The function `map`

```
suclist :: [Int] -> [Int]
suclist []     = []
suclist (x:xs) = suc x : suclist xs

sqrtlist :: [Float] -> [Float]
sqrtlist []     = []
sqrtlist (x:xs) = sqrt x : sqrtlist xs

map :: (a -> b) -> [a] -> [b]
map g [] = []
map g (x:xs) = g x : map g xs
```

⇓

```
suclist :: [Int] -> [Int]          sqrtlist :: [Float] -> [Float]
suclist  = map suc                 sqrtlist = map sqrt
```

The function `filter`

```
dropEven :: [Int] -> [Int]
dropEven [] = []
dropEven (x:xs) | odd x       = x : dropEven xs
                | otherwise = dropEven xs
```

```
dropUpper :: [Char] -> [Char]
dropUpper [] = []
dropUpper (x:xs) | isLower x = x : dropUpper xs
                 | otherwise = dropUpper xs
```

```
filter :: (a -> Bool) -> [a] -> [a]
filter g [] = []
filter g (x:xs) | g x         = x : filter g xs
                | otherwise = filter g xs
```

$$\Downarrow$$

```
dropEven :: [Int] -> [Int]
dropEven  = filter odd
```

```
dropUpper :: [Char] -> [Char]
dropUpper = filter isLower
```

The function fold

```
add :: (List Int) -> Int            prod :: (List Int) -> Int
add Nil         = 0                 prod Nil         = 1
add (Cons x xs) = plus x (add xs)   prod (Cons x xs) = times x (prod xs)

            concat :: List (List a) -> List a
            concat Nil         = Nil
            concat (Cons x xs) = append x (concat xs)

            fold :: (a -> b -> b) -> b -> (List a) -> b
            fold g e Nil         = e
            fold g e (Cons x xs) = g x (fold g e xs)
```

⇓

```
add :: (List Int) -> Int              prod :: (List Int) -> Int
add  = fold plus 0                    prod = fold times 1

            concat :: List (List a) -> List a
            concat = fold append Nil
```

The function `foldr`

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

```
prod :: [Int] -> Int
prod []      = 1
prod (x:xs)  = x * prod xs
```

```
concat :: [[a]] -> [a]
concat []      = []
concat (x:xs)  = x ++ concat xs
```

```
foldr :: (a -> b -> b) -> b -> [a] -> b
foldr g e []      = e
foldr g e (x:xs)  = g x (foldr g e xs)
```

$\Downarrow$

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

```
prod :: [Int] -> Int
prod = foldr (*) 1
```

```
concat :: [[a]] -> [a]
concat = foldr (++) []
```