PM1/PT Ruby: Iteratoren und Objektverträge

Iteratoren für Zahlen, Zeichenketten,
Range, Array, Set und Hash

Objektvertrag: Der Basisiterator each und
die Iterator-Methoden von Enumerable

Iteratoren für ganze Zahlen

- ***lower.upto(upper) {|i| block }***

  Der Blockvariablen ***i*** werden nacheinander die Zahlen ***lower, lower + 1, …, upper*** zugewiesen . Für jedes ***i*** wird der ***block*** ausgeführt. Wenn ***upper < lower***, dann wird der block nicht ausgeführt. Der Wert des Iteratorausdrucks ist ***lower***.

- ***upper.downto(lower) { |i| block}***

  Der Blockvariable i werden nacheinander die Zahlen ***upper, upper -1, …, lower*** zugewiesen. Für jedes ***i*** wird der ***block*** ausgeführt. Wenn ***upper < lower***, dann wird der block nicht ausgeführt. Der Wert des Iteratorausdrucks ist ***upper***.

- ***n.times { |i| block}***

  Der Blockvariable ***i*** werden nacheinander die Zahlen ***0, 1, 2, …, times-1*** zugewiesen und dann der block ausgeführt. Wenn ***times < 0***, dann wird der block nicht ausgeführt. Der Wert des Iterator-ausdrucks ist ***times .***

- **Merke**: Diese Iteratoren sind nur für ganze Zahlen definiert!

Iteratoren für Intervalle (*Range*)

- Den Zahleniteratoren ***upto*** und ***downto*** können wir keine Schrittweite mitgeben. Dazu benötigt man den Iterator ***step*** der Klasse ***Range:***

- ***rng.step(n=1) {|i| block}***

  Der Blockvariablen i werden nacheinander die Elemente ***rng.first, rng.first + n, … rng.last*** bzw. ***rng.last - 1*** für nach oben offene Intervalle zugewiesen. Für jedes ***i*** wird der ***block*** ausgeführt. Wenn ***rng.first > rng.last*** dann wird der ***block*** nicht ausgeführt. Wenn ***rng.first == rng.last***, dann wird ***block*** nur dann ausgeführt, wenn es sich um ein nach oben geschlossenes Intervall handelt. ***n*** gibt die **Schrittweite** an und ist mit 1 vorbelegt.

- **Ü-7-b-1:** Wenn Sie ***step*** auf den Range ***aaa..aba*** anwenden, welche Werte nimmt dann die Blockvariable ***i*** an? Überprüfen Sie dies bitte mit einer geeigneten Ausgabe auf der Konsole!

Iteratoren für Zeichenketten

Ruby 1.8

- String ist ein Enumerable und implementiert den Basisiterator each und hat daher alle Iteratormethoden von Enumerable.
  - each(separator= $/) {|t_str| block} zerlegt einen String in Teilstrings, die zwischen den Trennzeichen stehen. Der Default ist ein Zeilenumbruch
- spezielle Iteratoren:
  - each_byte: Liefert die Bytes eines String
  - each_line arbeitet wie each

Ruby 1.9 und höher

- String ist kein Enumerable mehr, den Basisiterator each gibt es nicht mehr.
- Es gibt nur noch spezielle Iteratoren für String:
  - each_byte: wie in 1.8
  - each_line: wie in 1.8
  - each_char: liefert die Zeichen eines String.
  - each_codepoint: liefert den Unicode (2 Byte) als Zahlenwert.

