[] Instanzen

```haskell
instance Eq a => Eq [a] where
  [] == [] = True
  (x:xs) == (y:ys) = x == y && xs == ys
  _ == _ = False

instance Ord a => Ord [a] where
  compare [] [] = EQ
  compare [] _  = LT
  compare _ []  = GT
  compare (x:xs) (y:ys) = compare x y <> compare xs ys

instance Show a => Show [a] where
  showsPrec _ = showList

instance Read a => Read [a] where
  readsPrec = readList

instance Functor [] where
  fmap _ [] = []
  fmap f (x:xs) = f x : fmap f xs

instance Applicative [] where
  pure x = [x]
  fs <*> xs = [f x | f <- fs, x <- xs]

instance Monad [] where
  return x = [x]
  xs >>= f = concatMap f xs

instance Foldable [] where
  foldr _ z [] = z
  foldr f z (x:xs) = f x (foldr f z xs)

  foldl _ z [] = z
  foldl f z (x:xs) = foldl f (f z x) xs

  foldMap _ [] = mempty
  foldMap f (x:xs) = f x `mappend` foldMap f xs
```

```haskell
instance Traversable [] where
  traverse _ [] = pure []
  traverse f (x:xs) = (:) <$> f x <*> traverse f xs

instance Alternative [] where
  empty = []
  (<|>) = (++)

instance Semigroup [a] where
  (<>) = (++)

instance Monoid [a] where
  mempty = []
  mappend = (<>)
  mconcat = concat

instance MonadPlus [] where
  mzero = []
  mplus = (++)

instance IsList [a] where
  type Item [a] = a
  fromList = id
  toList = id
```