WiSe 2020/2021

Funktionale Programmierung

1. Übungsblatt (für das Tutorium)

Prof. Dr. Margarita Esponda

Ziel: Auseinandersetzung mit der Haskell-Syntax, vordefinierten Haskell-Funktionen und ersten einfachen Funktionsdefinitionen.

1. Aufgabe

Schreiben Sie eine Funktion, die ein Zeichen als Argument bekommt und entscheiden kann, ob das Zeichen eine Zahl ist.

Anwendungsbeispiel: **isDigit** '3' => True

Lösung:

```
isDigit :: Char -> Bool
isDigit chr = ('0'<= chr) && (chr <= '9')
```

2. Aufgabe

Schreiben Sie eine Funktion, die bei Eingabe von zwei Zahlen **a** und **b** entscheiden kann, ob **a** Teiler von **b** ist.

a) Benutzen Sie eine ***if-then-else***-Ausdruck.
b) Verwenden Sie ***Guards***.
c) Kann die Funktion mit einer ***case***-Ausdruck programmiert werden?

Lösung a):

```
istTeiler :: Integer -> Integer -> Bool
istTeiler a b = if (a /= 0) then mod b a == 0 else error "not defined for a=0"
```

Lösung b):

```
istTeiler :: Integer -> Integer -> Bool
istTeiler a b | a==0 = error "not defined for a=0"
              | otherwise = mod b a == 0
```

Lösung c):

```
istTeiler :: Integer -> Integer -> Bool
istTeiler a b = case a of 0  -> error "not defined for a=0"
                          otherwise  -> (mod b a) == 0
```

3. Aufgabe

Definieren Sie eine Haskell-Funktion **bogen2winkel**, die bei Eingabe eines Bogenmasses die entsprechenden Winkel in Grad berechnet und definieren Sie die **winkel2bogen** Funktion, die die Umrechnung von Winkel zu Bogenmass macht.

Anwendungsbeispiele: **bogen2winkel** pi => 180.0
winkel2bogen 90 => 1.5707963267948966

Lösung:

```
winkel2bogen :: Double -> Double
winkel2bogen g = g*pi/180

bogen2winkel :: Double -> Double
bogen2winkel r = r*360/(2*pi)
```

4. Aufgabe

Die Berechnung der Großkreis-Entfernung (***gke***) zwischen zwei Orten kann bei Eingabe der geographischen Breiten- und Längengrade (x_1, y_1, x_2, y_2) der Orte mit folgender Formel realisiert werden:

$$gke = c \cdot \arccos(\sin(x_1) \cdot \sin(x_2) + \cos(x_1) \cdot \cos(x_2) \cdot \cos(y_1 - y_2))$$

mit ***c*** *= 111.2225685* = Kilometerabstand zwischen zwei Längengraden

Schreiben Sie eine Haskell-Funktion, die die Großkreis-Entfernung berechnet.
Die Haskell Funktion ***gke*** bekommt die Argumente in Grad. Aber die trigonometrischen Funktionen erwarten Bogenmaß. Deswegen müssen die Argumente vorher umgerechnet werden.

Zum Testen berechnen Sie die Entfernung zwischen Paris und New York.

Anwendungsbeispiel: (Luftlinie zwischen Berlin und Mexiko-Stadt):

gke 52.5192 13.4061 19.4326 -99.1332 => 9729.194434154851 (Kilometer)

Lösung:

```
gke :: Double -> Double -> Double -> Double -> Double
gke  b1  l1  b2  l2 = c * bogen2winkel (acos ((sin x1)*(sin x2) +
                                        ((cos x1)*(cos x2)*(cos (y2-y1)))))
                    where
                      c = 111.2225685
                      x1 = winkel2bogen b1
                      x2 = winkel2bogen b2
                      y1 = winkel2bogen l1
                      y2 = winkel2bogen l2
```

5. Aufgabe

Wenn die Seitenlängen eines rechtwinkligen Dreiecks **natürliche Zahlen** sind, werden diese Zahlen als **pythagoräische Zahlentripel** bezeichnet.

Definieren Sie eine Funktion in Haskell, die bei Eingabe dreier natürlicher Zahlen feststellen kann, ob es sich um die Seitenlängen eines rechtwinkligen Dreiecks handelt oder nicht. Mit anderen Worten, die Funktion soll testen, ob die eingegebenen natürlichen Zahlen pythagoräische Zahlentripel sind.

Anwendungsbeispiel: **pythagoras_tripel** 3 4 5 => True

Lösung:

```
pythagoras_tripel :: Integer -> Integer -> Integer -> Bool
pythagoras_tripel :: Integer -> Integer -> Integer -> Bool
pythagoras_tripel a b c  | a<=0 || b<=0 || c<=0 = False
                         | otherwise  = pyth a b c || pyth b c a || pyth c a b
                                        where
                                        pyth x y z = (x*x + y*y)==z*z
```

Alternative Lösungen:

```
pythagoras_tripel :: Integer -> Integer -> Integer -> Bool
pythagoras_tripel a b c
          | a<=0 || b<=0 || c<=0  = False
          | otherwise  = (max a (max b c))^2 == (min a b)^2 + (min c (max a b))^2

pythagoras_tripel :: Integer -> Integer -> Integer -> Bool
pythagoras_tripel a b c  | a>0 && b>0 && c>0 = a^2 + b^2 + c^2 - 2*(hypo^2) == 0
                         | otherwise = False
                                        where
                                        hypo = max a (max b c)
```

Wichtige Hinweise:

1) Verwenden Sie geeignete Namen für Ihre Variablen und Funktionsnamen, die den semantischen Inhalt der Variablen oder die Semantik der Funktionen wiedergeben.

2) Verwenden Sie vorgegebene Funktionsnamen, falls diese angegeben werden.

3) Kommentieren Sie Ihre Programme.

4) Verwenden Sie geeignete lokale Funktionen und Hilfsfunktionen in Ihren Funktionsdefinitionen.

5) Schreiben Sie in alle Funktionen die entsprechende Signatur.