Haskell CheatSheet

length	:: [a] -> Int	length "Abc" = 3
reverse	:: [a] -> [a]	reverse "abc" = "cba"
drop	:: Int -> [a] -> [a]	drop 2 "abcd" = "cd"
dropWhile	:: (a -> Bool) -> [a] -> [a]	dropWhile (>3) [5,3,5] = [3,5]
take	:: Int -> [a] -> [a]	take 3 "abcde" = "abc"
takeWhile	:: (a -> Bool) -> [a] -> [a]	takeWhile (> 2) [3,2,1] = [3]
elem	:: Eq a => a -> [a] -> Bool	'a' `elem` "abc" = True
head	:: [a] -> a	tail "abc" = 'a'
tail	:: [a] -> [a]	tail "abc" = "bc"
init	:: [a] -> [a]	init "abcd" = "abc"
last	:: [a] -> a	last "abcde" = 'e'
null	:: [a] -> Bool	null [1,2,3] = false null [] = true
minimum	:: Ord a => [a] -> a	minimum [2,3,1,4] = 1 // <> maximum
filter	:: (a -> Bool) -> [a] -> [a]	filter (=='a') "abcabcabc" = "aaa"
foldl	:: (a -> b -> a) -> a -> [b] -> a	foldl (+) 0 [a,b,c] = ((0+a)+b)+c
foldr	:: (a -> b -> b) -> b -> [a] -> b	foldr (+) 0 [a,b,c] = a+(b+(c+0))
zip	:: [a] -> [b] -> [(a, b)]	zip "abc" "de" = [('a','d'), ('b',e')]
zipWith	:: (a -> b -> c) -> [a] -> [b] -> [c]	zipWith (+) [1,2] [3,4] = [4,6]
iterate	:: (a -> a) -> a -> [a]	iterate (+2) 1 = [1,3,5,7,...] = odds
repeat	:: a -> [a]	repeat 'a' = "aaaaaaaaa..."
replicate	:: Int -> a -> [a]	replicate 4 'a' = "aaaa"

(!!) Listenzugriff [0,1,2] !! 1 = 1 (&&) Boolean 'and'

(||) Boolean 'or' (++) Listen koncat "ab" ++ "de" = "abde"

(^) power (^^) power (negativer exp erlaubt)

Datentypen

```
data Stack t =  Empty | Stacked t (Stack t)

pop Empty          = error "Empty"      push x s = Stacked x s
pop (Stacked x s) = s

top Empty          = error "Empty"
top (Stacked x s) = x

someStack :: Stack Integer
someStack  = Stacked 3 (Stacked 1 Empty)
```

'Empty' und 'Stacked' sind Konstruktoren für Stack. Zugriff auf Felder über Konstruktoren