1.Person:
Name: Theo
Vorname: Sommer
Stadt: Lodz
2. Person:
.....
2. Worte
a. Bilde ein Array mit den Worten: Pflaume, Bauschaum, Auster
b. Alle Element, die den String aus enthalten sollen ausgegeben werden.
c. erzeugen eine Ausgabe nach der Länge der Worte von klein nach groß
d. Ein Hash wird ausgegeben, das Wort als Index, die Anzahl der Buchstaben als Wert
3. Lottogenerator:
a. Schreiben Sie einen kleinen Lottozahlengenerator. Es sollen 7 Zahlen aus 49 Möglichkeiten in einem Array ausgegeben werden.
b. Peter, Paul und Mary bilden einen Hash, etwa derart: lotto = "Peter" => [ 2,22,33,11,23,32,26 ], ...
4. Vergleich:
str_ 1 = "Programmieren, Federball, Whisky"
str_ 2 = "Fussball, Bier, Programmieren"
Die Ausgabe lautet:
ergebnis = :diff => ["Federball", "Whisky", "Fussball", "Bier"], :match => ["Programmieren"]

```ruby
###1a,b
arr = (15..20)
arr.each do |v|
     puts v
end

###1c,d,e
hash = Hash.new

arr.each do |key|
   hash[key] = rand(100) + 100
end

hash.each do |key,val|
        if val > 150
           puts "Index: #{key} - Wert: #{val}. WOW"
        else
           puts "Index: #{key} - Wert: #{val}"
        end
end
###1f
quadrat = Array.new
arr.each do |val|
   quadrat << val * val
end
puts quadrat
###1g
hashquadrat = Hash.new
arr.each do |val|
   hashquadrat[val] = val * val
end
puts hashquadrat

###2a,c
prim = Array.new
   prim = %w[2 3 5 7 11 13 17 19 23 29]

prims = Hash.new
prim.each do |pri|
   if pri.to_i < 10
      puts "#{pri} ist einstellig"
         prims[pri] = "ist einstellig"
   elsif pri.to_i >= 10
      puts "#{pri} ist zweistellig"
         prims[pri] = "ist zweistellig"
   end
end
puts prims

###2b
personen = Hash.new
personen = {vorname: "Theo", nachname: "Sommer", stadt: "Lodz"}
   personen.each do |key,val|
      puts "#{key}: #{val} \n"
   end

###2d
einstellig = ""
```

```ruby
zweistellig = ""
prims.each do |key,val|
   if val == "ist einstellig"
      einstellig += "#{key} "
   end
   if val == "ist zweistellig"
      zweistellig += "#{key} "
   end
end
puts "Einstellig: #{einstellig}"
puts "zweistellig: #{zweistellig}"

###2e
adressen = Array.new

personen = Hash.new
personen = {vorname: "Theo", nachname: "Sommer", stadt: "Lodz"}
adressen << personen

personen = Hash.new
personen = {vorname: "Hans", nachname: "Mann", stadt: "Hamburg"}
adressen << personen

personen = Hash.new
personen = {vorname: "Angie", nachname: "Merkel", stadt: "Berlin"}
adressen << personen

count = 0
adressen.each do |var|
   count += 1
   puts "#{count.to_s}. Person\n"

      var.each do |val,key|
         puts "#{val}: #{key}\n"
      end
end

###Worte
###2a,b
worte = Array.new
worte = %w[Pflaume Bauschaum Auster]

worte.each do |var|
   if var.include?("aus") || var.include?("Aus")
      puts var
   end
end

###2c
puts worte.sort_by {|x| x.length}

###2d
liste = Hash.new

   worte.each do |val|
      var = val.length

      liste[val] = var
```

```
   end
puts liste

###3a

lotto = Array.new
lotto1 = Array.new
lotto2 = Array.new
person = Hash.new

until lotto.size == 7 do
   lotto << rand(1..49)
   lotto1 << rand(1..49)
   lotto2 << rand(1..49)
end

person = {Peter: lotto, Paul: lotto1, Mary: lotto2}

puts "lotto = #{person}"

###4.
str_1 = "Programmieren, Federball, Whisky"
str_2 = "Fussball, Bier, Programmieren"

arr1 = str_1.split(', ')
arr2 = str_2.split(', ')


match = Array.new
diff = Array.new

ergebnis = Hash.new


i = 0

while i < arr1.size do

   arr1.each do |var|
      if var == arr2[i]
         match << arr2[i]

      end
   end
   i += 1
end

u = 0

while u < match.size do

   arr1.each do |val|
      if match[u] != val
         diff << val
      end
   end
   arr2.each do |val|
      if match [u] != val
```

```
        diff << val
      end
    end

    u+=1

end

ergebnis[:diff] = diff
ergebnis[:match] = match

puts "ergebnis = #{ergebnis}"
```

