# 99 questions/Solutions/7

## From HaskellWiki

< 99 questions | Solutions

(**) Flatten a nested list structure.

```
data NestedList a = Elem a | List [NestedList a]

flatten :: NestedList a -> [a]
flatten (Elem x) = [x]
flatten (List x) = concatMap flatten x
```

or without concatMap

```
flatten :: NestedList a -> [a]
flatten (Elem a   )   = [a]
flatten (List (x:xs)) = flatten x ++ flatten (List xs)
flatten (List [])     = []
```

or using things that act just like
`concatMap`

```
flatten (Elem x) = return x
flatten (List x) = flatten =<< x

flatten (Elem x) = [x]
flatten (List x) = foldMap flatten x
flatten2 :: NestedList a -> [a]
flatten2 a = flt' a []
  where flt' (Elem x)      xs = x:xs
        flt' (List (x:ls)) xs = flt' x (flt' (List ls) xs)
        flt' (List [])     xs = xs
```

or with foldr

```
flatten3 :: NestedList a -> [a]
flatten3 (Elem x ) = [x]
flatten3 (List xs) =  foldr (++) [] $ map flatten3 xs
```

or with an accumulator function:

```
flatten4 = reverse . rec []
  where
  rec acc (List []) = acc
  rec acc (Elem x)  = x:acc
  rec acc (List (x:xs)) = rec (rec acc x) (List xs)
```

We have to define a new data type, because lists in Haskell are homogeneous. [1, [2, [3, 4], 5]] is a type error. Therefore, we must have a way of representing a list that may (or may not) be nested.

Our NestedList datatype is either a single element of some type (Elem a), or a list

of NestedLists of the same type. (List [NestedList a]).


Category:

- Programming exercise spoilers

---