Der Basisiterator *each* und *Enumerable*

Class: Range

In: range.c
lib/pp.rb

Parent: Object

Methods

== === begin each end eql? exclude_end? first hash include? inspect last member? new pretty_print step to_s

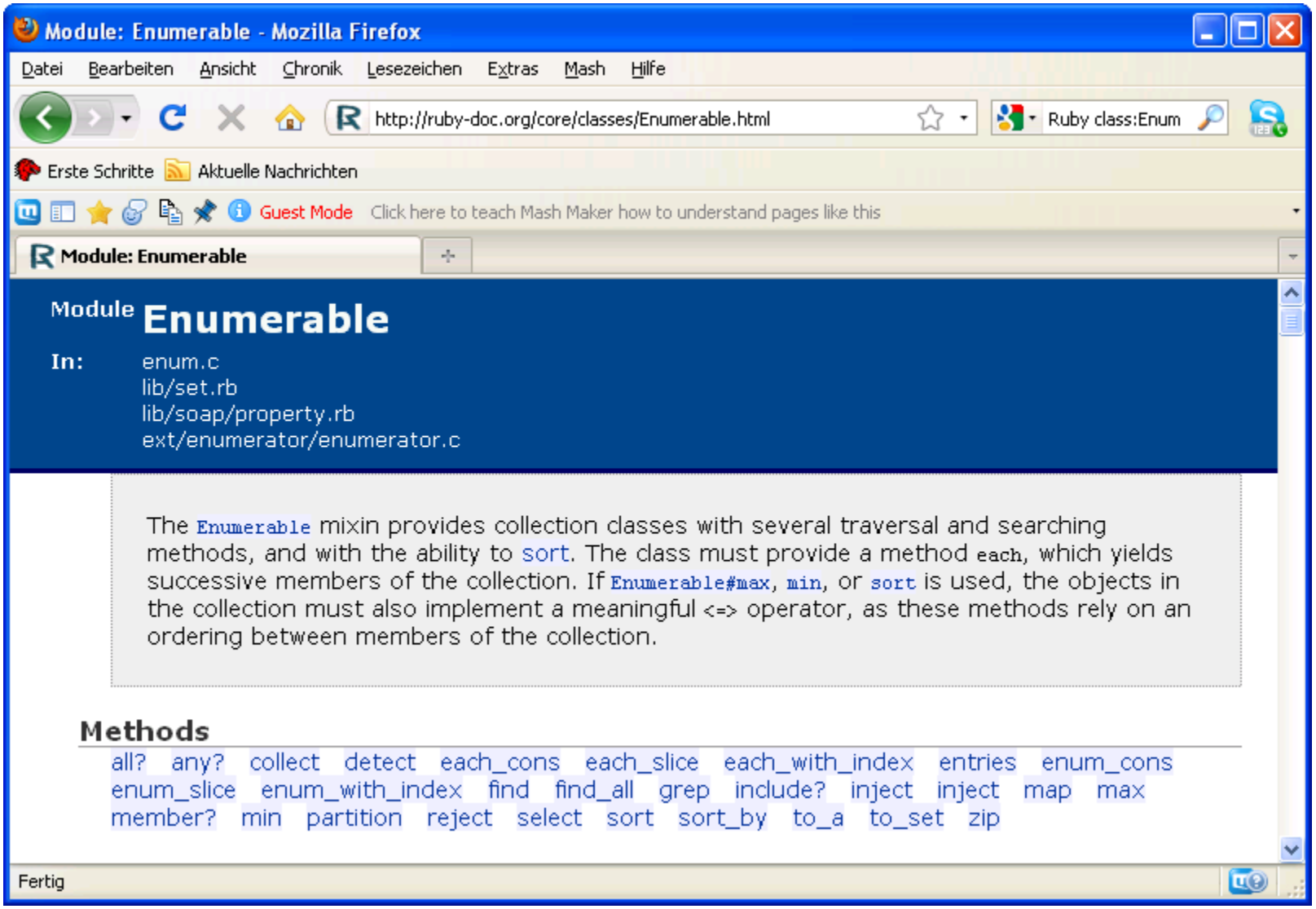
Included Modules

Enumerable

Public Class methods

- ***Range*** implementiert wie ***Array, Set*** und ***Hash*** den Basisiterator ***each.***
- ***Range*** inkludiert wie ***Array, Set*** und ***Hash*** das Modul ***Enumerable.***
- ***Enumerable*** führt alle Iteratormethoden alle auf den Basisiterator ***each*** zurück, den er jedoch selber nicht implementieren kann, da dieser von den speziellen Eigenschaften einer Klasse abhängt.
- Alle Objekte der Klassen ***Range, Array, Set*** und ***Hash*** bekommen über ***Enumerable*** deren Iterator-Methoden.

