

# Exercises

## Exercises on foldr

### Exercise 1

Using the higher-order function *foldr*, define a function *sumsq* which takes an integer *n* as its argument and returns the sum of the squares of the first *n* integers. That is to say,  $sumsq\ n = 1^2 + 2^2 + 3^2 + \dots + n^2$ .  
(answer given to start you off)

```
sumsq :: Integral a => a -> a
sumsq n = foldr op 0 [1..n]
      where op :: Num a => a -> a -> a
            op x y = x*x + y
```

### Exercise 2

Define *lengthr*, which returns the number of elements in a list, using *foldr*.

### \*\*\*\*\* Exercise 3

Define *minlist*, which returns the smallest integer in a non-empty list of integers, using *foldr1*. (*foldr1* is a Prelude function - look it up yourself or continue and come back to this)

### \*\*\*\* Exercise 4

Define *myreverse*, which reverses a list, using *foldr*.

### \*\*\*\* Exercise 5

Using *foldr*, define a function *remove* which takes two strings as its arguments and removes every letter from the second list that occurs in the first list. For example,

`remove "first" "second" = "econd"`.

**Hint:** Use a helper function in your lambda

### \*\*\*\*\* Exercise 6

The function *remdups* removes adjacent duplicates from a list. For example,

`remdups [1, 2, 2, 3, 3, 3, 1, 1] = [1, 2, 3, 1]`

Define *remdups* using *foldr*.

# Solutions

## Solutions to exercise 2

```
lengthr :: [Int] -> Int
lengthr = foldr (\x y -> 1 + y) 0
```

## Solutions to exercise 3

```
minlitr :: [Int] -> Int
minlitr = foldr1 min
```

## Solutions to exercise 4

```
myreverse :: [a] -> [a]
myreverse = foldr (\x y -> y ++ [x]) []
```

## Solutions to exercise 5

```
myremove :: Eq a => [a] -> [a] -> [a]
myremove xs = foldr (\y processed -> (aux y xs) ++ processed)
[]
    where aux :: Eq a => a -> [a] -> [a]
          aux x ys | x `elem` ys = []
                  | otherwise    = [x]
```

## Solutions to exercise 6

```
remdupsr :: Eq a => [a] -> [a]
remdupsr [] = []
remdupsr (y:ys) = foldr joinr [y] ys

joinr :: Eq a => a -> [a] -> [a]
joinr x [] = [x]
joinr x xs
    | x == head xs = xs
    | otherwise    = [x] ++ xs
```