Dieses Aufgabenblatt ist ausdrücklich **NICHT** auf Klausurniveau.

Es beinhaltet Lernaufgaben, um die grundlegenden Haskell-Inhalte besser zu verstehen.

Alle Aufgaben beziehen sich auf Haskell und sollen in *.hs-Dateien angelegt werden. Diese werden dann in den Interpreter `ghci` geladen und überprüft.

1 Variablendeklaration

Aufgabe 1

Deklariere eine Variable `a` vom Typ `Bool` mit dem Wert `False`.

Aufgabe 2

Erstelle zwei Variablen `b` und `c` vom Typ `Integer` und `String` mit den Werten `5` und "`Sieben`".

2 Funktionsdefinition

Aufgabe 3

Definiere eine Funktion `multiB` vom Typ `Integer -> Integer`, welche einen eingegeben Integer mit der Zahl b (Aufgabe 2) multipliziert.

Bsp: $multiB(3) = 3 \cdot 5 = 15$

Aufgabe 4

Schreibe eine Funktion `zweier` vom Typ `Integer -> Integer`, welche für einen eingegebenen Exponenten ≥ 0 die Zweierpotenz bestimmt.

Bsp: $zweier(3) = 2^3 = 8$

Aufgabe 5

Gib eine Funktion `istRest` vom Typ `(Integer, Integer, Integer) -> Bool` an, welche überprüft, ob der Rest der Division des ersten durch den zweite Wert ($\neq 0$) dem dritten Wert entspricht.
Bsp:

`istRest(5,4,1)==True,`

da der Rest der Division 5 durch 4 $==$ 1

`istRest(3,0,5)==False,`

da Division durch 0 nicht möglich und somit der Rest nicht bestimmbar.

3 Currying

Aufgabe 6

Wandle die Funktion `hoch` so um, dass sie gecurried ist, wobei

```
hoch :: (Integer, Integer) -> Integer
hoch (x, y) = if y < 0 then -1
                else if y == 0 then 1
                      else x * hoch (x, y - 1)
```

4 Pattern Matching und Guards

Aufgabe 7

Schreibe mit Hilfe von Pattern Matching die Funktionen `head :: [a] -> a` und `tail :: [a] -> [a]` neu. Bei leeren Listen liefere `undefined`.

Aufgabe 8

Gegeben sei die Funktion

```
mySum :: [Integer] -> Integer
mySum l = if null l then 0 else head l + mySum (tail l)
```

a) Definiere mySum mit Hilfe von Pattern Matching möglichst einfach.

b) Definiere mySum mit Hilfe von Guards.

5 Rekursion

Aufgabe 9

Schreibe eine Funkion `entferne :: String -> Char -> String`, welche den Charakter in einem String entfernt (HINWEIS: String entspricht [Char])
Bsp.:

```
entferne 'e' "Mein Text"  ==  "Min Txt"
```

Aufgabe 10

Gib eine Funktion `aufteilen::[Integer] -> ([Integer], [Integer])` an, welche eine Liste von Zahlen in das Tupel (gerade,ungerade) teilt.
Bsp:

```
aufteilen [2,7,4,2,7,2,4,8,8,4,23]  ==  ([2,4,2,2,4,8,8,4], [7,7,23])
```

6 (rekursive) Datenstrukturen

Aufgabe 11

Gegeben sei die folgende Datenstruktur

```
data Bahnhof = Bahnhof Int String                          deriving (Eq, Ord, Show)
data Strecke = Strecke Int Bahnhof Bahnhof [Bahnhof]    deriving (Eq, Ord, Show)
type Schienennetz = [Strecke]
```

Hierbei hat ein Bahnhof immer eine eindeutige, dreistellige ID und den zugehörigen Ort.

Eine Eisenbahnstrecke besteht auch aus einer eindeutigen ID, einem Anfangs- und einem Endbahnhof und aus Zwischenstationen. Die Strecken-ID setzt sich aus der aneinandergereihten Start- und Ziel-Bahnhofs-ID zusammen.
Bsp:

```
start = Bahnhof 123 "Kaiserslautern"
ziel = Bahnhof 321 "Saarbruecken"
```

⇒ Eisenbahnstrecke von Kaiserslautern nach Saarbruecken hat die ID 123321

a) Schreibe zwei Selektorfunktionen `getStart, getZiel :: Strecke -> Bahnhof`

b) Definiere die Funktion `sucheStrecke :: Schienennetz -> Bahnhof -> Bahnhof -> Schienennetz`, welche alle Verbindungen vom ersten zum zweiten Bahnhof filtert und ausgibt. Hierbei darf der zweite Bahnhof auch eine Zwischenstation sein.

c) Gib eine Funktion `konsistent :: Schienennetz -> Bool` an, welche für ein Schienennetz alle Eisenbahnstrecken überprüft, ob deren ID mit der zusammengesetzten ID des Start- und Zielbahnhofs übereinstimmt.

Aufgabe 12

Gegeben sei folgende rekursive Datenstruktur

```
data Baum =
    Blatt Int
  | Zweig Baum Baum
  deriving (Eq, Ord, Show)
```

a) Schreibe eine Funktion `summe :: Baum -> Int` ,welche alle Int-Werte der Blätter zusammenrechnet.

b) Gib eine Diskriminatorfunktion `istBlatt :: Baum -> Bool` an, welche für einen eingebenen Baum überprüft, ob er ein Blatt ist.

c) Gib eine Funktion `anzahl :: Baum -> Int` an, welche die Anzahl der Blätter eines Baumes ausgibt.

d) Definiere die Funktion `symm :: Baum -> Bool`, welche überprüft ob der Baum vom Aufbau her symmetrisch ist.

Viel Spaß und Erfolg beim Lösen der Aufgaben.
Marc Dahlem