COMP105 Lecture 17

More Higher Order Functions

takeWhile

takeWhile takes from a list **while a condition** is true

```
ghci> takeWhile (<=5) [1..10]
[1,2,3,4,5]
```

```
ghci> takeWhile (/=' ') "one two three"
"one"
```

```
ghci> takeWhile (\ x -> length x <= 2)
                                    ["ab", "cd", "efg"]
["ab","cd"]
```

takeWhile implementation

```haskell
takeWhile' :: (a -> Bool) -> [a] -> [a]

takeWhile' _ [] = []
takeWhile' f (x:xs)
    | f x = x : takeWhile' f xs
    | otherwise = []
```

dropWhile

dropWhile **drops** from a list while a condition is true

```
ghci> dropWhile (==1) [1,1,2,2,3,3]
[2,2,3,3]
```

```
ghci> dropWhile (`elem` ['a'..'z']) "smallBIG"
"BIG"
```

```
ghci> dropWhile (\x -> x < 10 && x > 0) [1,2,3,10,4,5]
[10,4,5]
```

dropWhile implementation

```haskell
dropWhile' :: (a -> Bool) -> [a] -> [a]

dropWhile' _ [] = []
dropWhile' f (x:xs)
    | f x = dropWhile' f xs
    | otherwise = x:xs
```

takeWhile and dropWhile example

```
split_words "" = []
split_words string =
    let
        first = takeWhile (/=' ') string
        up_to_space = dropWhile (/=' ') string
        after_space = dropWhile (==' ') up_to_space
    in
        first : split_words after_space
```

```
ghci> split_words "one two   three"
["one","two","three"]
```

words and unwords

The `split_words` function is called **words**

```
ghci> words "   foo   bar baz "
["foo","bar","baz"]
```

```
ghci> unwords ["foo","bar","baz"]
"foo bar baz"
```

Recap: zip

```
ghci> zip [1,2,3,4] [5,6,7,8]
[(1,5),(2,6),(3,7),(4,8)]
```

```
add_two_lists l1 l2 =
let
    zipped = zip l1 l2
in
    map (\ (x, y) -> x + y) zipped
```

```
ghci> add_two_lists [1,2,3,4] [5,6,7,8]
[6,8,10,12]
```

zipWith

`zipWith` zips two lists together **using a function**

```
ghci> zipWith (+) [1,2,3] [4,5,6]
[5,7,9]
```

```
ghci> zipWith (++) ["big", "red"] ["dog", "car"]
["bigdog","redcar"]
```

```
ghci> zipWith (\ x y -> if x then y else -y)
                    [True, False, False] [1,2,3]
[1,-2,-3]
```

zipWith implementation

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

zipWith' _ [] _ = []
zipWith' _ _ [] = []
zipWith' f (x:xs) (y:ys) = f x y : zipWith' f xs ys
```

zipWith examples

```
mult_by_pos list = zipWith (*) list [0..]
```

```
ghci> mult_by_pos [2,3,4,5]
[0,3,8,15]
```

zipWith examples

```
interleave str1 str2 =
    let
        zipped = zipWith (\ x y -> x : y : []) str1 str2
    in
        concat zipped
```

```
ghci> zipWith (\ x y -> x : y : []) "abc" "123"
["a1","b2","c3"]
```

```
ghci> interleave "abc" "123"
"a1b2c3"
```

Exercises

1. Write a function `getNumber` that takes a string and outputs the longest prefix of that string that contains only number characters. So `getNumber "1234hello5" = "1234"`

2. Write a function `wordCount` that takes a string and returns the number of words in the string

3. Write a function `numberWords` that takes a string and returns a list of tuples where the first element of each tuple is a word from the string, and the second element is the position of that word. So `numberWords "a test string"`
   `= [("a",1),("test",2),("string",3)]`