## --TYPSIGNATUREN

```
elem :: (Eq a) => a -> [a] -> Bool
-- Überprüft, ob ein Element in einer Liste enthalten ist.

foldr :: (a -> b -> b) -> b -> [a] -> b
-- Reduziert eine Liste von rechts nach links unter Anwendung einer Funktion.

scan :: (a -> a -> a) -> a -> [a] -> [a]
-- Erzeugt eine Liste von Zwischenergebnissen bei der Faltung von links nach
rechts.

odd :: Integral a => a -> Bool
-- Überprüft, ob eine ganze Zahl ungerade ist.

even :: Integral a => a -> Bool
-- Überprüft, ob eine ganze Zahl gerade ist.

fmap :: Functor f => (a -> b) -> f a -> f b
-- Wendet eine Funktion auf jeden Wert innerhalb eines Funktors an.

max :: Ord a => a -> a -> a
-- Gibt das größere von zwei Werten zurück.

span :: (a -> Bool) -> [a] -> ([a], [a])
-- Teilt eine Liste in zwei Teile: den maximalen Präfix, der die Bedingung
erfüllt, und den Rest.

filter :: (a -> Bool) -> [a] -> [a]
-- Wählt alle Elemente einer Liste aus, die eine bestimmte Bedingung erfüllen.

map :: (a -> b) -> [a] -> [b]
-- Wendet eine Funktion auf jedes Element einer Liste an.

concat :: [[a]] -> [a]
-- Verbindet eine Liste von Listen in eine einzige Liste.

concatMap :: (a -> [b]) -> [a] -> [b]
-- Wendet eine Funktion an, die Listen erzeugt, und verbindet die Ergebnisse.

reverse :: [a] -> [a]
-- Kehrt die Reihenfolge der Elemente in einer Liste um.

length :: [a] -> Int
-- Gibt die Anzahl der Elemente in einer Liste zurück.

null :: [a] -> Bool
-- Überprüft, ob eine Liste leer ist.
```

```haskell
head :: [a] -> a
-- Gibt das erste Element einer Liste zurück.

tail :: [a] -> [a]
-- Gibt alle Elemente einer Liste außer dem ersten zurück.

init :: [a] -> [a]
-- Gibt alle Elemente einer Liste außer dem letzten zurück.

last :: [a] -> a
-- Gibt das letzte Element einer Liste zurück.

take :: Int -> [a] -> [a]
-- Nimmt die ersten n Elemente einer Liste.

drop :: Int -> [a] -> [a]
-- Entfernt die ersten n Elemente einer Liste.

takeWhile :: (a -> Bool) -> [a] -> [a]
-- Nimmt die maximalen Präfixe einer Liste, solange die Bedingung erfüllt ist.

dropWhile :: (a -> Bool) -> [a] -> [a]
-- Entfernt den maximalen Präfix einer Liste, solange die Bedingung erfüllt
ist.

cycle :: [a] -> [a]
-- Wiederholt eine Liste unendlich oft.

repeat :: a -> [a]
-- Erzeugt eine unendliche Liste, in der ein Wert unendlich oft wiederholt
wird.

replicate :: Int -> a -> [a]
-- Erzeugt eine Liste, in der ein Wert n-mal wiederholt wird.

zip :: [a] -> [b] -> [(a, b)]
-- Kombiniert zwei Listen zu einer Liste von Paaren.

unzip :: [(a, b)] -> ([a], [b])
-- Zerlegt eine Liste von Paaren in zwei separate Listen.

and :: [Bool] -> Bool
-- Gibt `True` zurück, wenn alle Werte in einer Liste von Booleschen Werten
`True` sind.

or :: [Bool] -> Bool
-- Gibt `True` zurück, wenn mindestens ein Wert in einer Liste von Booleschen
Werten `True` ist.
```

```haskell
any :: (a -> Bool) -> [a] -> Bool
-- Gibt `True` zurück, wenn mindestens ein Element in einer Liste die
Bedingung erfüllt.

all :: (a -> Bool) -> [a] -> Bool
-- Gibt `True` zurück, wenn alle Elemente in einer Liste die Bedingung
erfüllen.

sum :: (Num a) => [a] -> a
-- Gibt die Summe aller Elemente in einer Liste zurück.

product :: (Num a) => [a] -> a
-- Gibt das Produkt aller Elemente in einer Liste zurück.

maximum :: (Ord a) => [a] -> a
-- Gibt das größte Element in einer Liste zurück.

minimum :: (Ord a) => [a] -> a
-- Gibt das kleinste Element in einer Liste zurück.

foldl :: (b -> a -> b) -> b -> [a] -> b
-- Reduziert eine Liste von links nach rechts unter Anwendung einer Funktion.

foldl1 :: (a -> a -> a) -> [a] -> a
-- Reduziert eine Liste von links nach rechts unter Anwendung einer Funktion
ohne Anfangswert.

foldr1 :: (a -> a -> a) -> [a] -> a
-- Reduziert eine Liste von rechts nach links unter Anwendung einer Funktion
ohne Anfangswert.

scanl :: (b -> a -> b) -> b -> [a] -> [b]
-- Erzeugt eine Liste von Zwischenergebnissen bei der Faltung von links nach
rechts mit Anfangswert.

scanr :: (a -> b -> b) -> b -> [a] -> [b]
-- Erzeugt eine Liste von Zwischenergebnissen bei der Faltung von rechts nach
links mit Anfangswert.

sequence :: (Monad m) => [m a] -> m [a]
-- Führt eine Liste von Monaden aus und kombiniert die Ergebnisse zu einer
Monade über einer Liste.

sequence_ :: (Monad m) => [m a] -> m ()
-- Führt eine Liste von Monaden aus und ignoriert die Ergebnisse.
```