1.

```
lst :: [a] -> a
lst []     = error "emptyList"
lst [x]    = x
lst (_:xs) = lst xs

initial :: [a] -> [a]
initial []     = error "emptyList"
initial [_]    = []
initial (x:xs) = x : initial xs

repl :: Int -> a -> [a]
repl n x
  | n <= 0    = []
  | otherwise = x : repl (n-1) x

drp :: Int -> [a] -> [a]
drp n xs
  | n <= 0    = xs
drp _ []      = []
drp n (_:xs)  = drp (n-1) xs

tk :: Int -> [a] -> [a]
tk n _
  | n <= 0    = []
tk _ []       = []
tk n (x:xs)   = x : tk (n-1) xs

has :: Eq a => a -> [a] -> Bool
has _ [] = False
has y (x:xs)
```

```
  | y == x    = True
  | otherwise = has y xs

concat2 :: [a] -> [a] -> [a]
concat2 [] ys     = ys
concat2 (x:xs) ys = x : concat2 xs ys
```

2.

```
join :: [Char] -> [[Char]] -> [Char]
join _ []      = ""
join _ [x]     = x
join sep (x:xs) = x ++ sep ++ join sep xs
```

3.

```
splits :: [a] -> [([a], [a])]
splits xs
  | length xs < 2 = error "shortList"
  | otherwise     = go 1
  where
    n = length xs
    go ix
      | i >= n    = []
      | otherwise = (take i xs, drop i xs) : go (i + 1)
```

4.

```
rme :: Integral a => a -> a
rme n = helper n 0 1
  where
    helper 0 acc _ = acc
    helper m acc place =
      let (q, r) = m `divMod` 10
```

```
in if odd r
   then helper q (acc + r * place) (place * 10)
   else helper q acc place
```