99 questions/Solutions/17

From HaskellWiki

< 99 questions | Solutions

(*) Split a list into two parts; the length of the first part is given.

Do not use any predefined predicates.

Solution using
take
and
drop
:

```
split xs n = (take n xs, drop n xs)
```

Or even simpler using
splitAt
:

```
split = flip splitAt
```

But these should clearly be considered "predefined predicates". Alternatively, we have the following recursive solution:

```
split :: [a] -> Int -> ([a], [a])
split []          _             = ([], [])
split l@(x : xs) n | n > 0      = (x : ys, zs)
                   | otherwise = ([], l)
    where (ys,zs) = split xs (n - 1)
```

The same solution as above written more cleanly:

```
split :: [a] -> Int -> ([a], [a])
split (x:xs) n | n > 0 = let (f,l) = split xs (n-1) in (x : f, l)
split xs _             = ([], xs)
```

Or (ab)using the "&&&" arrow operator for tuples:

```
split :: [a] -> Int -> ([a], [a])
split (x:xs) n | n > 0 = (:) x . fst &&& snd $ split xs (n - 1)
split xs _             = ([], xs)
```

A similar solution using foldl:

```
split :: [a] -> Int -> ([a], [a])
split [] _ = ([], [])
split list n
  | n < 0 = (list, [])
  | otherwise  = (first output, second output)
    where output = foldl (\acc e -> if third acc > 0 then (first acc ++ [e], second acc, thir
```

Note that for the above code to work you must define your own first, second, and third functions for tuples containing three elements like so:

```
first :: (a, b, c) -> a
first (x, _, _) = x

second :: (a, b, c) -> b
second (_, y, _) = y

third :: (a, b, c) -> c
third (_, _, z) = z
```

Another foldl solution without defining tuple extractors:

```
split :: [a] -> Int -> ([a],[a])
split lst n = snd $ foldl helper (0,([],[])) lst
    where helper (i,(left,right)) x = if i >= n then (i+1,(left,right++[x])) else (i+1,(left+
```

A solution that dequeues onto a stack and then reverses at the end:

```
split :: [a] -> Int -> ([a], [a])
split xs n = let (a, b) = helper [] xs n in (reverse a, b)
  where helper left right@(r:rs) n
          | n == 0    = (left, right)
          | otherwise = helper (r:left) rs (n - 1)
```

A recursive solution constructing the 2-tuple:

```
split :: [a] -> Int -> ([a],[a])

split [] _ = ([],[])

split (x:xs) n
  | n > 0  = (
      x: (fst (split xs (n-1))),
      snd (split xs (n-1))
    )
  | n <= 0 = (
      fst (split xs 0),
      x:(snd (split xs 0))
    )
```


Category:

- Programming exercise spoilers
