Haskell Cheetsheet

Haskell Code Beispiele

Kommandozeile

```
-- ghci/REPL starten (Auf Kommandozeile)
-- >ghci
-- Zum beenden: Strg + D
-- Datei ausf hren:
-- > runhaskell <dateiname.hs>
-- Compilieren
-- > ghc <dateiname.hs> -o <executable>
```

Basics

```
"Hello" == ['H', 'e', 'l', 'l', 'o'] --
    True
[0, 2, 4, 6, 8, 10] == [0, 2..10] -- True
```

Listen

```
[1..10] !! 3 -- 4
[1..5] ++ [6..10] -- [1..10]
(++) [1,2] [3]  -- [1,2,3]
0:[1..5] -- [0, 1, 2, 3, 4, 5]
head [1..5] -- 1
tail [1..5] -- [2, 3, 4, 5]
init [1..5] -- [1, 2, 3, 4]
last [1..5] -- 5
null []          -- True
length [1,2,3]   -- 3
```

List Comprehensions

```
[x*2 | x <- [1..5], x*2 > 4] -- [6, 8,
    10]
```

Tuples

```
let tuple = ("haskell", 1)
fst tuple -- "haskell"
snd tuple -- 1
```

Funktionen

```
add a b = a + b
let add a b = a + b -- in ghci

add 1 2 -- 3
1 `add` 2 -- 3
```

Guards

```
oneOrTwo x
  | x == 2 = 2
  | otherwise = 1
```

Pattern Matching

```
oneOrTwo 2 = 2
oneOrTwo 1 = 1
```

Anonyme Funktionen

```
foldl1 (\acc x -> acc + x) [1..5] -- 15
```

Funktion filter

```
filter (>5) [1,2,3,4,5,6,7,8] -- [6, 7,
    8]
filter odd [3,6,7,9,12,14] -- [3, 7, 9]
filter (\x -> length x > 4) ["aaaa","
    bbbbbbbbbbbbb","cc"] -- ["
    bbbbbbbbbbbbb"]
```

Composition und Precedence

```
foo = (4*) . (10+)
foo 5 -- 4*(10+5) = 60

even (fib 7) -- False
even $ fib 7 -- False
even . fib $ 7 -- False
```

Import Packages

```
import Package.Name
import Package.Name ( func1, func2, func3
    )

\subsection*{Module Schreiben}
\begin{lstlisting}[style=haskell]
module ownmodule (
  funktion1,
  funktion2
) where

funktion1 x = "Hier regiert funktion1"
funktion2 x = "Hier handelt funktion2"
```

I/O

```
main = do
  putStrLn "Ich gebe eine Zeile aus"
  line <- getLine
  putChar 'a'
  char <- getChar
  let x = 10
  print x
```

Sonstiges

```
map f [x1, x2, x3] -- [f x1, f x2, f x3]
nub [1,2,3,4,1,2,3,4,5] -- [1,2,3,4,5]
words "Eine String mit W rtern" -- ["
    Eine", "String", "mit", "W rtern"]
unwords ["Eine", "Liste", "mit", "Strings
    "] -- "Eine Liste mit Strings"


x=""
y=[]
null x -- True
null y -- True

\subsection*{Algebraische Datentypen (ADT
    )}
\begin{lstlisting}[style=haskell]
data Color = Red | Blue | Green

say :: Color -> String
say Red = "You are Red!"
say Blue = "You are Blue!"
say Green = "You are Green!"
```

ADT mit Konstruktoren

```
data Person =
    Student String String Int Double
  | Teacher String String
  deriving (Show)

say :: Person -> String
say (Student _ _ _ _) = "Student!"
say (Teacher _ _) = "Teacher!"
```

ADT mit Pattern matching

```
say :: Person -> String
say (Student String String Int Double) =
    "Student!"
say (Teacher String String) = "Teacher!"
```

Typdecleration für Funktionen

```
addThree :: Int -> Int -> Int -> Int
addThree x y z = x + y + z
```

Maybe & Either

```
-- Maybe
data Maybe a = Nothing | Just a

-- Either
data Either a b = Left a | Right b
```

Typklassen und Instanzen

```
class Eq a where
  (==) :: a -> a -> Bool
  (/=) :: a -> a -> Bool

instance Eq Person where
    Teacher firstName1 lastName1 ==
        Teacher firstName2 lastName2 =
        firstName1 == firstName2 &&
            lastName1 == lastName2
    _ == _ = False
```

Rekursion: Basis- und Rekursionsfall

```
factorial :: Int -> Int
factorial 0 = 1
factorial n = n * factorial (n - 1)
```

Rekursion: Listen

```
length :: [a] -> Int
length [] = 0
length (_:xs) = 1 + length xs
```

Rekursion: Listen: Summe aller Listenelemente

```
sum :: [Int] -> Int
sum [] = 0
sum (x:xs) = x + sum xs
```

Tail-Rekursion

```
sumTail :: [Int] -> Int
sumTail = go 0
  where
    go acc []     = acc
    go acc (x:xs) = go (acc + x) xs
```

Monaden

```
-- Monadischen Typen erstellen
data Box a = Box a deriving (Show, Eq)

-- Werte in Monadische Typen verpacken
unit :: a -> Box a
unit x = Box x

-- Funktionen mit Monadischen typen
   entpacke und verpacken
bind :: Box a -> (a -> Box b) -> Box b
bind (Box x) f = f x

-- Hebe  Funktionen zu Funktionen mit
   Monadischen typen
lift :: (a -> b) -> Box a -> Box b
lift f (Box x) = Box (f x)
```