Das Modul *Enumerable*



Objektverträge

- Ein **Vertrag** zwischen Objekten regelt, welche Teilfunktionalität die einzelnen Klassen zur Erreichung einer Gesamtfunktionalität beitragen müssen.
- Jeder der beteiligten Partner sichert in dem Vertrag die Erfüllung der Teilfunktionalität zu.
- In Beispiel von ***Enumerable*** und den Klassen ***Range, Array, Set*** und ***Hash***, übernimmt der Vertragspartner Enumerable die Implementierung der Iteratormethoden, die anderen Vertragspartner sichern zu, den Basisiterator ***each*** zu implementieren.
- Die Aufgaben sind nicht wirklich gerecht verteilt, Enumerable‘s Beitrag liegt um einiges höher als die der anderen Partner.
- Dahinter steckt das Konzept von Framework-Klassen, die aufbauend auf einer mininalen Funktionalität von Benutzergeschriebenen Klassen umfangreiche Dienste anbieten.
- Wir werden im weiteren Verlauf der Vorlesung noch andere Verträge kennen lernen.
- Einige kennen wir bereits, die Verträge zwischen ***Kernel.puts*** / ***Kernel.p*** und den Methoden ***to_s*** bzw. ***inspect*** in eigenen Klassen.

Die Iterator-Methoden von *Enumerable*

Methoden ohne Zusatzanforderungen für enthaltene Elemente

- Prädikate:
  - ***any?, all?, include? (member?)***
- Auffinden
  - ***find (detect), find_all (select), reject***
- Sammeln und/oder Abbilden
  - ***collect (map)***
- Sortieren nach Eigenschaften
  - ***sort_by***

Methoden für Elemente, auf denen eine Ordnungsrelation definiert sein muss

- ***min, max, sort***
- Um das Minimum oder Maximum einer Sammlung zu bestimmen, müssen Elemente der Sammlung bzgl. der Größer-Kleiner Relation vergleichbar sein.
- Gleiches gilt auch für die Methode ***sort***, da Sortierung auf dem Größer-Kleiner Vergleich basiert.
- Für Zahlen, Zeichenketten und Arrays ist diese Ordnungsrelation in Ruby definiert.

Die Prädikate *all?* und *any?* für Objektsammlungen

sam.all? {|elem| block }

- ***all?*** prüft, ob für alle Elemente einer Ob-jektsammlung (***sam***) die Bedingung im ***block*** erfüllt ist.
- ***all?*** bricht ab, sobald ein Element der Sammlung (***sam***) die Bedingung nicht er-füllt.
- Das Ergebnis von ***all?*** ist **true**, wenn alle Elemente die Bedingung erfüllen, sonst ***false***.
- Die Bedingung muss als boolescher Aus-druck im ***block*** definiert sein.
- ***all?*** ist eine Implementierung des Allquan-tors (∀), den Sie in der Vorlesung Logik noch kennenlernen werden oder vielleicht aus der Mathematik erinnern.

Beispiele

- Sind alle Elemente der Sammlung < 10?

```
sam.all?{ |elem| elem < 10 }
```

- Beginnen alle Elemente einer Sammlung mit ***"A"*** ?

```
sam.all?{|elem| elem[0,1]=="A"}
```

- Haben alle Schlüssel eines Hashes eine Länge < 10?

```
sam.all?{|k,v| k.length() < 10 }
```

- Sind alle Werte eines Hashes, gerade Zahlen?

```
sam.all?{|kv_paar|
                kv_paar[1]%2 == 0 }
```

Die Prädikate *all?* und *any?* für Objektsammlungen

sam.any? {|elem| block }

- ***any?*** prüft, ob für **mindestens ein** Element einer Objektsammlung (***sam***) die Bedingung im ***block*** erfüllt ist.
- ***any?*** bricht ab, sobald **ein** Element der Sammlung (***sam***) die Bedingung erfüllt.
- Das Ergebnis von ***any?*** ist ***true***, wenn ein Element die Bedingung erfüllt, sonst ***false***.
- ***any?*** ist eine Implementierung des Existenzquantors (∃). Auch diesen werden Sie in der Vorlesung Logik noch kennen-lernen werden.

