Tutorium 03: Typen und Typklassen

Paul Brinkmeier

24. November 2020

Tutorium Programmierparadigmen am KIT

Heutiges Programm

Programm

- Übungsblätter 1 und 2
- Wiederholung der Vorlesung: Typen und Typklassen
- Datentypen selbst definieren

Übungsblatt 1

Aufgaben 2 und 3

Todo

Übungsblatt 2

Todo

Wiederholung: Typen

Cheatsheet: Listenkombinatoren

- `foldr :: (a -> b -> b) -> b -> [a] -> b`
- `foldl :: (b -> a -> b) -> b -> [a] -> b`
- `map :: (a -> b) -> [a] -> [b]`
- `filter :: (a -> Bool) -> [a] -> [a]`
- `zipWith :: (a -> b -> c) -> [a] -> [b] -> [c]`
- `zip :: [a] -> [b] -> [(a, b)]`
- `and, or :: [Bool] -> Bool`

Idee: Statt Rekursion selbst zu formulieren verwenden wir fertige „Bausteine“, sogenannte „Kombinatoren“.

Cheatsheet: Tupel und Konzepte

- List comprehension, Laziness
- `[f x | x <- xs, p x]` $\equiv$ `map f (filter p xs)`
  Bspw.: `[x * x | x <- [1..]]` $\Rightarrow$ `[1,4,9,16,25,...]`
- Tupel
- `(,) :: a -> b -> (a, b)` („Tupel-Konstruktor“)
- `fst :: (a, b) -> a`
- `snd :: (a, b) -> b`

Cheatsheet: Typen

- `Char`, `Int`, `Integer`, ...
- `String`
- Typvariablen/Polymorphe Typen:
  - `(a, b)`: Tupel
  - `[a]`: Listen
  - `a -> b`: Funktionen
  - Vgl. Java: `List<A>`, `Function<A, B>`
- Typsynonyme: `type String = [Char]`

Wiederholung: Typklassen

Typen selbst definieren

Typklassen definieren und implementieren