### 2.1.13 vorbereitung__ kla.rb

```ruby
#aus vorbereitung_kla.txt aufgaben 1-7
#Aufgabe 7

class Driver
   attr_accessor :name
   def initialize(name)
      @name = name
      @cars3 = []
   end
   def add_cars(var)
      @cars3 << var
   end
   def show_cars
      @cars3.each do |car|
         puts car.typ
      end
   end
end



#Aufgabe 1
class Car


   attr_accessor :ps, :typ, :verbrauch

   #Aufgabe 3
      def initialize(ps, typ="")
         @ps = ps
         @typ = typ
      end

   #erweiterte Aufgabe 1 Lösung
   #  attr_accessor :ps :verbrauch
   #   attr_reader :typ
   #   def typ=(typ)
   #     @typ = typ.to_s.capitalize
   #   end

   #2te erweiterte Aufgabe 1 Lösung
   #  attr_accessor :ps :verbrauch
   #   attr_reader :typ
   #   def typ=(typ)
   #     if typ.is_a?(String) && typ.size >= 3
   #        @typ = typ.capitalize
   #     else
   #        puts "war kein string"
   #     end
   #   end

   #3te erweiterte Aufgabe 1 Lösung
   #  attr_accessor :verbrauch
   #   attr_reader :typ, :ps
   #  def ps=(ps)
   #     @ps = ps if ps > 0
```

```ruby
  #  end
  #   def typ=(typ)
  #     if typ.is_a?(String) && typ.size >= 3
  #        @typ = typ.capitalize
  #     else
  #        puts "war kein string"
  #     end
  #   end
  #Ende aufgabe 1
  #Aufgabe 2
  def beschleunigen(zeit)
     speed = @ps*zeit/100

     puts "Das Fahrzeug hat #{convert_to_kmh(speed)} km/h erreicht"
  end

  private
     def convert_to_kmh(var)
        var * 3.6
     end
  #Ende aufgabe 2
end

#für vererbung.txt als übung
#class Lkw < Car

#end
#obj2 = Lkw.new
#obj2.typ = 1
#obj2.ps = 50
#obj2.beschleunigen(2)

#Ende für vererbung.txt als übung

#Ausgaben für aufgabe 2
#obj = Car.new
#obj.typ = 1
#obj.ps = 50
#obj.beschleunigen(2)

#Für Aufgabe 4
car3 = Car.new(200, "Klaumich")
car4 = Car.new(80, "Hauni")
car5 = Car.new(100, "Audi")

#Für Aufgabe 5
cars = [car3, car4, car5]

#cars.each do |car|
#  puts car.typ.empty? ? "Kein Wert" : car.typ
#
#  if car.typ.empty?
#     puts "Kein Wert"
#  else
#     puts car.typ
#  end
#end
```

```ruby
#Aufgabe 6
vals = { 'BMW' => 200, 'trabbi' => 150, 'Honda'=> 20}
cars2 = []
vals.each do |typ, ps|
   cars2 << Car.new(ps, typ)

end
#puts cars2.inspect


#Aufgabe 7 ausgabe


theo = Driver.new("theo")
theo.add_cars(cars[0])
theo.add_cars(cars2[0])
theo.show_cars
```

## 2.2 Dokumentation

### 2.2.1 abfragen

if - Anweisungen
Die Überprüfung einer oder mehrerer Bedingungen findet in if-Blöcken statt.
Müssen mehrere Bedingungen überprüft werden, dann findet eine Verknüpfung mit dem Und- Operator statt:
if typ.is_ a?(String) && typ.size > 2
Es müssen beide Bedingungen true sein, damit die Anweisungen in dem Block ausgeführt werden
Muss von mehreren Bedingungen eine zutreffen ist die Verwendungen des Oder-Operators sinnvoll:
if bedingung1 || bedingung2
Wenn einer der Bedingungen zutrifft werden die Anweisungen im Block ausgeführt

if - elsif - else -end KONSTRUKT:

```
def typ=(typ)
 if !typ.is_ a?(String)
  puts "war kein String"
 elsif typ.size < 3
  puts "String zu kurz"
 else
  @typ = typ.capitalize
 end
end
```

Kurzform der if-ANweisung:

```
def ps=(ps)
 @ps = ps if ps > 0
end
```

Steht am Ende einer Anweisung (in der gleichen Zeile) eine if-Bedinung, dann wird die vorhergehende Anweisung (z.B.: Zuordnung eines Wertes) nur ausgeführt, wenn die Bedingung true ist.

Ternärer Operator:
Es gibt die Möglichkeit einen if-else-Block in einer Zeile auszuführen:

```
puts car.typ == "" ? "Kein Wert" : car.typ
```

Die Anweisung puts, also Erzeugen eine Ausgabe, bekommt den Wert 'Kein Wert' falls die Bedingung 'car.typ == ""' leer ist; bzw. den aktuellen Objekt-Wert für 'typ' falls dieser gefüllt ist.

Das Fragezeichen ist ein Operator der die linksstehende Bedingung von den auszuführenden Anweisungen trennt.
Der Doppelpunkt fungiert als Kurzform von 'else'

### 2.2.2 array

Definition
Array ist eine geordnete Liste, die Verweise auf andere Objekte enthält.

Neues Array anlegen
```
seats = []
seats = Array.new
seats = ["Wert1", "Wert2"]
seats = % w(Wert1 Wert2)
```

Werte hinzufügen
```
seats « "Wert3"
seats[3] = "Wert4"
seats.push("Wert5")
```

Methoden von Arrays
- .push()
  Hinzufügen eines Wertes
- .size
  Größe des Elements
- .pop
  Löschen des letzten Wertes
- .insert(Index, "Wert")
  Setzt einen Wert auf genanntem index
- .empty?
  Löschen der Einträge
- .delete_ at(Index)
  Löschen des Wertes am Index

Iterations-Methoden:

var.each
Methode zum Durchlaufen der einzelnen Elemente

var.select
Methode, die im Block einen boolschen Wert erwartet und bei true den Inhalt zurückgibt
var.select |v| v.size > 1

var.reject
opposite von select

var.sort
gibt ein sortiertes Array zurück

3. times puts "irgendwas"
gibt 3 * "irgendwas" aus

numbers = (1..10).to_ a
erstellt ein Array von 1-10

var = numbers.select |n| n > 4
speichern von 5 6 7 8 9 10 in var

var = numbers.select |n| n.even?
alle geraden Zahlen aus numbers werden in var gespeichert

var = numbers.reject |n| n.even?
dreht die select Methode um. So werden alle ungeraden Zahlen in var gepeichert

evens, odd = numbers.partition |n| n.even?
teilt das Ergebnis in die Arrays evens und odd auf.
In even werden die positiven Ergebnisse der Abfrage 2 4 6 usw gespeichert und in odd die negativen Ergebnisse 1 3 5 usw.

names = % w(anhalten gehen sitzen)
macht ein Array aus den 3 Angaben

puts names.sort_ by |n| n.length
sortiert das array nach der kürzesten Länge abwärts also wäre "gehen" an Array postion 0

puts names.sort_ by |n| n.length.reverse
dreht das Ergebnis um

### 2.2.3 bloecke

Ruby verwendet für Definitionen, Wiederholungen und Abfragen ein Blockkonstrukt.