Beispiele

- Existiert ein Element in der Sammlung, dass < 10 ist?

```
sam.any?{ |elem| elem < 10 }
```

- Existiert ein Element in der Sammlung, das mit *"A"* beginnt?

```
sam.any?{|elem| elem[0,1]=="A"}
```

- Gibt es einen Schlüssel im Hash, der eine Länge < 10 hat?

```
sam.any?{|k,v| k.length() < 10 }
```

- Gibt es einen Wert im Hash, der eine gerade Zahl ist?

```
sam.any?{|kv_paar|
                kv_paar[1]%2 == 0 }
```

Übungen

- **Ü-7-b-2:** Schreiben Sie bitte eine Methode, die prüft, ob alle Namen in einer Sammlung (***Array*** oder ***Set***) auf ***".rb"*** enden!
- **Ü-7-b-3:** Schreiben Sie bitte eine Methode, die prüft, ob alle Namen in einer Sammlung (***Array*** oder ***Set***) den Präfix ***"Ruby"*** haben!
- **Ü-7-b-4:** Schreiben Sie bitte eine Methode, die prüft, ob die Summe aus Schlüssel und Wert jedes Paares in einem ***Hash*** durch 3 teilbar ist!
- **Ü-7-b-5:** Schreiben Sie bitte eine Methode, die prüft, ob für alle Schlüssel eines Hashes, der Zahlen auf Zahlen abbildet, gilt: wenn ***key_x < key_y*** dann ***val_x < val_y***! **Hinweis**: Sie müssen jedes Schlüssel-Wert Paar mit jedem Schlüssel-Wert Paar vergleichen.
- **Ü-7-b-6:** Schreiben Sie bitte in der Klasse ***Telefonbuch*** eine Methode, die prüft, ob eine Sammlung von Präfixen, die der Methode als Parameter übergeben werden, in den Schlüsseln des Telefonbuchs enthalten ist!

Übungen

- **Ü-7-b-7:** Schreiben Sie bitte eine Methode, die prüft, ob eine Sammlung ganzer Zahlen eine Kubikzahl enthält!
- **Ü-7-b-8:** Schreiben Sie bitte eine Methode, die prüft, ob eine Sammlung die Zeichenkette ***"end"*** enthält! Verwenden Sie bitte die ***any?*** Methode!
- **Ü-7-b-9:** Schreiben Sie bitte eine Methode, die prüft, ob alle Elemente der Sammlung Vielfache eines Elementes der Sammlung sind! x ist Vielfaches von sich selber.
- **Ü-7-b-10:** Schreiben Sie bitte eine Methode, die für ein 2-dim Array prüft, ob alle Elemente die gleiche Länge haben.!
- **Ü-7-b-11:** Schreiben Sie bitte eine Methode, die für ein 3-dim Array prüft, ob alle Elemente die gleiche Länge haben! **Hinweis**: Sie müssen die enthaltenen 2-dim und 1-dim Arrays prüfen.
- **Ü-7-b-12:** Schreiben Sie bitte eine Methode, die prüft, ob es in einem Hash ein Schlüssel-Wert Paar gibt, deren Inhalte gleich sind! (***key==value***)
- **Ü-7-b-13:** Schreiben Sie bitte eine Methode, die prüft, ob es ein Element in einer Sammlung gibt, das größer als alle anderen Elemente der Sammlung ist! **Hinweis**: Sie müssen jedes Element mit jedem Element vergleichen.
- **Ü-7-b-14:** Schreiben Sie bitte eine Methode, die prüft, ob es in einem Hash ein Schlüssel-Wert gibt, dessen Wert kleiner aller Werte der sonstigen Schlüssel-Wert Paare ist! **Hinweis**: Sie müssen jedes Schlüssel-Wert Paar mit jedem Schlüssel-Wert Paar vergleichen.

