COMP105 Lecture 9

# Recursion with multiple lists

## Recursion across multiple lists

Sometimes we want to use recursion on more than one list

```
add_lists _ [] = []
add_lists [] _ = []
add_lists (x:xs) (y:ys) = x+y : add_lists xs ys
```

```
ghci> add_lists [1..5] [1..5]
[2,4,6,8,10]
```

- Base cases stop when either of the lists is empty
- Recursive rule pulls an element from both lists

## Testing whether two lists are equal

```haskell
list_equal [] [] = True
list_equal _  [] = False
list_equal []  _ = False
list_equal (x:xs) (y:ys)
    | x == y    = list_equal xs ys
    | otherwise = False
```

In reality, you should use == to test list equality

- But this is how == is implemented

## Splitting a list in two

Other functions can take a list and return a pair of lists

```
gt_10 [] = ([], [])
gt_10 (x:xs)
    | x > 10    = (x:gt, lt)
    | otherwise = (gt, x:lt)
    where (gt, lt) = gt_10 xs
```

```
ghci> gt_10 [8..12]
([11,12],[8,9,10])
```

- The base case sets up the tuple
- The recursive rule modifies one of the two lists

## Zip

**Zip** takes two lists and returns a list of pairs

```
zip' [] _ = []
zip' _ [] = []
zip' (x:xs) (y:ys) = (x, y) : zip' xs ys
```

```
ghci> zip' [1,2,3] ['a', 'b', 'c']
[(1,'a'),(2,'b'),(3,'c')]
```

This is frequently used in functional programming

## Exercises

1. Write a recursive function `multiplyLists` that takes two inputs lists and multiplies the lists together element-wise. So `multiplyLists [2,4,6] [10, 20, 30] = [20, 80, 180]`

2. Write a recursive function `zip3'` that takes three lists and zips them together, so `zip3' [1,2] [3, 4] "ab" = [(1,3,'a'),(2,4,'b')]`