Bsp:

```
3.times do
  puts "Irgendwas"
end
```

Bei der Wiederholung beginnt der Block mit einem do und endet mit einem end.

### 2.2.4 essentials

Konsole:

irb

die Eingabe des Befehls irb auf der Linuxconsole öffnet eine Ruby- Console, in der alle Ruby-Klassen bekannt sind und genutzt werden können

Arbeiten mit Dateien:

Ruby-Code in Dateien wird auf der Linux-Console mit dem Befehl:

ruby datei_ name

ausgeführt.

Basismethoden:

puts

Erzeugt eine Consolenausgabe

### 2.2.5 links

Dokumentation für Klassen:

Ruby Doc

Ruby lang Doku

Tutorials:

tutorialspoint

### 2.2.6 variablen

Ruby ist eine interpretierende Sprache, d.h. es ist keine Datentyp-Festlegung der Variablen nötig.
Der Interpreter erkennt an dem zugewiesenen Wert um welchen Datentyp es sich handelt.

Bsp:

```
var = "Ein String"  var ist eine Objekt der Klasse String
var.class    erzeugt Ausgabe String, also den Datentyp
```

Besonderheiten von Strings:

Konkatenieren:

```
var = "String"
puts "Dies ist ein " + var
```

Ausgeben von Ruby-Code im String:

Das Konstrukt: " Zeichenkette # ruby code " ermöglicht es in einer Zeichenkette Code auszuführen und Variablen einzufügen.

Bsp:

```
movie = "Goonies"
rank = 10
puts "#movie hat am #Time.now das Ranking #rank"
```

### 2.2.7 hash

Ein HASH ist ein assoziatives Array, d.h. ein Array mit einem selbstdefinierbaren Index.

```
  cars =  'eins' => 'Audi', 'zwei' => 'BMW'
```

Der Entwickler von Ruby hat aus Performance-Gründen für den Hash-Index das Symbol vorgesehen:

```
  cars =  :eins => 'Audi', :zwei => 'BMW'
```

Kurzschreibweise:

```
  cars =  eins: "Audi", zwei: "BMV"
```

Iteration über einen Hash:

```
  cars.each do |key, val|
    puts "Index: #key ist ein #val"
  end
```

Beispiele

```
puts names.sort by |n| n.length.reverse.class
  #gibt "Array" aus, da die Klasse ein Array ist
cars =
cars = 'eins' => 'BMW', 'zwei' => 'Audi'
  #erstellt ein assosiatives array
cars['zwei']
  #gibt "Audi" aus
```

### 2.2.8 vererbung

Vererbung:
Die OOP basiert auf dem Prinzip der Überschaubarkeit. Deshalb wurde die Vererbung als grundlegendes Merkmal eingeführt.
Erbt eine Klasse von einer anderen Klasse (z.B.: class Lkw < Car), dann stehen in der erbenden Klasse alle Methoden und Eigenschaften zur Verfügung, die in der Elternklasse definiert wurden.

Beispiel:

```
class Car
  attr_accessor :ps
end
class Lkw < Car
end
obj = Lkw.new
obj.ps = 50
puts obj.ps
```

Zugriffsmodifizierer
Es kommt vor, dass bestimmte Methoden in der Elternklasse nur innerhalb díeser Klasse benutzt werden sollen. Dann werden diese als 'private' gekennzeichnet.
Falls eine Methode in der Elternklasse nicht öffentlich sein soll, aber vererbt werden soll, dann wird diese als 'protected' gekennzeichnet.
Falls kein Zugriffsmodifizierer angegeben ist, gelten die definierten Methoden als 'public'.

```
class Car
  #von aussen erreichbar: public
  attr_ accessor :ps
  def beschleunigen
    ...
  end
  #von Kindklassen erreichbar: protected
  protected
  def umwandeln_ in_ kmh(var)
    ...
  end
  #nur innerhalb der Elternklasse aufrufbar: private
  private
    def irgendwas
      ...
    end
end
```

### 2.2.9 classes

OOP
Klassen
Klassen bilden den Plan von konkreten Objekten.
Def:

```
class Player
  ...
end
```

Eine Klasse wird durch die Verwendung des Schlüsselwortes class definiert.
Die Definition ist immer in einen Block eingeschlossen.

Objekt:
Wenn ich mit einer Klasse arbeiten will, dann erzeuge ich ein Objekt, das in einer Variable gespeichert wird.

```
obj = Player.new
```

Ein neues Objekt wird in einer Variable mit beliebiger Bezeichnung gespeichert.
Es wird mit dem Namen der Klasse und dem Aufruf der Methode new erzeugt.
Das Objekt hat alle Eigenschaften (Attribute) und Fähigkeiten (Methoden), die in der Klasse definiert sind.

Konstruktor:
Beim Erzeugen eines neuen Objektes mit Player.new wird automatisch eine Methode 'initialize' aufgerufen.
Diese Methode gibt es in jeder Ruby-Klasse. Es ist üblich diese existierende Methode mit einer eigenen Methode zu überschreiben, um dort die Basiswerte des Objektes festzulegen.
Bsp:

```
def initialize(name, health)
  @name = name
  @health = health
end
```

Instanzvariable / Eigenschaften
In der OOP hat ein Objekt bestimmte Eigenschaften, die es charakterisieren.
In Ruby werden diese Eigenschaften als Instanzvariablen definiert und ein '@' wird dem Namen vorangestellt.
Beispiel: @name
Das Besondere einer Instanzvariable oder Eigenschaft ist, dass sie in der ganzen Klasse gilt und verwertet werden kann.
Wichtig: Variablen die innerhalb von einem Block definiert werden und kein @ vorangestellt haben, sind sogenannte lokale Variablen und gelten nur innerhalb des Blockes in dem sie definiert sind.

