Signatur
Beispiel
Beschreibung

Signatur
Beispiel
Beschreibung

Signatur
Beispiel
Beschreibung

Signatur
Beispiel
Beschreibung

Signatur
Beispiel
Beschreibung

Signatur

Beispiel
Beschreibung

Signatur
Beispiel
Beschreibung

Signatur
Beispiel
Beschreibung

Signatur
Beispiel
Beschreibung

Signatur
Beispiel
Beschreibung

Signatur
Beispiel
Beschreibung

Signatur

Beispiel
Beschreibung

Signatur
Beispiel
Beschreibung

λ List Functions λ

Verwenden Sie ghci und http://www.haskell.org/hoogle um diese Liste zu vervollständiger

(++) :: [a] -> [a] -> [a]
[1,2] ++ [3,4,5] ~> [1,2,3,4,5] "Hallo " ++ show 12 ~> "Hallo 12"
Hängt zwei Listen aneinander. Wird entsprechend verwendet um Strings zu konkatenieren.

take :: Int -> [a] -> [a]
take 3 ['a','b','c','d','e'] ~> ['a','b','c']
Nimmt einen Integer n und eine Liste und gibt die ersten n Elemente der Liste zurück

drop :: Int -> [a] -> [a]
drop 2 [1,2,3] ~> [3]
Wirft die ersten n Elemente weg.

(!!) :: [a] -> Int -> a
[1,2,3,4] !! 2 ~> 3
List Index: Nimmt eine Liste und ein Integer n und gibt den Index n der Liste zurück

last :: [a] -> a
last [1,2,3,4] ~> 4
Gibt das letzte Element zurück.

init :: [a] -> [a]

init ['a','b','c','d'] ~> ['a','b','c']
Return all the elements of a list except the last one. The List must be non-empty

reverse :: [a] -> [a]
reverse "Hallo" ~> "ollaH
Dreht eine Liste um.

elem :: Eq a => a -> [a] -> Bool
elem 2 [1,3,4,5,6] ~> False
Does the element occur in the structure?

maximum :: Ord a => [a] -> a
maximum [1,4,3] ~> 4 minimum [1,4,3] ~> 1
The largest element of a non-empty structure.

sum :: Num a => [a] -> a
sum [1,2,3] ~> 6
Gibt die Summe/ das Produkt zurück. Die Listenelemente müssen von einem Zahlen Typen sein (Num).

zip :: [a] -> [b] -> [(a,b)]
zip [1,2,3] [3,4,5] ~> [(1,3),(2,4),(3,5)]
zip' takes two lists and returns a list of corresponding pairs

concat :: [[a]] -> [a]

concat [[1],[2,3],[4]] ~> [1,2,3,4] concat ["abc","def"] ~> "abcdef"
Nimt eine Liste von Listen entgegen und returned eine Liste. The concatenation of all the elements of a container of lists.

zipWith :: (a -> b -> c) -> [a] -> [b] -> [c]
zipWith (+) [1,2,3] [10,11,12] ~> [11,13,15] zipWith (++) ["Ha","Ec"] ["llo","ho"] ~> ["Hallo","Echo"]
/O(min(m,n))/. 'zipWith' generalises 'zip' by zipping with the function given as the first argument, instead of a tupling function. For example, @'zipWith' (+)@ is applied to two lists to produce the list of corresponding sums: >>> zipWith (+) [1, 2, 3] [4, 5, 6] [5,7,9]

า.