1 It begins

This exercise sheet is all about higher order functions (functions that take functions as a parameter), the most important ones that you will get to understand are `map, filter, foldr, foldl`, and along with them you will sometimes have to use lambda functions. At the time of writing, there are no exercises explicitly for practicing lambdas.

1.1 Map

- Make your own implementation of map for lists, `mymap`.

`map` is often used in conjunction with a function that makes some transformation on an element. Use `map` or `mymap` to solve the following exercises. When a helper function is needed, you can write a function like normal or use a lambda function.

- Get a list of square roots, from a list of numbers.
- Get a list of lengths, from a list of strings.
- Use `toUpper` from the `Data.Char` module, to turn a string to uppercase.
- Make a function that takes a list of strings and turns them all to uppercase.
- Add your favorite number to a list of numbers.
- Double the numbers in a list unless they exceed some threshold.
- Given a list, make a list of lists (each element is its own list).
- Using `fizzbuzz` from the second exercise sheet (if you made it), make a list of the first few (e.g. 20) "numbers" in FizzBuzz.
- Make an infinite list of the square numbers, i.e. 1, 4, 9, 16, ...
- Zip the list with `[1..]` and use a lambda to make a list of the numbers added together, i.e. from `[(1,1),(2,4),(3,9),..]` you should get `[2,6,12,..]`.

1.2 Filter

Remember that `filter` keeps the elements that match the predicate, rather than exclude them.

- Make your own implementation of filter for lists, `myfilter`.

Use `filter` or `myfilter` to solve the following exercises.

- Get a list of numbers below some threshold, from a list of numbers.
- Come up with your own function that returns a bool and use it with `filter`.
- Recreate quicksort.
- Make a function that given a number finds all the positive integers that are a factor of it (evenly divides it).
  - Use it to make a naïve `isPrime` function.

– Make a list of the first 10 primes.

1.3 Fold

To better understand what `foldr` and `foldl` does, try expanding the following expressions by hand, according to the pseudo definitions:

```
foldr f z [x1, x2, ..., xn] == x1 `f` (x2 `f` ... (xn `f` z)...)
```

```
foldl f z [x1, x2, ..., xn] == (...((z `f` x1) `f` x2) `f`...)
`f` xn
```

- `foldr (||) False [False, True, False]`
- `foldl mod 1337 [1166, 86, 43]`
- Make your own fold function for lists, e.g. `myfoldl` (which probably is slightly easier than `myfoldr`).

Now try to use `foldr`, `foldl`, or your own fold function, to solve the following exercises.

- Make your own `sum` function.
- Make a factorial function.
- Make your own `maximum` function, you may use `max`. Hint: assume numbers are $>= 0$.
- Make your own `reverse` function. Hint: use `flip`.
- Make `last` using `foldl1`.

The following is an attempt to make a function `isSorted` that checks if a list is sorted:

```
isSorted :: Num a => [a] -> Bool
isSorted ls = foldr1 (<=) ls`
```

but using it throws an error. Why is that? What should it do instead? Implement it.

For an additional challenge try to implement it in one line (it need only work for lists with at least 2 elements). Hint: One way is to at least use a combination of `foldr, map, zip, tail` and a lambda function. Another solution combines the functionality of the `map, zip` and lambda, in `zipWith`.

- Given the following code, define `foo` to finish the implementation of `myfilter'`:

```
myfilter' :: (a -> Bool) -> [a] -> [a]
myfilter' p l = foldl1 (++) (map foo l)
  where
foo x =
```

2 Modules Beginning

If you haven't already, start reducing the amount of parentheses you use, by instead using function composition and application (the '.' and '$').

2.1 Usage

Make a function that checks whether a string contains all the letters of the English alphabet. Import `Data.Char` to get the function `toLower :: Char -> Char`. For the string `"The quick brown fox jumps over the lazy dog"` it should return `True`.

Make yet another version of quicksort using `partition` from `Data.List`.

Make a `length` function using a fold.

Make an `elem` function using a `map` and a fold. Remake it using `any`.

Use `takeWhile` to find out how many numbers that squared are less than some threshold.

Use `find` and `elemIndices` to find a string with, for example more than 5 e's, in a list of strings.

2.2 (Dis-)Qualified

Complete the previous exercise in a file if you haven't already.

- Write your own `partition` function, and make sure it is not being imported from `Data.List`.
- Import `find` and `elemIndices` explicitly.
  - Make the imports qualified and update your program accordingly.

2.3 Define your own

Export some of your cool new functions as a module. Note: you should use the same name for your file as your module, e.g. you export a module `MyFuncs`, that file should be named `MyFuncs.hs`.

Import and use your functions in another file. If you named your files correctly, you should be able to do something like `import MyFuncs`.