Eine Instanzvariable wird zu einer Eigenschaft, indem es in der Klasse Methoden gibt, die den Zugriff von außerhalb (Aufruf durch das Objekt) ermöglichen.
Eine Eigenschaft hat einen sogenannten Setter, der es ermöglicht einen Wert zu setzen; und einen Getter, der einen Wert zurückgibt.

Getter:

```
def name
  @name
end
```

Die Methode heisst genauso wie die Instanzvariable, ohne das @. Sie hat die Aufgabe, den Wert der Instanzvariable name zurückzugeben.

Aufruf:

```
puts obj.name
  gibt den Namen des Players aus
```

Kurzform:

Anstelle einer Methodendefinition kann ich eine Kurzform verwenden:

```
attr_ reader :name
```

Setter:

```
def name=(name)
  @name = name
end
```

Der Setter zum Speichern eines Wertes in einer Instanzvariablen und heisst wie die Instanzvariable mit einem = im Namen.

Aufruf:

```
obj.name = "Supergirl"
```

Kurzform:

```
attr_ writer :name
```

Kurzform für Setter und Getter:

```
attr_ accessor :name
```

Virtuelles Attribut:

Eine virtuelles Attribut ist eine Methode, die so aussieht, bzw. genutzt werden kann, wie eine Eigenschaft, die allerdings keine eigene Instanzvariable hat; also keine echte Eigenschaft ist, obwohl sie von außen so aussieht.

```
def normalized_ health
  @health * 1000
end
```

Aufruf:

```
puts obj.normalized_ health
```

### 2.2.10 vorbereitung_ kla

Um Ruby Code auszuführen kann ich entweder

mit irb
  eine Konsole öffnen um Life Code zu produzieren und auszuführen

mit ruby <Dateiname>
  greifen wir auf eine Datei zu, in welcher Ruby Code steht und führen diesen aus.

1. erstellen sie eine klasse Car mit den Eigenschaften: ps, typ, verbrauch
2. Das Fahrzeug kann beschleunigen, die Methode Beschleunigen bekommt einen Parameter zeit,
   der die Sekundenzahl des Beschleunigungsvorgangs enthält.
   Ausgegeben wird die erreichte Geschwindigkeit in km/h.
   Die Geschwindigkeit wird errechnet aus PS mal Sekunden geteilt durch 100 und ergibt die Einheit meter/sekunden.
3. Erstellen Sie einen Konstruktor,
   der für die Eigenschaft ps einen Wert erwartet, für die Eigenschaft optional eine Initialisierung ermöglicht
4. legen Sie 3 Objekte von dieser Klasse an
   Objekt1 ps = 200 typ='Klaumich';Objekt2 ps = 80 typ 'hauni'; Objekt 3 = 100PS
5. Speichern Sie die Objekte in einem Array
   Lassen Sie das Array durchlaufen geben Sie den Typ aus,
   wenn kein Typ angegeben ist soll "Kein Wert" ausgegeben werden.
6. Legen Sie drei weitere Objekte von der Klasse Car an.
   Die Daten stehen in einem hash, wobei die erste Position die den Typ und die zweite Postion PS enthält:
   vals = { 'BMW' => 200, 'trabbi' => 150, 'Honda'=> 20} .
   Lassen Sie den Hash durchlaufen, erzeugen Sie jeweils ein Car Objekt und legen dieses im Array cars2 ab.
7. Erstellen Sie eine Klasse 'Driver'. mit der Eigenschaft "name" die im Konstrukor gesetzt werden muss.
   Es gibt ein Array cars, dass im Konstruktor initialisiert wird.
   Es gibt eine methode add_ cars mit der für das Driver Objekt ein Fahrzeug hinzugefügt werden kann.
   Es gibt eine Ausgabe show_ cars die die Fahrzeuge mit der Typenbezeichnung auflistet.
   Legen Sie ein Objekt Theo an und fügen Sie von den car arrays jeweils das Erste dem Driver Objekt zu.
   Lassen Sie die Daten der zugeordneten Fahrzeuge ausgeben.

elsif = else if

public
Allgemeiner Bereich für Methoden, diese sind von außen zugänglich
private
Methode nur innerhalb der Klasse
protected
Methoden um mit Vererbung zu arbeiten

---

Ruby spezifisches Problem
Es gibt für eine if Version eine Kurzform
Variable gegen eine einzige Bedingung prüfen geht mit @ps = ps if ps > 0

---

3 Sätze zu Klassen

- Es ist ein Bauplan
- ich habe die Möglichkeit geschaffen Objekte zu erschaffen
- Besteht aus Eigenschaften(Instanzvariablen, Methoden) und Fähigkeiten(Definiert in Methoden)

---

Anmerkung zu Aufgabe 1:

Eigenschaften haben nichts mit der initialize Methode zu tun
Sie werden im attr_ accessor definiert

Erweiterung von Aufgabe 1 z.B.
"stellen Sie sicher das die Eigenschaft typ beim ersten Buchstaben
einen Großbuchstaben erstellt"
"Stellen Sie sicher, dass bei der Eingabe einer Zahl keine Ausnahme geworfen wird"

2te Erweiterung
"Stellen Sie sicher, dass es sich bei typ um einen String handelt der mind. 3 Ziffern lang ist"

3te Erweiterung
"Stellen Sie sicher, dass es sich bei typ um einen String handelt der mind. 3 Ziffern lang ist"
"Stellen Sie sicher das die Instanzvariable PS nur dann gefüllt wird, wenn es sich um einen positiven Wert größer als Null handelt"

