1 Einfache Funktionen mit Mustervergleichen definieren

1. Implementieren Sie eine Funktion `oder :: Bool -> Bool -> Bool` welche die logische Disjunktion ($\vee$) berechnet:

   $$\bot \vee \bot = \bot \qquad \bot \vee \top = \top \qquad \top \vee \bot = \top \qquad \top \vee \top = \top$$

   Dabei steht $\bot$ für das logische falsch (Haskell: `False`), und $\top$ für wahr (`True`). Es soll zum Beispiel gelten:

   ```
   *U3> oder True True
   True
   ```

2. Definieren Sie eine Funktion `fst3` welche für ein gegebenes Triple die erste Komponente zurückgibt.

3. Definieren Sie eine Funktion `oneOfFour` welche für eine gegebene Zahl $1 \leq n \leq 4$ und ein gegebenes Quadruple, die entsprechende Komponente zurückgibt. Nutzen Sie dabei eine bewachte Gleichung! Dabei soll z.B. gelten:

   ```
   *Main> oneOfFour 1 ('a','b','c','d')
   'a'
   *Main> oneOfFour 3 ('a','b','c','d')
   'c'
   ```

   Geben Sie den Typ dieser Funktion an!

4. Definieren Sie eine Funktion `countNumOccurrences` welches für einen gegebenen Wert und eine List bestimmt, wie oft der Wert in der Liste vorkommt! Dabei soll z.B. gelten:

   ```
   *Main> countNumOccurrences 'a' "Hallo Welt"
   1
   *Main> countNumOccurrences 'l' "Hallo Welt"
   3
   ```

   Geben Sie den Typ dieser Funktion an!

2 Einfache Funktionen definieren

1. Implementieren Sie eine Funktion `ggt :: (Int, Int) -> Int`, welche den größten gemeinsamen Teiler zweier übergebener Zahlen bestimmt:

$$ggt(x,y) = \begin{cases} \text{nicht definiert} & \text{wenn } x = y = 0 \\ y & \text{wenn } x = 0 \\ ggt(y \bmod x, x) & \text{sonst} \end{cases}$$

(de.wikipedia.org/wiki/Euklidischer_Algorithmus#Moderner_euklidischer_Algorithmus)
Beginnen Sie Ihre Implementation wie folgt:

```
1 ggt::(Int, Int) → Int
2 ggt (0,0) = error "Nicht␣definiert"
3 ...
```

Es soll zum Beispiel gelten:

```
*U3> ggt (44,12)
4
```

2. Implementieren Sie eine Funktion `kuerzen :: (Int, Int) -> (Int, Int)`, welche den als Paar gegebenen Bruch gekürzt zurückgibt:

$$\begin{aligned} \text{kuerzen} : \mathbb{N} \times \mathbb{N} &\to \mathbb{N} \times \mathbb{N} \\ (x,y) &\mapsto \left(\frac{x}{g}, \frac{y}{g}\right) \qquad (\text{mit } g = ggt(x,y)) \end{aligned}$$

Es soll zum Beispiel gelten:

```
*U3> kuerzen (44,12)
(11,3)
```

Definieren Sie die Funktion einmal unter Benutzung von `where` und einmal mit `let`!

3 Zahlen und Zahlentypen in Haskell

1. Erläutern Sie den Unterschied zwischen den Typen `Int` und `Integer`! Informationen erhalten Sie z.B. in der Haskell-Dokumentation zur Prelude, Abschnitt „Numeric types“ unter http://hackage.haskell.org/package/base-4.12.0.0/docs/Prelude.html.

   Lassen Sie sich den Wert der Zahl `12345678901234567890` sowohl als `Int` als auch als `Integer` ausgeben. Erläutern und begründen Sie den Unterschied!

2. Definieren Sie die inverse Funktion `bin2int :: [Int] → Int`, welche eine gegeben Zahl im Binärsystem auf ihre Darstellung im Dezimalsystem abbildet! Dabei sollen die Bits in aufsteigender Wertigkeit sortiert werden (d.h., genau andersrum als normalerweise, aber dies macht die Definition der Funktionen viel einfacher!). So soll z.B. gelten:

```
*Main> bin2int [1,1]     -- d.h., 1*2^0 + 1*2^1
3
*Main> bin2int [0,0,1]   -- d.h., 0*2^0 + 0*2^1 + 1*2^2
4
*Main> bin2int [1,0,1]   -- d.h., 1*2^0 + 0*2^1 + 1*2^2
5
*Main> bin2int [0,1,1]   -- d.h., 0*2^0 + 1*2^1 + 1*2^2
6
```

3. Definieren Sie eine Funktion `int2bin :: Int → [Int]`, welche eine gegeben Zahl im Dezimalsystem auf ihre Darstellung im Binärsystem abbildet! Dabei sollen die Bits wieder in aufsteigender Wertigkeit sortiert angegeben werden. So soll z.B. gelten:

```
*Main> int2bin 3
[1,1]   -- d.h., 1*2^0 + 1*2^1
*Main> int2bin 4
[0,0,1] -- d.h., 0*2^0 + 0*2^1 + 1*2^2
*Main> int2bin 5
[1,0,1] -- d.h., 1*2^0 + 0*2^1 + 1*2^2
*Main> int2bin 6
[0,1,1] -- d.h., 0*2^0 + 1*2^1 + 1*2^2
```

4. Definieren Sie eine Funktion `sumBins :: ([Int], [Int]) → [Int]`, welche zwei gegebene Zahlen im Binärsystem auf ihre Summe abbildet! So soll z.B. gelten $5+6=11$:

```
*Main> int2bin 5
[1,0,1]
*Main> int2bin 6
[0,1,1]
*Main> sumBins ([1,0,1], [0,1,1])
[1,1,0,1]
*Main> bin2int [1,1,0,1]
11
```