Der Code sollte korrekt kompilieren!

--2. Übungszettel - Simeon Vasilev, Tomás Proaño, Vasile Popa

-- Aufgabe 1 --

-- a)
```
isDigit :: Char -> Bool
isDigit x =  if x `elem` ['0'..'9']
             then True
             else False
```
-- b)
```
isDigitG :: Char -> Bool
isDigitG x
        | x `elem` ['0'..'9'] = True
        | otherwise           = False
```

-- c)
```
isDigitP :: Char -> Bool
isDigitP '1' = True
isDigitP '2' = True
isDigitP '3' = True
isDigitP '4' = True
isDigitP '5' = True
isDigitP '6' = True
isDigitP '7' = True
isDigitP '8' = True
isDigitP '9' = True
isDigitP '0' = True
isDigitP _ = False
```

-- d)
```
isDigitCase :: Char -> Bool
isDigitCase x = case x of
      '0' -> True
      '1' -> True
      '2' -> True
      '3' -> True
      '4' -> True
      '5' -> True
      '6' -> True
      '7' -> True
      '8' -> True
      '9' -> True
      _ -> False
```

Die 4 Testaufrufe fehlen

8/9

```
-- Aufgabe 2
{-
x≡13 (mod 3)
x≡12 (mod 5)
x≡11

x = 13t + 3

3t + 13 ≡ 13 - t ≡  12 (mod 5)
-}

-- Aufgabe 3
schaltjahr :: Int -> Bool
schaltjahr jahr = if (mod jahr 4 == 0 || mod jahr 400 == 0) then True
      else if mod jahr 100 == 0 then False
           else False

-- zweite Version mit Guards
schaltjahr :: Int -> String
schaltjahr x

      | ((mod x 4) ==0) =
           ("Das Jahr " ++ show x ++ " ist ein Schaltjahr.")
      | ((mod x 100) ==0) =
           ("Das Jahr " ++ show x ++ " ist kein Schaltjahr.")
      | ((mod x 400)    >0) =
           ("Das Jahr " ++ show x ++ " ist kein Schaltjahr.")

-- Aufgabe 4

{-- Ohne Rekursion:
countAsTrolls  :: Int -> String
countAsTrolls  0 = ""
countAsTrolls  1 = "One"
countAsTrolls  2 = "Two"
countAsTrolls  3 = "Three"
countAsTrolls  4 = "Many"
countAsTrolls  5 = "Many-One"
countAsTrolls  6 = "Many-Two"
countAsTrolls  7 = "Many-Three"
countAsTrolls  8 = "Many-Many"
countAsTrolls  9 = "Many-Many-One"
countAsTrolls  10 = "Many-Many-Two"
countAsTrolls  11 = "Many-Many-Three"
countAsTrolls  13 = "Many-Many-Many"
countAsTrolls  14 = "Many-Many-Many-One"
countAsTrolls  15 = "Many-Many-Many-Two"
countAsTrolls  16 = "Many-Many-Many-Three"
countAsTrolls  17 = "LOTS-One"
countAsTrolls  18 = "LOTS-Two"
countAsTrolls  19 = "LOTS-Three"
countAsTrolls  20 = "LOTS-LOTS"
```

Es sollte ein Listengenerator verwendet werden!
Welche Größen sind denn möglich?

Wie viele Möglichkeiten gibt es für (b)?

0/5

Wenn das Jahr durch 4 teilbar ist, dann wird die Ausnahme mit der Teilbarkeit durch 100 nicht beachtet.

2/4

Die Aufgabe sollte rekursiv gelöst werden

?

```
countAsTrolls  21 = "LOTS-LOTS-One"
countAsTrolls  22 = "LOTS-LOTS-Two"
countAsTrolls  23 = "LOTS-LOTS-Three"
countAsTrolls  24 = "LOTS-LOTS-LOTS"
countAsTrolls  25 = "LOTS-LOTS-LOTS-One"
countAsTrolls  26 = "LOTS-LOTS-LOTS-Two"
countAsTrolls  27 = "LOTS-LOTS-LOTS-Three"
countAsTrolls  28 = "LOTS-LOTS-LOTS-LOTS"
countAsTrolls  29 = "LOTS-LOTS-LOTS-LOTS-One"
countAsTrolls  30 = "LOTS-LOTS-LOTS-LOTS-Two"
countAsTrolls  31 = "LOTS-LOTS-LOTS-LOTS-Three"
countAsTrolls  32 = "LOTS-LOTS-LOTS-LOTS-LOTS"
countAsTrolls  33 = "LOTS-LOTS-LOTS-LOTS-LOTS-One"
--}

{-- Anderes Vorgehen Concat mit error:
countAsTrolls :: Int -> String
countAsTrolls x
   | x < 3 = as !! x
   | x > 3 = bs !! as ++ bs
   | otherwise = undefined
   where
      as = "" : ("One Two Three")
      bs = "" : "" : ("Many- Many-Many- Many-Many-Many- LOTS- LOTS-LOTS- LOTS-LOTS-
LOTS-")
--}
```

f. P.

Die Funktion ist auch nicht rekursiv :/

0/6

```
-- Aufgabe 5

-- a)
countOnes :: [Int] -> Int
countOnes xs = length (filter (== 1) xs)

-- b)
count :: Int -> [Int] -> Int
count x =  length . filter (==x)
```

da die Aufgabe "Rekursion und Listen" heißt, würde ich hier eigentlich eine rekursive Funktion erwartet ^^

✓ +3

✓ +3

sehr elegant :D

6/6

```
{- Hier habe ich Lambda-Funktionen und ein Paar Funktionen höherer Ordnung benutzt, da ich
-- dieses Modul früher bestanden habe und diese mir schon bekannt sind
count :: Ord a => a -> [a] -> Integer
count _ [] = 0
count x list = sum $ map(\a -> 1) $ filter (==x) list
-}
```

16/30