99 questions/Solutions/16

From HaskellWiki

< 99 questions | Solutions

(**) Drop every N'th element from a list.

```
dropEvery :: [a] -> Int -> [a]
dropEvery [] _ = []
dropEvery (x:xs) n = dropEvery' (x:xs) n 1 where
    dropEvery' (x:xs) n i = (if (n `divides` i) then
        [] else
        [x])
        ++ (dropEvery' xs n (i+1))
    dropEvery' [] _ _ = []
    divides x y = y `mod` x == 0
```

An alternative iterative solution:

```
dropEvery :: [a] -> Int -> [a]
dropEvery list count = helper list count count
  where helper [] _ _ = []
        helper (x:xs) count 1 = helper xs count count
        helper (x:xs) count n = x : (helper xs count (n - 1))
```

A similar iterative solution but using a closure:

```
dropEvery :: [a] -> Int -> [a]
dropEvery xs n = helper xs n
    where helper [] _ = []
          helper (x:xs) 1 = helper xs n
          helper (x:xs) k = x : helper xs (k-1)
```

Or, counting up (and using guards instead of pattern matching):

```
dropEvery :: [a] -> Int -> [a]
dropEvery xs n = helper xs 1
  where helper :: [a] -> Int -> [a]
        helper [] _ = []
        helper (x:xs) i
          | i == n  = helper xs 1
          | i /= n  = x:helper xs (i + 1)
```

Yet another iterative solution which divides lists using Prelude:

```
dropEvery :: [a] -> Int -> [a]
dropEvery [] _ = []
dropEvery list count = (take (count-1) list) ++ dropEvery (drop count list) count
```

A similar approach using guards:

```
dropEvery :: [a] -> Int -> [a]
dropEvery xs n
  | length xs < n = xs
  | otherwise     = take (n-1) xs ++ dropEvery (drop n xs) n
```

Using zip:

```
dropEvery = flip $ \n -> map snd . filter ((n/=) . fst) . zip (cycle [1..n])
```

Using zip and list comprehensions

```
dropEvery :: [a] -> Int -> [a]
dropEvery xs n = [ i | (i,c) <- ( zip xs [1,2..]), (mod c n) /= 0]
```

A more complicated approach which first divides the input list into sublists that do not contain the nth element, and then concatenates the sublists to a result list (if not apparent: the author's a novice):

```
dropEvery :: [a] -> Int -> [a]
dropEvery [] _ = []
dropEvery xs n = concat (split n xs)
 where
  split _ [] = []
  split n xs = fst splitted : split n ((safetail . snd) splitted)
   where
    splitted = splitAt (n-1) xs
    safetail xs | null xs = []
                | otherwise = tail xs
```

First thing that came to mind:

```
dropEvery xs n = map fst $ filter (\(x,i) -> i `mod` n /= 0) $ zip xs [1..]
```

The filter function can be simplified as seen above:

```
dropEvery xs n = map fst $ filter ((n/=) . snd) $ zip xs (cycle [1..n])
```

And yet another approach using folds:

```
dropEvery :: Int -> [a] -> [a]
dropEvery n xs = snd $ foldl (\acc e -> if fst acc > 1 then (fst acc - 1, snd acc ++ [e]) els
```

Another very similar approach to the previous:

```
dropEvery :: [a] -> Int -> [a]
dropEvery xs n = fst $ foldr (\x (xs, i) -> (if mod i n == 0 then xs else x:xs, i - 1)) ([],
```

Another foldl solution:

```
dropEvery :: [a] -> Int -> [a]
dropEvery lst n = snd $ foldl helper (1, []) lst
    where helper (i,acc) x = if n == i
                             then (1,acc)
                             else (i+1,acc++[x])
```

Retrieved from "https://wiki.haskell.org/index.php?title=99_questions/Solutions/16&oldid=59149"

Category:

- Programming exercise spoilers
