Übungsaufgaben 1

Ziel

Ziel der ersten Übungsaufgabe ist es, sich mit der grundlegenden Syntax von Haskell vertraut zu machen. Darüber hinaus soll der Umgang mit Rekursion und unendlich langen Listen geübt werden.

Alle Lösungen zu diesen Aufgaben lassen sich leicht durch Recherche auf Stackoverflow finden, dies ist aber nicht der Übung. Es geht darum zu üben Probleme funktional zu lösen.

Eigene Listenfunktionen

Nimm an, Dir stehen nur folgende Syntaxelemente zu Verfügung:

- Zerlegung einer Liste mit dem ersten Element im Parameter: f (x:xs) = …
- if then else
- == Vergleichsoperator
- [a] zum Erzeugen einer Liste mit einem Wert
- [] Leere Liste
- [a] ++ [a] zur Verkettung von Listen

Definierte auf dieser Grundlage die Funktionen myhead, mytail, mylast, myinit und myreverse. Die Funktionen dürfen sich gegenseitig verwenden.

FizzBuzz

Schreibe eine Funktion, die das FizzBuzz Wortspiel implementiert. Die Funktion fizzBuzz x gibt eine Liste von String zurück. Diese enthalten die Zahlen in der Range 1..x, wobei Zahlen die durch 3 teilbar sind durch die Zeichenkette „Fizz“ ersetzt werden. Zahlen die durch 5 teilbar sind werden durch „Buzz“ ersetzt und Zahlen die sowohl durch 3 als auch durch 5 teilbar sind werden durch „FizzBuzz“ ersetzt.

Verwende hierzu List Comprehension , Guards, mod und ==.

Der Typ der Funktion ist: fizzBuzz :: (Integral a, Show a) => a -> [String]

Fibonacci-Folge

Die Fibonacci-Folge ist definiert durch die Startzahlen 0 und 1. Alle weiteren Elemente sind jeweils die Summe der zwei vorherigen Zahlen, also: 0, 1, 1, 2, 3, 5, 8, 13, ….

Schreibe eine Definition für *fibonacci*, die eine unendlich lange List der Fibonacci-Reihe darstellt. Definiere hierfür eine rekursive Funktion ohne Abbruchbedingung. In der Funktion sind folgende Operatoren erlaubt:

- [a] ++ [a] zur Verkettung von Listen
- [a] zum Erzeugen einer Liste mit einem Wert
- (+) zum addieren zweier Zahlen

Der Typ der Funktion ist: fibonacci :: Num a => [a]

Anschließend sollen die normalen Listenfunktionen auf die Rückgabe angewendet werden können.

> take 7 fibonacci

[0,1,1,2,3,5,8]

> fibonacci !! 7

13

Primzahlen

Schreibe eine Definition von *primes*, die über List Comprehension eine unendlich lange Liste von Primzahlen erzeugt. Erstelle hierfür ggf. eine Hilfsfunktion *isPrime*, die eine übergebene Zahl daraufhin prüft ob sie eine Primzahl ist.

Am Ende muss folgender Ausdruck funktionieren.

> take 10 primes

[2,3,5,7,11,13,17,19,23,29]

Primfaktorenzerlegung

Schreibe die Funktion *primeFactors*, die eine übergebene Zahl in ihre Primzahlenfaktoren zerlegt und diese als Liste zurückgibt. Bediene dich hierfür der Funktionen *isPrime* und der Definition *primes*. Die Funktion kann sehr einfach mittels einer Guard und einer Hilfsfunktion in einer where Klausel definiert werden.

Der Typ der Funktion ist: primeFactors :: Integral a => a -> [a]

Am Ende soll folgender Ausdruck möglich sein

> [primeFactors x| x <- [1..10]]

[[],[2],[3],[2,2],[5],[2,3],[7],[2,2,2],[3,3],[2,5]]