Abbildungen mit *collect* und *map*

sam.collect {|elem| block}

- ***collect/map*** sammelt das Ergebnis der Auswertung des Blocks für jedes Element in ***sam*** in einem Array und gibt dieses Array als Ergebnis zurück.
- Die Kardinalität des Ergebnis-Arrays ist gleich der Kardinalität von ***sam***.
- ***collect/map*** entspricht also einer Abbildung von ***sam*** auf das Ergebnisarray. Die Abbildungsvorschrift wird in ***block*** angegeben.

Beispiele

- Quadratzahlen zu den Elementen einer Sammlung berechnen. $f(x) = x^2$

```
sam = [1.2, 3,4,5, 89, 3.6]
sam.collect {|x| x**2 }
```

- 4-elementige Präfixe der Elemente einer Sammlung berechnen. f(x) = x[0,4]

```
sam = ["Wotan" , "Odin","Thor",
"Frija", "Gautaz"]
sam.map {|elem| elem[0,4]}
```

- Die Schlüssel eines Hashes berechnen:

```
sam = {"Anna" => 12, "Alfred" =>
2, "Antilope" => 78}
p sam.map{|k,v| k }
```

Abbildungen mit *collect* und *map*

- Quadratzahlen zu den Elementen einer Sammlung berechnen. f(x) = x^2

```
sam = [1.2, 3,4,5, 89, 3.6]
sam.collect {|x| x**2 }
```

```
[1.44, 9, 16, 25, 7921, 12.96]
```

- 4-elementige Präfixe der Elemente einer Sammlung berechnen. f(x) = x[0,4]

```
sam = ["Wotan" , "Odin","Thor",
"Frija", "Gautaz"]
sam.map {|elem| elem[0,4]}
```

```
["Wota", "Odin", "Thor", "Frij", "Gaut"]
```

- Die Schlüssel eines Hashes berechnen:

```
sam = {"Anna" => 12, "Alfred" =>
2, "Antilope" => 78}
p sam.map{|k,v| k }
```

```
["Anna", "Alfred", "Antilope"]
```

Übungen

- **Ü-7-b-15:** Berechnen Sie bitte mit ***map*** zu einer Sammlung die ***sin*** Funktion!
- **Ü-7-b-16:** Berechnen Sie bitte mit ***map*** zu einer Sammlung die ***sin*** Funktion und geben Sie als Ergebnis ein Array mit 2-elementigen Arrays zurück, die an erster Stelle den ***x*** Wert und an zweiter Stelle den ***sin(x)*** Wert enthalten!
- **Ü-7-b-17:** Berechnen Sie bitte mit ***map*** für eine Sammlung die Funktion ***Math.sqrt***!
- **Ü-7-b-18:** Berechnen Sie bitte mit ***map*** für eine Sammlung von Zeichenketten die Dateiendungen! Geben Sie bitte das Ergebnis auf der Konsole aus!
- **Ü-7-b-19:** Bilden Sie bitte mit ***collect*** eine Sammlung von Arrays auf eine Sammlung von Strings ab, indem Sie auf den einzelnen Arrays die Methode ***to_s()*** aufrufen!
- **Ü-7-b-20:** Berechnen Sie bitte für einen Hash die Sammlung der Werte!
- **Ü-7-b-21:** Berechnen Sie bitte aus einem Hash die vertauschten Schlüssel-Wert Paare als Array von 2-elementigen Arrays! Die Werte stehen an der ersten die Schlüssel an der 2‘ten Position der 2‘elementigen Arrays. Verwenden Sie bitte die Methode ***map*** oder ***collect***!

max / min ohne Block

sam.max (***min*** analog)

- Liefert das maximale / minimale Element einer Sammlung ***sam***. Die Elemente der Sammlung müssen den allgemeinen Vergleichsoperator (***<=>***) verstehen.
- **Hashes**: Das Maximum bzw. Minimum wird über die 2-elementigen Arrays aus Schlüssel-Wert Paaren gebildet.