## 2.3 Unterricht

### 2.3.1 car.rb

```ruby
class Car
   attr_accessor :verbrauch
   attr_reader :typ, :ps

   def initialize(ps, typ="")
      @ps = ps
      @typ = typ
   end
   def ps=(ps)
      @ps = ps if ps > 0
   end

   def typ=(typ)
      if !typ.is_a?(String)
         puts "war kein String"
      elsif typ.size < 3
         puts "String zu kurz"
      else
         @typ = typ.capitalize
      end
   end

   def beschleunigen(zeit)
      geschwindigkeit = @ps * zeit / 100

      puts "Die Geschwindigkeit beträgt nun: #{convert_to_kmh(geschwindigkeit)} km
      pro Stunde."
   end

   private
      def convert_to_kmh(var)
         var * 3.6
      end
end


car1 = Car.new(200, 'Klaumich')
car2 = Car.new(80, 'Hauni')
car3 = Car.new(100)

cars = [car1, car2, car3]
cars.each do |car|
   #puts car.typ.empty? ? "Kein Wert" : car.typ
end

vals = { 'BMW' => 200, 'Trabbi' => 150, 'Honda' => 20}
cars2 = []
vals.each do | typ, ps |
   cars2 << Car.new(ps, typ)
end
#puts cars2.inspect
#puts cars2[1].typ

cars2 << cars
```

```ruby
cars2[3].each do |car|
   #puts car.typ
end

class Driver
   attr_accessor :name
   def initialize(name)
      @name = name
      @cars = []
   end
   def add_cars(car)
      @cars << car
   end
   def show_cars
      @cars.each do |car|
         puts car.typ
      end
   end
end
theo = Driver.new('Theo')
theo.add_cars(cars[0])
theo.add_cars(cars2[0])
theo.show_cars
```

### 2.3.2 flicks.rb

```
class Playlist
  def initialize(name)
    @name = name
    @movies = []
  end
  def add_movie(movie)
    @movies << movie
  end
  def play
    puts "#{@name} Playlist:"
    @movies.each do |movie|
      puts movie
    end
  end
end
class Movie
  attr_accessor :title
  def initialize(title, rank=0)
    @title = capitalize_title(title)
    @rank = rank
  end
  def capitalize_title(str)
    str_array = str.split
    ret = ""
    str_array.each do |teilstring|
      ret = ret + " " + teilstring.capitalize
    end
    ret.strip
  end
  def thumbs_down
    @rank -= 1
  end
  def to_s
    "#{@title} has a rank of #{@rank}."
  end
end

movie1 = Movie.new("Goonies super", 10)
puts movie1.title
#movie2 = Movie.new("Batman", 5)
#movie3 = Movie.new("Spiderman", 8)

#playlist1 = Playlist.new("Theo")
#playlist1.add_movie(movie1)
#playlist1.add_movie(movie2)
#playlist1.play
```

### 2.3.3 start.rb

```ruby
class Movie

  attr_reader :title
  attr_writer :title
  attr_accessor :title

  def initialize(title, rank=0)
    @title = title.capitalize
    @rank = rank
  end

  def normalized_rank
   @rank * 1000
  end
  #Getter für einen veränderten rank
  def thumbs_up
    @rank += 1
  end

  def thumbs_down
    @rank -= 1
  end

  def to_s
    "#{@title} has a rank of #{@rank}."
  end
end

movie = Movie.new("Goonies", 10)
puts movie.title
movie.title = "GYnniies"
puts movie.title
puts movie.normalized_rank
```

### 2.3.4 studio_ game.rb

```ruby
class Player

  attr_accessor :name
  attr_reader :health

  def initialize(name, health=100)
    @name = name.capitalize
    @health = health
  end

  def to_s
    "I'm #{@name} with a health of #{@health} and a score of #{score}."
  end

  def blam
    @health -= 10
    puts "#{@name} got blammed!"
  end

  def w00t
    @health += 15
    puts "#{@name} got w00ted!"
  end

  def score
    @health + @name.length
  end

end

player1 = Player.new("moe")
player2 = Player.new("larry", 60)
player3 = Player.new("curly", 125)

players = [player1, player2, player3]

puts "There are #{players.size} players in the game:"
players.each do |player|
  puts player
end

players.each do |player|
  puts player.health
end

players.each do |player|
  player.blam
  player.w00t
  player.w00t
  puts player
end

players.pop
player4 = Player.new("Shemp", 90)
players.push(player4)
puts players
```

### 2.3.5 test.rb

```ruby
class Customer
   #Methode zum Setzen der Instanzvariable @name
   #def set_name(name)
   #  @name = name
   #end
   #Kurzform fuer Eigenschaft name
   attr_writer :name
   #Langform fuer Eigenschaft name
   #def name=(name)
   #  @name = name
   #end
   def ausgabe
      @name
   end
end
obj = Customer.new
#Eine Variable kann gesetzt werden, indem ich eine beliebige Methode aufrufe
#obj.set_name("Theo")

obj.name = "Theo"
puts obj.ausgabe
```

# 3 Rails

## 3.1 Dokumentation

### 3.1.1 active_record_commands.txt

Alle Befehle mit der Model-Klasse Event.
Anlegen eines neuen Events:
  e = Event.new
  e.name = "Eventname"
  ...
  e.save

Anzeigen aller Events:
  Event.all

Suchen eines speziellen Events:
  e = Event.find(id)

  e = Event.find_by(name: "Eventname")

  e = Event.where(name: "Eventname").first

Beziehungen zwischen den Tabellen events + registrations
  e = Event.first
  e.registrations.create
    über den konkreten Event (Objekt: e) kann eine Registrierung angelegt werden

  e.registrations
    zeigt alle Registrierungen, die diesem Event zugeordnet sind

  r = Registration.last
  r.event
    auf das Event-Objekt dem die Registrierung zugeordnet ist, zugreifen
  r.event.title
    der Titel des Events kann ausgegeben werden

### 3.1.2 configurations.txt

Für Rails gilt der Grundsatz: Convention Over Configuration, das bedeutet,
dass der Programmierer versucht so wenig wie irgend möglich zu konfigurieren.

