

# Übungen zu Funktionaler Programmierung

## Übungsblatt 5

Ausgabe: 11.11.2016, Abgabe: 18.11.2016

**Aufgabe 5.1** (4 Punkte) Lösen Sie die folgenden Aufgaben mithilfe der Funktion **map** oder **zipWith** (Funktionslifting).

1. **double** :: [Int] -> [Int] multipliziert alle Ganzzahlen in einer Liste mit zwei.  
Beispiel: `double [1,2,3] ~> [2,4,6]`
2. **maxima** :: [Int] -> [Int] -> [Int] vergleicht die Werte aus zwei Listen und gibt eine Liste mit den Maxima zurück.  
Beispiel: `maxima [10,3,5,23] [1,14,6,10] ~> [10,14,6,23]`
3. **funs** :: Int -> [Int] erzeugt eine Liste mit drei Werten. Dem Nachfolger der Eingabe, dem Doppeltem der Eingabe und die Eingabe hoch 2.  
Beispiel: `funs 3 ~> [4,6,9]`
4. **toUnicode** :: String -> [Int] wandelt einen String in eine Liste der Unicode-Codierungen der einzelnen Zeichen um. Mit der Funktion `fromEnum` kann ein Zeichen (Char) in seine Codierung gewandelt werden.  
Beispiel: `toUnicode "%hello!" ~> [37,104,101,108,108,111,33]`

### Lösungsvorschlag

```
double :: [Int] -> [Int]
double = map (*2)

maxima :: [Int] -> [Int] -> [Int]
maxima = zipWith (\x y -> if x < y then y else x)
-- alternativ:
-- maxima = zipWith max

funs :: Int -> [Int]
funs i = map ($ i) [(+1), (*2), (^2)]

toUnicode :: String -> [Int]
toUnicode = map fromEnum
```

**Aufgabe 5.2** (4 Punkte) Werten Sie folgende Haskell-Ausdrücke schrittweise aus.

1. `foldl (-) 5 [1, 3]`
2. `foldr (-) 5 [1, 3]`

### Lösungsvorschlag

```

-- foldl (-) 5 [1, 3] = (5-1)-3
foldl (-) 5 [1, 3]
~> foldl (-) ((-) 5 1) [3]
~> foldl (-) 4 [3]
~> foldl (-) ((-) 4 3) []
~> foldl (-) 1 []
~> 1

-- foldr (-) 5 [1, 3] = (1-(3-5))
foldr (-) 5 [1, 3]
~> foldr (-) 5 [1, 3]
~> (-) 1 $ foldr (-) 5 [3]
~> (-) 1 $ (-) 3 $ foldr (-) 5 []
~> (-) 1 $ (-) 3 $ 5
~> (-) 1 ((-) 3 5)
~> (-) 1 (-2)
~> 3

```

**Aufgabe 5.3** (4 Punkte) Definieren Sie folgende Funktionen mithilfe der Listenkompensation.

1. `inBoth :: [Int] -> [Int] -> [Int]` gibt nur die Werte aus der ersten Liste aus, die auch in der zweiten Liste vorkommen.  
Beispiel: `inBoth [1,2,3,4] [1,4,5,3,4] ~> [1,3,4]`
2. `map2 :: (a -> b -> c) -> [a] -> [b] -> [c]` wendet einen binären Operator auf jeder Kombination von Werten aus zwei Eingabelisten.  
Beispiel: `map2 (+) [1,2] [10,20,30] ~> [11,21,31,12,22,32]`
3. `divisors :: Int -> [Int]` gibt eine Liste aller Teiler des Eingabewertes.  
Beispiel: `divisors 12 ~> [1,2,3,4,6,12]`
4. `solutions :: [(Int, Int, Int)]` enthält Tripel  $(x,y,z) \in \mathbb{N} \times \mathbb{N} \times \mathbb{N}$ , welche die Gleichung  $3x^2 + 2y + 1 = z$  lösen. Nehmen Sie für  $x, y$  und  $z$  nur Werte von 0 bis 100.

### Lösungsvorschlag

```

inBoth :: [Int] -> [Int] -> [Int]
inBoth xs ys = [ x | x <- xs, x `elem` ys]

map2 :: (a -> b -> c) -> [a] -> [b] -> [c]
map2 f xs ys = [f x y | x <- xs, y <- ys]

divisors :: Int -> [Int]
divisors n = [ m | m <- [1..n], n `mod` m == 0 ]

solutions :: [(Int, Int, Int)]
solutions = [ (x,y,z)
  | z <- [0..100]
  , y <- [0..100]
  , x <- [0..100]
  , 3*x^2 + 2*y + 1 == z
  ]

```