Beispiele:

- Maximum einer Sammlung von Zahlen

```
sam = [1,8,7611,3,5]
p sam.max()
```

- Minimum einer Menge von Zeichenketten

```
sam = Set.new(["Anna", "Alfred",
"Antilope"])
p sam.min()
```

- Minimum eines Hashes.

```
sam = {"Anna" => 12, "Alfred" =>
25, "Antilope" => 7}
p sam.min()
```

max / min ohne Block

Beispiele:

- Maximum einer Sammlung von Zahlen

```
sam = [1,8,7611,3,5]
p sam.max()
```

```
7611
```

- Minimum einer Menge von Zeichenketten

```
sam = Set.new(["Anna",
"Alfred", "Antilope"])
p sam.min()
```

```
"Alfred"
```

- Minimum eines Hashes.

```
sam = {"Anna" => 12, "Alfred"
=> 25, "Antilope" => 7}
p sam.min()
```

```
["Alfred", 25]
```

max / min mit Block

sam.min {\|elem\| block} (analog ***max***)

- Liefert das Element einer Sammlung ***sam***, das bzgl. eines Vergleichskriteriums minimal bzw. maximal ist. Das Vergleichskriterium muss im ***block*** mit dem allgemeinen Vergleichsoperator (***<=>***) ausgedrückt werden.
- Die Blockvariablen ***x,y*** enthalten in jedem Schritt die Werte von zwei benachbarten Elementen
- **Hashes**: Über die Blockvariablen ***x,y*** -- je ein 2-elementiges Array für ein Schlüssel-Wert Paar -- können die Schlüssel und Werte separat adressiert werden.

Beispiele:

- Kürzestes enthaltenes Array:

```
sam = [[9], [4,5], [6,7]]
sam.min{|x,y|
        x.length <=> y.length}
```

- Die kürzeste Zeichenkette in einem Set:

```
sam = Set.new(["Anna", "Alfred",
"Antilope"])
sam.min {|x,y|
      x.length <=> y.length()}
```

- Eintrag mit dem kleinsten Wert in einem Hash:

```
sam = {"Anna" => 12, "Alfred" =>
25, "Antilope" => 7}
p sam.min {|x,y| x[1] <=> y[1]}
```

max / min mit Block

Beispiele:

- Kürzestes enthaltenes Array:

```
sam = [[9], [4,5], [6,7]]
sam.min{|x,y|
       x.length <=> y.length}
```

```
[9]
```

- Die kürzeste Zeichenkette in einem Set:

```
sam = Set.new(["Anna", "Alfred",
"Antilope"])
sam.min {|x,y|
     x.length <=> y.length()}
```

```
"Anna"
```

- Eintrag mit dem kleinsten Wert in einem Hash:

```
sam = {"Anna" => 12, "Alfred" =>
25, "Antilope" => 7}
p sam.min {|x,y| x[1] <=> y[1]}
```

```
["Antilope", 7]
```

Beispiele:

Übungen

- **Ü-7-b-22:** Schreiben Sie bitte eine Methode die aus einem Satz (= Zeichenkette aus Wörtern) das größte Wort berechnet! Zerlegen Sie den Satz zuvor mit der Methode ***split***! Verwenden Sie ***max***!

- **Ü-7-b-23:** Berechnen Sie bitte für einen Hash, dessen Werte Arrays sind, das Element, dessen Wert die maximale Länge hat! Verwenden Sie ***max***!
- **Ü-7-b-24:** Berechnen Sie bitte für einen Hash, dessen Werte Arrays von Zahlen sind, das Element, für das Summe der in den Arrays enthaltenen Zahlen maximal wird! Verwenden Sie ***max*** und ***inject***!

Sortieren von Objektsammlungen nach Eigenschaften

sam.sort_by{|x| block}

- ***sort_by*** sortiert eine Objektsammlung nach der Eigenschaft von ***x***, die im ***block*** genannt wird.
- Das Ergebnis von ***sort_by*** ist ein Array.
- **Hashes**: ***x*** ist ein 2-elementiges Array aus Schlüssel und Wert.
- Sortieren eines Hashes nach Länge der Elemente ist **Unsinn**, da alle 2-elem Array gleiche Länge haben.

