COMP105 Lecture 16

Fold

Recap: list recursion

Some functions take lists and turn them into a **single value**

```
sum' [] = 0
sum' (x:xs) = x + sum' xs

ghci> sum [1..10]
55
```

```
product' [] = 1
product' (x:xs) = x * product' xs

ghci> product [1..10]
3628800
```

Recap: list recursion

```
sum' [] = 0
sum' (x:xs) = x + sum' xs
```

The only things that **change** are

- The initial value: `0`, `1`, ...
- The operation use in each recursive step: `+`, `*`, ...

These are examples of **folds**

Foldr

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

foldr' _ init []     = init
foldr' f init (x:xs) = f x (foldr' f init xs)
```

```
ghci> foldr' (+) 0 [1..10]
55

ghci> foldr' (*) 1 [1..10]
3628800
```

The folded function

```
sum'' list = foldr (\ x acc -> acc + x) 0 list
```

The folded function `f` takes two arguments

- `x` is an element from the list
- `acc` is the **accumulator**

The function outputs a **new** value for the accumulator

- The initial value for the accumulator is `init`
- The final value for the accumulator is the output of `foldr`

Foldr

Consider:

```
foldr (\ x acc -> acc + x) 0 [1,2,3,4]
```

Values for the accumulator:

```
init  = 0
0 + 4 = 4
4 + 3 = 7
7 + 2 = 9
9 + 1 = 10
```

Final output: 10

An imperative equivalent

```
foldr f init input_list
```

In **python** this would be implemented as

```
acc = init
input_list.reverse()

for i in range(len(input_list)):
    acc = f(input_list[i], acc)

return acc
```

Foldr examples

```
concat' list = foldr (++) [] list

ghci> concat' ["a", "big", "bad"]
"abigbad"


all' list = foldr (&&) True list

ghci> all' [True, True, True]
True


length' list = foldr (\_ acc -> acc + 1) 0 list

ghci> length' [1,2,3,4]
4
```

Foldr examples

```
count_ones list =
    foldr (\ x acc -> if x == 1 then acc + 1 else acc)
          0 list
```

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

Exercises

Use `foldr` to solve the following problems

1. Write a function `any'` that takes a list of booleans and returns true if any of the elements are true

2. Write a function `countEs` that takes a string and counts the number of `'e'` characters

3. Write a function `sumOfEvens` that takes a list and sums the even numbers in the list