Für die Standardkonfiguration gibt es im Ordner 'config' einige Dateien und Möglichkeiten.
a. Basiseinstellungen über die Zeit- und Länderzone:
  config/application.rb
    config.i18n.available_locals = ["de_DE", :en]
    config.time_zone = "Berlin"
    config.i18n.default_locate = :'de_DE'

b. Ändern des Default_Formats für Zeit-Datum
  config/initializers/time_format.rb
    Time::DATE_FORMATS[:default] = "%d.%m.%Y %H:%M"

  Überall in der Webapplikation gilt dieses Format als Standardformat bei der Darstellung vom
  DateTime-Datentyp.

### 3.1.3 helpers.txt

Es gibt Helper sowohl auf der Darstellungsebene (View-Html-Bereich) aus auch auf der Logik-Ebene im Bereich der Controller.

VIEW-Ebene
a. bereitgestellte Helper
Helper, die von den Rails-Entwicklern zur Verfügung gestellt werden.
Links: Ruby on Rails Helper-Liste

Beispiele:

```
truncate
content_tag
number_to_currency
```

b. eigene Helper
Gründe

- falls kein imannenter Helper zur Verfügung steht
- wenn ein Helper den Code in der HTML-Datei übersichtlicher/eleganter macht
- wenn Code sonst doppelt vorkommt

Es gibt im Bereich app einen speziellen Ordner helpers, in dem der Rails-Server bei jedem Aufruf der Seite nach Funktionen scannt. Dort werden eigene Helper als einfache Methoden definiert.
Beispiel:
Definition
app/helpers/events_helper.rb

```
    def format\_price(event)
       if event.free?
          #<strong>Free</strong>".html_safe
          content_tag(:strong, 'FREE')
       else
          number_to_currency event.price
       end

    end
```

Aufruf:
app/views/events/index.html.erb

```
<%= format_price(event) %>
```

### 3.1.4 basis.txt

Anlegen eines neues Projektes

```
rails new name_des_projektes
```

Ordnerstruktur
  app
    controller
    model
    view

  config

Prinzip MVC
  Rails realisiert das Pattern: ModelViewController,
  Model
    enthält alles zum Thema Daten
  View
    enthält die Darstellung
  Controller
    die Steuerzentrale der Applikation mit der Logik

Verwalten der Module
  Eine Railsapplikation besteht im Hintergrund aus einer Vielzahl von Modulen, die gems genannt werden.
  Diese Module sind in der Datei Gemfile eines Projektes organisiert.
  Neue Module oder Funktionen werden einer Applikation hinzugefügt, indem sie im Gemfile definiert werden.
  Nach jeder Änderung der Datei Gemfile muss auf der Console im Projektverzeichnis der Befehl
  <bundle install> abgesetzt werden

Server starten
  Ein bereits installiertes Modul heisst: puma. Dieses enthält einen internen Web-Ruby-Server

```
rails s
```

  Aufruf mit Browser:
    http://localhost:3000

Ablauf der Seitenerstellung:
  Aufruf vom Browser: localhost:3000/events
  Was passiert in der Applikation:
    1. Gibt es diese Resource in der routes.rb
       (get "events" => "events#index")
    2. Gibt es einen Controller mit dem Namen: EventsController
    3. Gibt es eine Action (Methode) mit dem Namen index
    4. Gibt es eine Datei im Views-Ordner in dem Unterordner mit der Bezeichnung
       des Controllers (events) und dem Namen der Action (index.html.erb)

### 3.1.5 layout.txt

ASSETS
Im Ordner app/assets werden alle Stylesheets, Javascript-Dateien und Bilder abgelegt.

JAVASCRIPT
/app/javascripts/application.js
Für Rails die Startdatei. Hier werden alle Dateien, die geladen werden sollen notiert
z.B.: //= require jquery bindet jquery ein

STYLESHEETS
/app/stylesheets/application.css
Rails Startdatei, alle hier notierten Anweisungen werden ausgeführt
z.B.: *= require styles bindet die Datei styles.css ein

IMAGES
Rails sucht standardmäßig alle Bilder die mit dem Helper Image_tag eingebunden werden im Ordner:
app/assets/images
z.B.: <%= image_tag 'logo.jpg' %>

LAYOUT IN RAILS
Standardmäßig gibt es in app/views einen layouts - Ordner.
Dort sind alle Dateien, die dem Corporate Design der App zugeordnet sind, d.h. Elemente,
die bei allen Seitenaufrufen identisch sind, wie etwa Header, footer, navigation.
Wenn der Dispatcher die Anweisungen im Controller abgearbeitet hat
und die Seiten zum Browser schicken will, dann sucht er standardmäig die Datei:
app/views/layouts/application.html.erb.
In dieser Seite gibt es den Aufruf <%= yield %>
in dem der jeweils spezielle Seiteninhalt eingefügt wird.

FOUNDATION
Zur-Foundation ist ein CSS-Framework, das mit Rails gut zusammenarbeitet.
Das Einbinden wird unter: http://foundation.zurb.com/sites/docs/v/ 5.5.3/applications.html beschrieben.
Siehe ausführliche Dokumentation.

### 3.1.6 validations.txt

Link:
http://guides.rubyonrails.org/active_record_validations.html

Rails hat standardmäßig eine Validierungsfunktion für Usereingaben.
Jedes Datenbank-Tabellenfeld kann in der jeweiligen Model-Klasse überprüft werden.
Vor dem Speichern eines Datensatzes überprüft Rails, ob alle im model definierten Validierungen true ergeben.