Beispiele

- Sortieren einer Sammlung von Zeichenketten nach Wortlänge:

```
sam = ["Antilope", "Alfred", "Anna"
]
p sam.sort_by {|x| x.length()}
```

- Sortieren einer Menge von Zeichenketten nach Dateiendung:

```
sam = Set.new([ "sort_by.docx", …
])
p sam.sort_by {|x|
   i = x.index(".")+1
   x[(i..x.length()-1)]
}
```

- Sortieren eines Hashes nach Wortlänge des Schlüssels.

```
sam = {"Anna" => 12, "Alfred" =>
25, "Antilope" => 7}
p sam.sort_by {|x| x[0].length() }
```

Sortieren von Objektsammlungen nach Eigenschaften

- Sortieren einer Sammlung von Zeichenketten nach Wortlänge:

```
sam = ["Antilope", "Alfred",
"Anna" ]
p sam.sort_by {|x| x.length()}
```

```
["Anna", "Alfred", "Antilope"]
```

- Sortieren einer Menge von Zeichenketten nach Dateiendung:

```
sam = Set.new([ "sort_by.docx",
      … ])
p sam.sort_by {|x|
   i = x.index(".")+1
   x[(i..x.length()-1)]
}
```

```
["min.dll", "sort_by.docx", "max.java",
"min.java"]
```

- Sortieren eines Hashes nach Wortlänge des Schlüssels:

```
sam = {"Anna" => 12, "Alfred" =>
25, "Antilope" => 7}
p sam.sort_by {|x| x[0].length()
}
```

```
[["Anna", 12], ["Alfred", 25],["Antilope", 7]]
```

Sortieren von Sammlungen durch impliziten Objektvergleich

sam.sort ()

- sortiert eine Objektsammlung ***sam*** in aufsteigender Reihenfolge, indem die enthaltenen Elemente paarweise mit ***<=>*** verglichen werden.
- Wenn ***x, y*** Elemente der Sammlung sind und ***x<=>y<0***, dann steht ***x*** in der sortierten Sammlung vor ***y***.
- Das Ergebnis einer Sortierung ist immer ein Array.
- **Hashes**: Es werden die 2-elementigen Arrays der Schlüssel-Wert Paare verglichen. Ergebnis: Array mit 2-elementigen Arrays.
- **Anforderung an die Elemente**: Müssen vergleichbar sein (***Comparable*** inkludieren) .

Beispiele

- Sortieren einer Sammlung von Zahlen:

```
sam = [7611,3,8,1,5]
p sam.sort()
```

- Sortieren einer Sammlung von Zeichenketten:

```
sam = [ "bz77zzzzz",  "abaaaa",
"Bz78xxx", "bz78", "abc"]
p sam.sort()
```

- Sortieren eines Hashes:

```
sam = {"Anna" => 12, "Alfred"
=> 25, "Antilope" => 7}
p sam.sort()
```

Sortieren von Sammlungen durch impliziten Objektvergleich

- Sortieren einer Sammlung von Zahlen:

```
sam = [7611,3,8,1,5]
p sam.sort()
```

```
[1, 3, 5, 8, 7611]
```

- Sortieren einer Sammlung von Zeichenketten:

```
sam = [ "bz77zzzzz",
"abaaaa", "Bz78xxx", "bz78",
"abc"]
p sam.sort()
```

```
["Bz78xxx", "abaaaa", "abc", "bz77zzzzz", "bz78"]
```

- Sortieren eines Hashes:

```
sam = {"Anna" => 12,
"Alfred" => 25, "Antilope" =>
7}
p sam.sort()
```

```
[["Alfred", 25], ["Anna", 12], ["Antilope", 7]]
```

Sortieren von Sammlungen durch expliziten Objektvergleich

sam.sort {|x,y| block}

- sortiert eine Objektsammlung ***sam.*** Das Sortierkriterium ergibt sich indem auf die enthaltenen Elemente paarweise der ***block*** angewendet wird.
- ***block*** muss einen Vergleich über den allgemeinen Vergleichsoperator **<=>** definieren.
- Die Blockvariablen ***x, y*** enthalten die Elemente der Sammlung, die miteinander verglichen werden.
- **Hashes**: ***x*** und ***y*** sind 2-elementige Arrays der Schlüssel-Wert Paare.
- **Anforderung an die Elemente**: Müssen die die Methode **<=>** implementieren.

