16. März 2023

Haskellskripte

Dargestellt sind einige Lösungen der Aufgaben im Skript *Haskelleinführung*.

Listing 1: Einfache Funktionen, Basistypen

```
-- Loesungen zu den Uebungen vom Blatt Haskelleinfuehrung.pdf

-- Eine Testliste zum Testen
liste = [1,3,5,2,12,33,8]

-- ====================================================
-- Aufgabe: Summe, Produkt, Durchschnitt, geometrisches Mittel, harmonisches
    Mittel zweier und dreier Zahlen.
-- ====================================================
summe:: Int -> Int -> Int
summe a b = a+b
-- summe mit drei Zahlen entsprechend erweitern

add:: Float -> Float -> Float
add x y = x+y

produkt:: Float -> Float -> Float
produkt a b = a*b
-- produkt mit drei Zahlen entsprechend erweitern

arith:: Float -> Float -> Float
arith a b  = add a b / 2

{-
Variante für viele Elemente
arith:: [Float] -> Float
arith [] = 0
arith (x:xs) = add (x:xs) / len (x:xs)
-}

geom:: Float -> Float -> Float
geom a b = (produkt a b) ** (1/2)

{-
Variante für viele Elemente
geom:: [Float] -> Float
geom [] = 0
geom (x:xs) = (mul (x:xs)) ** (1 / len (x:xs))
-}
```

```haskell
harm:: Float -> Float -> Float
harm a b = 2 / add (1/a) (1/b)

{-
Variante für viele Elemente
harm:: [Float] -> Float
harm [] = 0
harm (x:xs) = len (x:xs) / add (rez (x:xs))
-}

-- ======================================================
--Aufgabe: Schreibe eine Funktion, die für gegebenes n die Summe der Zahlen von 1 bis n zurück gibt (Rekursion!).
-- ======================================================
sumn :: Int -> Int
sumn 0 = 0
sumn n = n + sumn(n-1)

-- Variante mit Liste [1..n]
sumn2 n = sum1' [1..n] 0
sum1' [] acc = acc
sum1' (x:xs) acc = sum1' xs (acc+x)

-- ======================================================
-- Programmiere die boolschen Funktionen, AND, EXOR (wer mag kann auch XNOR nehmen) und die Äquivalenz.
-- ======================================================

-- AND
und:: Bool -> Bool -> Bool
und True True = True
und _ _ = False

-- AND -> Variante
und2:: Bool -> Bool -> Bool
und2 x y = if x && y then True else False

-- XOR
xoder x y = if (not x && y) || (not y && x) then True else False

-- Äquivalenz
äqui:: Bool -> Bool -> Bool
äqui x y = if (not x && not y) || (x && y) then True else False

-- ======================================================
-- Aufgabe:  Es ist ein Halb- und Volladdierer zu programmieren.
-- ======================================================
```

```haskell
-- Halbaddierer

had:: Bool -> Bool -> (Bool,Bool)
had x y = if (not x && not y) then (False,False)
                              else if (x && not y) || (not x && y) then (False,True)
                              else if (x && y) then (True,False)
                              else (True,True)

-- Volladierer
fullader:: Bool -> Bool -> Bool -> (Bool,Bool)
fullader x y cin = (xoder s cin,(und cin s) || (und x y))
                   where s = xoder x y

-- ====================================================
-- Fakultäts- und Fibonaccizahl berechnen (Das muss natürlich sein. Zum Vergleichen gegenüber C\#.)
-- ====================================================
fak:: Int -> Int
fak 0 = 0
fak n = n*fak (n-1)

fib:: Int -> Int
fib 0 = 0
fib 1 = 1
fib n = fib (n-1) + fib (n-2)

-- endrekursive Variante
fib2 :: Int -> Int
fib2 n = foo n (0,1)
         where foo n (a,b) | n == 0 = a
                           | otherwise = foo (n - 1) (b,a + b)

-- ====================================================
-- Die Ackermann-Funktion
-- ====================================================
acker:: Integer -> Integer -> Integer
acker 0 n = n+1
acker m 0 = acker (m-1) 1
acker m n = acker (m-1) (acker m n-1)

-- ====================================================
-- Schreibe eine Funktion, die für gegebene b und k die k-fache Potenz von b berechnet.
-- ====================================================
pow:: Int -> Int -> Int
pow b 0 = 1
```

```
129 pow b k = b * (pow b (k-1))
130
131 -- ======================================================
132 -- Der ggt (n,m) ist zu ermitteln.
133 -- ======================================================
134 ggT :: Integer -> Integer -> Integer
135 ggT a b | b == 0 = a
136         | otherwise = ggT b (a `mod` b)
137
138 -- ======================================================
139 -- Schreibe eine Funktion, die für gegebene m und n die Zahl m genau n mal
    aufsummiert.
140 -- ======================================================
141 nsum:: Int -> Int -> Int
142 nsum 0 a = a
143 nsum n a = a + (nsum (n-1) a)
144
145 -- ======================================================
146 -- Benutze die beiden vorherigen Funktionen die für gegebene r, s und t die
    Summe s^t für t = 1 bis r berechnet.
147 -- ======================================================
148
149 -- todo
150
151
152 -- ======================================================
153 -- Die Türme von Hanoi sind rekursiv zu programmieren. Man wird staunen, es
    sind nur ein paar Zeilen nötig.
154 -- ======================================================
155 hanoi :: Integer -> a -> a -> a -> [(a, a)]
156 hanoi 0 _ _ _ = []
157 hanoi n a b c = hanoi (n-1) a c b ++ [(a,b)] ++ hanoi (n-1) c b a
158
159 -- ======================================================
160 -- Und noch ein paar interessante Funktionen
161 -- ======================================================
162
163 -- Aufgabe: teilerliste
164
165 teilerliste:: Int  -> [Int]
166 teilerliste n = [ i | i<-[1..n], mod n i == 0]
167
168 -- Aufgabe: primliste
169
170 -- todo
171
172 -- Aufgabe: pyTriple: Liste aller pythagoraeischen Tripel
```

```
174 pyTriple n = [(x,y,z)| x<-[2..n], y<-[x+1..n], z<-[y+1..n], x*x+y*y==z*z]

176 -- Aufgabe: Anzahl eines bestimmten Elementes in einer Liste

178 -- Aufgabe: Das Collatz-Problem

180 collatz 1 = 1
181 collatz n = if even n then collatz (n `div` 2) else collatz (3*n+1)

183 collatz2 1 = [1]
184 collatz2 n = if even n then (n `div` 2):collatz2 (n `div` 2) else (3*n+1):
        collatz2 (3*n+1)

186 -- Collatzfolge mit List Comprehensions
187 -- collatzlist n = [:collatz (n `div` 2) | even n]

189 -- Aufgabe: Verwendung von foldr und lambda-Funktion

191 foo = foldr (\x y -> (x+y)/2) 54 [12,4,10,6]

193 -- Aufgabe: Arbeitsweise von list comprehensions

195 pairs:: [a] -> [b] -> [(a,b)]
196 pairs xs ys = [(x,y) | x<-xs , y<-ys]

198 -- Aufgabe: quad und mal 2

200 mapListe = map ((*2).quad)
201             where quad x = x*x

203 -- Loesung mit $ Operator
204 mapListe2 = map (\x -> (*2)$quad x)
205              where quad x = x*x

207 -- Aufgabe: Vektoraddition

209 -- todo
```