COMP105 Lecture 16

More on fold

Revisiting the fold type

```
foldr :: (a -> b -> b) -> b -> [a] -> b
```

Note that **two** type variables are used here

- The input list has type `[a]`
- The accumulator has type `b`

So a fold can output a **different** type to the input list

Folds that output different types

```
sum_of_lengths list =
    foldr (\x acc -> acc + length x) 0 list
```

```
ghci> sum_of_lengths ["one", "two", "three"]
11
```

Folds that output different types

```
to_csv list =
    foldr (\x acc -> show x ++ "," ++  acc) "" list

ghci> to_csv [1,2,3,4]
"1,2,3,4,"
```

foldr1

The function `foldr1` uses the **final value** of the list to initialize the accumulator

```
foldr1' _ []     = error "empty list"
foldr1' _ [x]    = x
foldr1' f (x:xs) = f x (foldr1' f xs)
```

```
ghci> foldr1' (+) [1,2,3,4,5]
15
```

`foldr1`

Note that the type of `foldr1` is

```
foldr1' :: (a -> a -> a) -> [a] -> a
```

The accumulator has the same type as the list elements

- So `foldr1` **cannot** be used to change the type of a list

foldr1 examples

```
sum' list = foldr1 (+) list

product' list = foldr1 (*) list

concat' list = foldr1 (++) list
```

```
ghci> concat [[1,2,3], [4], [3,2,1]]
[1,2,3,4,3,2,1]
```

foldr1 examples

```
maximum' list =
    foldr1 (\ x acc -> if x > acc then x else acc) list
```

```
ghci> maximum [1,2,3,4,3,2,1]
4
```

Folding right

`foldr` processes lists from the **right**

```
foldr (+) 0 [1..4]

= 1 + (2 + (3 + (4 + 0)))
```

```
foldr (/) 1 [1..4]

= 1 / (2 / (3 / (4 / 1)))
= 0.375
```

Folding left

`foldl` processes lists from the **left**

```
foldl (+) 0 [1..4]

= ((((0 + 1) + 2) + 3) + 4)
```

```
foldl (/) 1 [1..4]

= ((((1 / 1) / 2) / 3) / 4)
= 0.0416
```

The type of `foldl`

```
foldr :: (a -> b -> b) -> b -> [a] -> b
foldl :: (b -> a -> b) -> b -> [a] -> b
```

Observe that the function `f` has its type **flipped**

- `foldr (\ x acc -> ...`
- `foldl (\ acc x -> ...`

Reversing a list with `foldl`

```
reverse_list list = foldl (\ acc x -> x : acc) [] list
```

```
ghci> reverse_list "hello"
"olleh"
```

Exercises

1. Use `foldr` to write a function `sumFsts` that takes a list of pairs and returns the sum of the first elements of the pairs

2. Use `foldr1` to write a function `minimum` that takes a list of numbers and returns the smallest element of that list

3. Use `foldl` to write a function `dash` that takes a string and inserts a `'-'` character after each character in the string