Beispiele

- Absteigendes Sortieren einer Sammlung:

```
sam = [7611,3,8,1,5]
p sam.sort {|x,y| y <=> x }
```

```
sam = [ "bz77zzzzz",  "abaaaa",
"Bz78xxx", "bz78", "abc"]
p sam.sort {|x,y| y <=> x }
```

- Absteigendes Sortieren eines Hashes nach Werten:

```
sam = {"Anna" => 12, "Alfred" =>
25, "Antilope" => 7}
p sam.sort{|x,y| y[1] <=> x[1]}
```

Sortieren von Sammlungen durch expliziten Objektvergleich

- Absteigendes Sortieren einer Sammlung:

```
sam = [7611,3,8,1,5]
p sam.sort {|x,y| y <=> x }
```

```
[7611, 8, 5, 3, 1]
```

```
sam = [ "bz77zzzzz",
"abaaaa", "Bz78xxx", "bz78",
"abc"]
p sam.sort {|x,y| y <=> x }
```

```
["bz78", "bz77zzzzz", "abc", "abaaaa", "Bz78xxx"]
```

- Absteigendes Sortieren eines Hashes nach Werten:

```
sam = {"Anna" => 12, "Alfred"
=> 25, "Antilope" => 7}
p sam.sort{|x,y| y[1] <=>
x[1]}
```

```
[["Alfred", 25], ["Anna", 12], ["Antilope", 7]]
```

Übungen

- **Ü-7-b-25:** Sortieren Sie bitte einen Hash nach Schlüsseln, je einmal aufsteigend und absteigend!
- **Ü-7-b-26:** Sortieren Sie bitte einen Hash nach Werten, je einmal aufsteigend und absteigend!

- **Ü-7-b-27:** Sortieren Sie bitte eine Sammlung von Dateinamen aufsteigend nach dem Dateinamen ohne Dateiendung!

- **Ü-7-b-28:** Schreiben Sie bitte Methoden für die Klasse ***Telefonbuch***, die ein Telefonbuch nach Namen und nach Nummern sortieren!

Zusammenfassung

- Iteratoren gibt es in Ruby für Zahlen, Zeichenketten, Intervalle, Objektsammlungen und Hashes.
- Von Ruby 1.8 zu 1.9 wurde der Basisiterator für Zeichenketten entfernt.
- **Objektvertrag**: Ein Vertrag zwischen Objekten regelt, welche Teilfunktionalität die einzelnen Klassen zur Erreichung einer Gesamtfunktionalität beitragen müssen.
- Der Vertrag zwischen ***Enumerable*** und anderen Klassen beinhaltet, dass andere Klassen den **Basisiterator** ***each*** implementieren müssen, damit sie die **Iteratormethoden** von ***Enumerable*** nutzen können. Der Beitrag von ***Enumerable*** ist die Implementierung dieser Iteratormethoden.
- ***Enumerable*** führt die Implementierung seiner Iteratormethoden auf den Basisiterator ***each*** zurück, ohne ***each*** selber zu implementieren.

Ausblick

- Wir wollen im Folgenden selber Vertragspartner von ***Enumerable*** werden und in der Klasse ***Telefonbuch*** den Basisiterator ***each*** implementieren.

- Wir werden das ***each*** von Telefonbuch an das ***each*** des internen Hashes delegieren.

- Da ***each*** einen Block und Blockvariablen benötigt, um sinnvoll zu arbeiten, müssen wir uns zunächst damit auseinandersetzen, wie wir Blöcke in der Methodendefinition ansprechen, mit ihren Variablen ausführen und beim Aufruf anderer Methoden übergeben können.