Model:
Beispiele:

```
validates :name, :description, :location, presence: true
validates :description, length: { minimum: 25 }
validates :price, numericality: { greater_than_or_equal_to: 0 }
```

### 3.1.7 routing.txt

Die Verbindung zwischen Browseraufruf und Serverantwort wird in Rails mit Hilfe der Datei config/routes.rb hergestellt; damit der Server eine aufgerufene URL findet muss sie mit einem Eintrag in der routes.rb matchen.

BASISURL

```
root 'events#index'
```

der Aufruf localhost:3000 wird nun zur index-Action des Events- Controllers weitergeleitet

Resourcen

```
resources :events
```

Mit der Funktion resources und dem Namen eines Controller als Symbol,
wird beim Start des Servers gesichert, dass ein Aufruf von:
http://localhost:3000/events
bei index-Action des Events-Controllers landet

Verschachtelte Resourcen:

```
resources :events do
    resources :registrations
end
```

Die Resource Registrations kann nur in Verbindung mit einem Event aufgerufen, gespeichert, gelöscht werden. Der Browseraufruf lautet dann:
http://localhost:3000/events/4/registrations

Methode zur Kontrolle der Resourcen:

```
rake routes
```

gibt alle in der routes.rb definierten Routen aus

### 3.1.8 rails.txt

Der Befehl rails wird immer im Basisordner der Applikation ausgeführt.
Das Programm rails ermöglicht auf der Console ausgeführt eine Reihe von Aktionen:

```
rails new projcect_name
```

Erstellt ein neues Projekt

```
rails s
```

startet einen Server, der auf Port 3000 lauscht

```
rails g controller events
```

erstellt in der App einen Controller mit dem Namen movies _controller.rb

```
rails d controller events
```

löscht die Dateien die durch den Generator erstellt wurden

```
rails g model event name:string location:string price:decimal
```

es werden Dateien zur Erstellung einer Tabelle erzeugt,
mit dem Namen events und den Spalten: name, location, price

```
rails g scaffold event name:string location:string price:decimal
```

es wird eine komplette Resource ersteltt; d.h.:

- in routes rb wird die Zeile hinzugefügt:
  resources :event
- Dateien zur Erstellung einer Tabelle erzeugt,
  mit dem Namen events und den Spalten: name, location, price
- Controller mit allen Actions
- View-Ordner events mit allen Dateien

### 3.1.9 patterns.txt

MVC
  Model - View -Controller auf verschiedenen, voneinader getrennten Ebenen

Convention Over Configuration
  Es gibt Regeln die eine Automatisierung durch den rails generator ermöglichen
  z.B:
    controller sind immer im Plural und die Klasse wird im CamelCasing benannt:
      movies_controller.rb
        class MoviesController
    models sind im Singular benannt:
      app/models/event.rb
        class Event
    Die Tabellen, die zu den Models gehören sind immer im Plural
      Tabelle heisst: events

  Prinzip der geringsten Überraschung
    Der User sollte Funktionen und Methoden erraten können

CRUD (Create, Read, Update, Delete)
  Die Standardactions einer Webandwendung:

    index: alle Elemente werden gezeigt
    show: ein einzelnes Element wird angezeigt
    new: Formular zur Erzeugung eines Elements
    create: Action zum SPeichern eines neuen Elements
    edit: Formular zum Verändern eines elements
    update: Action zur Durchführung der Änderung in der Tabelle
    destroy: Action zum Löschen eines Elements

### 3.1.10 nested_resource.txt

1. Die Definition der Abhängigkeit von Resourcen wird in der config/routes.rb festgelegt.

```
resources :events do
     resources :registrations
end
```

   Die Registrations-Resource ist der Event-Resource untergeordnet

2. Falls die Resource registrations mit einem Scaffold angelegt wurde, müssen alle Links geändert werden.

3. Im Registrations-Controller sollte eine Before-Methode definiert sein,
   die bei jeder Action des Controllers eine Instanzvariable @event erzeugt.

```
before_action :set_event
 def set_event
   @event = Event.find(params[:event_id])
 end
```

4. In der Index-Action sollten nur die Elemente gefunden werden, die zu diesem Event gehören.

```
@registrations = @event.registrations
```

5. In der new und edit - Action muss sichergestellt sein, dass die Event-ID mitgeleifert wird:

```
@registration = @event.registrations.new
```

6. Im Formular muss der Link zur Create- und/oder Update - Action angepasst werden.

```
<%= form_for([@event,@registration]) do |f| %>
```

### 3.1.11 models.txt

Der Zugang zu Datenbank wird bei Rails in den Model-Klassen geregelt.

Die Datenbank-Einstellungen der Applikation sind in der Datei:
config/database.yml

Im Allgemeinen gilt:
einer Tabelle in der Datenbank entspricht eine Klasse im Unterordner app/models
Models sind grundsätzlich im Singular benannt:
Bsp:
app/models/event.rb
class Event
Die Tabellen, die zu den Models gehören sind immer im Plural
Bsp.:
Tabelle heisst: events

Erstellen von Tabellen/Models:
Der Befehl:

```
rails g model event name:string location:string price:decimal
```

erstellt:
Datei app/models/event.rb mit Klasse: class Event
Datei db/migrate/324242424_create_events.rb

Alle Änderungen an einer Datenbank werden mit Hilfe von Migrationen erstellt,
d.h. Ruby-Klassen in denen Tabellen neu angelegt oder verändert werden können.
Diese Migrations müssen mit einem seperaten Befehl zur Datenbank gesendet werden:

```
rake db:migrate
```

Beziehungen:
1:n - In der untergordneten Tabelle sind beliebig viele Elemente einem Element in der Haupttabelle zugeorednet.
Beispiel: Ein Event hat beliebig viele Teilnehmer.

