

Die Datenstruktur Liste in Haskell¹- kleine aber feine Kurz-Übersicht

Grundlagen der Datenstruktur Liste

- Aufbewahren von Daten **gleichen** Typs
- Einzelne Elemente können komplex sein
- Im Unterschied zu mathematischen Mengen sind doppelte Elemente erlaubt
- Schreibweise: [2,4,3,4]
- Punktschreibweise ist möglich: [3..12] oder [1,3..99]
- Auch unendliche Mengen sind definiert: [1..] (Menge der natürlichen Zahlen)
Macht aber nur Sinn unter Verwendung anderer Funktionen (sonst Endlosschleife!)

Grundstruktur der Datenstruktur Liste

- Eine Liste ist entweder leer
[] (leere Menge)
oder enthält ein erstes **Element** und eine **Restliste**
Kopf:Rest (mind. 1 Element)
Kopf - erstes Element der Liste
Rest - Restliste (kann auch leer sein!)
: - trennt den Kopf vom Rest
Achtung: Kopf und Rest haben unterschiedlichen Typ!
Kopf ist ein **Element** (z.B. Int)
Rest ist eine **Liste** (z.B. [Int])

Nach dem Gleichheitszeichen (**Definitionsbereich**):

- Zwei Listen werden mit ++ verkettet
- Mit : wird ein Element vor eine Liste gesetzt (Listenkonstruktion)
- es hat sich die Schreibweise x:xs durchgesetzt (**Notation**) – auch links vom Gleichheitszeichen zur Verwendung des Pattern-Matching(sic!;-). Vgl. cons-Operator an anderer Stelle.

Basisfunktionen

Haskell kennt eine Reihe von vordefinierten Funktionen auf Listen, hier wird die Liste mit xs bezeichnet:

- tail xs -- Rest einer Liste
- head xs -- Kopf einer Liste
- init xs -- Anfang einer Liste
- last xs -- letztes Element
- reverse xs -- Liste umkehren
- length xs -- Länge einer Liste
- xs!n -- n-tes Element

Listcomprehension (List(en)generator)

Besonders bequem ist das Erzeugen von Listen in einer beschreibenden Form („Listcomprehension“):

```
[ x | x <- [1..100] , div x 5 > 4 , mod x 7 == 0 ] – was erzeugt dieser List-Generator?
```

Lies:

```
x <- [1..]           -- Generator-Teil
  div x 5 > 4       -- 1. Filter (Bedingung)
  mod x 7 == 0     -- 2. Filter (Bedingung)
```

- alle Elemente, die **beide** Filter überwinden, landen in der Ergebnis-**Liste**.
- die Filter werden durch ein **Komma** getrennt. Das Komma bedeutet eine logische UND-Verknüpfung der Filter-Bedingungen.

Ergebnis: [56,63,70,77,84,91,98]

¹ mit Material von W.Gussmann

High-Order-Funktionen – Funktionen als Parameter

Die map-Funktion

Häufig muss mit jedem Element einer Liste etwas **gemacht** werden. Dafür gibt es die (interne) Funktion `map`. Diese ist in Haskell intern vordefiniert und kann verwendet werden.

Beispiel:

Problemstellung: *Bestimme alle Wurzeln von 1 bis 10*

```
map sqrt [1..10] - beispielhafter Aufruf der internen Funktion map mit
                  der (internen) Funktion sqrt als Parameter!
```

Problemstellung: *Wir wollen eine eigene Funktion `map1` definieren*

Lösung 1: Mit „Listcomprehension“ - Listengenerator

```
map1 f xs = [f x | x<-xs] - wendet auf die gegebene Liste xs die
                          Funktion f an
```

Die Funktion `f` muss passend gewählt werden, damit die Typen stimmen

```
f :: a -> b, und
map1 :: (a -> b) -> [a] -> [b]
```

Als Beispielaufruf können wir natürlich auch unserer selbst definierte Funktion `quadrat` verwenden, um alle Quadratzahlen von 1 bis 10 in einer Liste anzuzeigen:

```
map1 quadrat [1..10]
```

map2 rekursiv

Weiteres Beispiel:

Problemstellung wie oben: *Bestimme alle Wurzeln von 1 bis 10*

```
map2 sqrt [1..10]
```

Lösung 2: „Rekursion“

```
map2 f [] = []
map2 f (x:xs) = (f x) : map2 f xs      -- vgl. CONS-Operator
```

Die filter-Funktion

Eine **Teilliste** erreicht man mit der `filter`-Funktion:

```
filter :: (a -> Bool) -> [a] -> [a]
```

Die Filterfunktion wendet die Boolesche-Funktion auf alle Elemente an. In die **Ergebnismenge** kommen nur Elemente, die `True` liefern.

Beispiel: *Filtere aus einer gegebenen Liste alle Primzahlen*

```
filter prim [1,2,4,5,7,6,8,9,3]
prim :: Int -> Bool
prim n = [x|x<-[2..n], mod x n == 0]==[n]
```

oder

```
prim n = filter (teilbar n) [2..n] == [n]
teilbar b :: Int -> Bool
teilbar b c = mod b c == 0
```