# Tutorial 3A: Map, filter, zip

We will practice the **higher-order** functions `map`, `filter`, `zip`, and `zipWith`. The function `map` takes as its first argument a function `f :: a -> b` and applies it to each element in a list, its second argument.

```
map :: (a -> b) -> [a] -> [b]
map _ [] = []
map f (x:xs) = f x : map f xs
```

The function `filter` takes a **property** `p :: a -> Bool` (something that is true or false for a value of type `a`) and selects those elements from a list for which the property is true.

```
filter :: (a -> Bool) -> [a] -> [a]
filter _ [] = []
filter p (x:xs)
    | p x       = x : filter p xs
    | otherwise = filter p xs
```

The function `zip` combines two lists into a list of pairs. If one list is longer than the other, the remaining elements are discarded.

```
zip :: [a] -> [b] -> [(a,b)]
zip _ [] = []
zip [] _ = []
zip (x:xs) (y:ys) = (x,y) : zip xs ys
```

The function `zipWith` is similar, except that instead of forming pairs, it combines the two lists with an arbitrary function.

```
zipWith :: (a -> b -> c) -> [a] -> [b] -> [c]
zipWith f _ [] = []
zipWith f [] _ = []
zipWith f (x:xs) (y:ys) = f x y : zip xs ys
```

**Exercise 1:** Write two versions of each of the following functions, one using **recursion** and one using **map** and **filter**.

a) A function `doubles` that multiplies every number in a list by two. It may be helpful to create a function `double` to double a single number.

b) A function `odds` that removes any even numbers from a list (use the built-in `odd`).

c) A function `doubleodds` that doubles every odd number in a list (and removes the even numbers).

```
*Main> doubles [1..10]
[2,4,6,8,10,12,14,16,18,20]
*Main> odds [1..10]
[1,3,5,7,9]
*Main> doubleodds [1..10]
[2,6,10,14,18]
```

**Exercise 2:**

a) Complete the function `shorts` that removes all strings longer than 5 characters from a list, using `filter`. Use a where-clause to define the filtering property.

b) Complete the function `squarePositives` that takes all positive integers in a list and squares them. Use `map` and `filter`, with a where-clause to define any auxiliary functions you might need.

c) Complete the function `oddLengthSums` that given a list of integer lists, returns for each odd-length list its sum. Use `map` and `filter`, and you may find `odd`, `length`, and `sum` helpful.

d) As an optional challenge, use **sections** or **anonymous functions** to avoid the use of where-clauses in the above functions.

```
*Main> shorts ["The","truth","is","that","in","London","it","is",
                                    "always","a","sickly","season"]
["The","truth","is","that","in","it","is","a"]
*Main> squarePositives [-3,4,1,-2,0,3]
[16,1,9]
*Main> oddLengthSums [[1],[1,2],[1,2,3],[1..4],[1..5]]
[1,6,15]
```

**Exercise 3:**

a) Complete the functions `remove` and `removeAll` from Tutorial 2B again using `filter`. This time, if an element occurs multiple times, remove all occurrences. For `removeAll`, consider using the functions `not` and `elem`.

b) Complete the function `numbered` that indexes the elements in a list by pairing them with numbers, counting up from 1. Use the function `zip`.

c) Complete the function `everyother` that takes every other element from a list, starting with the first. Use `numbered` to count elements, `filter`, `fst`, and `odd` to select those pairs to keep, and `map` and `snd` to remove the indexing again.

d) Complete the function `same` that takes two lists and returns a list of the positions where their elements coincide. For instance, the strings `"Mary"` and `"Jane"` have the same 2nd characters, so `same` should return the list `[2]`. Use the functions `filter`, `map`, `fst`, `snd`, `zip` (or `numbered`), and `zipWith`.

```
*Main> remove "Goerge" 'e'
"Gorg"
*Main> removeAll "Fitzwilliam" "Fitz"
"wllam"
*Main> numbered "days"
[(1,'d'),(2,'a'),(3,'y'),(4,'s')]
*Main> everyother "Elizabeth"
"Eiaeh"
*Main> same "Charles" "Charlotte"
[1,2,3,4,5]
```