Eine Beziehung wird in ActiveRecord über eine Definition in der Model- Klasse hergestellt:

```
class event
  has_many :members
```

```
class member
  belongs_to :event
```

Durch diese Zuordnung werden beim Starten des Rails-Server eine Vielzahl von Funktionen erstellt, die es ermöglichen über die eine Klasse auf die korresponiderende zuzugreifen.

### 3.1.12 migrations.txt

Link:
http://guides.rubyonrails.org/active_record_migrations.html

Prinzip der Migrations:

- Die Kommunikation zwischen Rails und der Datenbank findet über die Klasse ActiveRecord statt.
- Eine Klasse von ActiveRecord ist die Klasse Migrations. Diese Bibliothek regelt das Anlegen und Löschen von Tabellen ebenso wie Änderungen, d.h. hinzufügen, ändern, löschen von Spalten in einer Tabelle

Vorgehensweise:

```
rails g model name_des_models spalte1:string
```

Eine neue Klasse im Ordner app/models wird angelegt
Eine Datei im Ordner db/migrate wird angelegt.
Diese Datei enthält die Anweisungen, die der Befehl 'rake db:migrate' zur Tabelle schickt.

Befehle:

```
rails g model name_des_models spalte1:string
```

erzeugt neue Model- und Migrationsklassen

```
rails g migration AddFielsToNameDesModels neue_spalte:string
```

erzeugt eine neue Migrationsdatei in db/migrate

```
rake db:migrate
```

Führt die Änderungen aus der Migrations-Datei an der Tabelle in der Datenbank durch

```
rake db:migrate:status
```

Zeigt welche Migrations bereits durchgefüfhrt wurden (up)
oder welche noch nicht in der Tabelle wirksam sind (down)

```
rake db:rollback
```

Nimmt die Anweisungen der letzten Migration-Datei wieder zurück

```
rake db:seed
```

Führt die Anweisungen in der db/seeds.rb - Datei aus

```
rake db:reset
```

Löscht alle Tabellen in der Datenbank, führt alle Migrations aus, führt die seeds.rb aus

### 3.1.13 kla_notizen.txt

1. Erstellen Sie eine Applikation books.

```
rails new books
```

   Dann in den neuen Ordner navigieren mit "cd books/"

2. Nutzen Sie für diese Applikation das CSS-Framework Foundation.
   link: http://foundation.zurb.com/sites/docs/v/5.5.3/applications.html

   (a) gem 'foundation-rails' #unter Gemfile hinzufuegen
   (b) bundle install #in Konsole
   (c) rails g foundation:install #in Konsole -> Y
   (d) <%= javascript_include_tag "vendor/modernizr" %>
   in app/views/layouts/application.html.erb hinzufügen
   (e) <%= javascript_include_tag "application", 'data-turbolinks-track' => true %>
   muss auch vorhanden sein, ggf hinzufügen
   (f) unter app/assets/stylesheets unter application.css '*= require foundation' hinzufuegen
   (g) von http://foundation.zurb.com/sites/docs/top-bar.html die topbar kopieren und unter app/views/layouts als z.B. _top_bar.html.erb speichern
   (h) in die application.html.erb -> ' <%= render 'layouts/top_bar' %>' hinzufuegen

3. Erstellen Sie eine Resource books mit den Spalten: author, title, description (text), published_on (Date)

```
rails g scaffold book author:string title:string description:text
published_on:date
```

4. Stellen Sie sicher, dass beim Aufruf der Seite die index-Action der Resource Books aufgerufen wird.

```
root 'books#index'
```

5. Binden Sie die topbar ein. Erstellen Sie einen Link in der Navigationsleiste, der auf die Index-Seite verweist.

   - top_bar aus altem Projekt kopieren
   - Routenfehler beim Browseraufruf provozieren
   - in top_bar einmal die zeile <%= link_to "Index", root_path() %> einfügen

6. Erstellen Sie eine untergeordnete Resource Likes, die eine Spalte quality (integer) hat. Sichern Sie ab, dass ein Like immer einem Book-Objekt zugeordnet ist.

   (a) ganz normal scaffold

   (b) verschachteln in der routes.rb

   (c) alle links im views des untergeordneten Objekts den neuen Routen anpassen

   (d) Alle Variablenzuweisung im untergeordneten Controller ändern

```
  def index
    @likes = @book.likes
  end

  def new
    @like = @book.likes.new
  end

  def create
    @like = @book.likes.new(like_params)
      format.html { redirect_to book_likes_path(@book.id, @like.id),

  def update
       format.html { redirect_to book_likes_path(@book.id, @like.id),

  def destroy
        format.html { redirect_to book_likes_url(@book.id),
```

   (e) Instanzvariable im untergeordneten Controller erstellen

```
class LikesController < ApplicationController
  before_action :set_book

  ...

  private
    def set_book
      @book = Book.find(params[:book_id])
    end

  ...
end
```

   (f) Untergeordnetes model mit

```
   belongs_to
```

   versehen

   (g) Übergeordnetes Model mit

```
   has_many
```

   versehen

7. Stellen Sie sicher, dass der Benutzer bei der Neuanlage/Update eines Likes eine Zahl von 1 bis 5 eingeben muss. Ansonsten wird das Formular nicht gespeichert.

   Im Model

```
HOW_HEARD_OPTIONS = [
   1...5
]
validates :<Spaltenname>, inclusion: { in: HOW_HEARD_OPTIONS }
```

8. Erstellen Sie einen View-Helper, der für ein Book-Objekt den Durchschnitt aller Likes für dieses Objekt errechnet und auf der Book-Show-Seite ausgibt.

   App Helpers

   Wir holen uns das book und die Likes mit z.b. book.likes.each do und denn mal durchschnitt berechnen

```
def mittel(book)
   i=0
   book.likes.each do |f|
      i=i+f.quality
   end
   e=i/book.likes.count
end
```

# Stichwortverzeichnis