Lecture 14:

HOFs Part II

```
fun map(f: 'a -> 'b, l: 'a list):
                               'b list =
  case l of
      [] => []
    | x::xs => f(x) :: map(f, xs)
```

$$\mathrm{map}(f, [x_1 \ldots x_n]) = [f(x_1), f(x_2) \ldots \boxed{f(x_n)}]$$

```
fun evens(l: int list): int list =
  case l of
    [] => []
  | x::xs => case evenP(x) of
               true => x::evens(xs)
             | false => evens(xs)
```

evens [1,2,3,4]
=
[2,4]

```
fun upper(l: char list): char list =
  case l of
    [] => []
  | x::xs => case Char.isUpper
                  (x) of
               true => x::upper(xs)
             | false => uppr(xs)
```

upper[#"A", #"a", #"B"]

= [#"A", #"B"]

```
(* outputs a list containing all elts x of l such that
                                              p(x) is true *)
fun filter(p: 'a -> bool, l: 'a list) : 'a list
    case l of
        [] => []
      | x::xs => case p(x) of
                     true => x:: filter(     xs)
                   | false => filter(xs)

fun evens(l) = filtr(evenP, l)
fun upper(l) = filter(Char.isUpper, l)
```

Pipeline of HOFs

solve problems by chaining together HOFs

Goal: add 1 to all numbers < 7 in a list, drop all numbers ≥ 7

E.g.

$$\begin{tikzcd}
\text{input} & {[1, 8, 7, 5]} \arrow[d, "\leftarrow \text{filter}(\underline{\quad\quad}, \_)"] \\
\text{step 1} & {[1, 5]} \arrow[d, "\leftarrow \text{map}(\text{fn } x \Rightarrow x+1, \text{ step1})"] \\
\text{Step 2} & {[2, 6]}
\end{tikzcd}$$

```
fun add1Lt7(l: int list): int list =
   let
                                 lt7
      val step1 = filter( fn x => x < 7 , l)
      val step2 = map( fn x => x+1 , step1)
   in
      step2
   end

   fun lt7 (x) = x < 7
```

$$\begin{tikzcd}
& {[1,8,7,5,6]} \arrow[dl, "\text{Step 1: filter}"] \arrow[dr, "\text{step 1: map (add 1)}"] \arrow[dd, bend right=90, "\text{faster?}"'] \arrow[dd, bend left=90, "\text{faster?}"] & \\
{[1,5,6]} \arrow[dr, "\text{step 2: map}"'] & & {[2,9,8,6,7]} \arrow[dl, "\text{Step 2: filter (fn x => x<8)}"] \\
& {[2,6,7]} &
\end{tikzcd}$$

```
fun add1lt7(l) =
  let val step1 = map(fn x => x+1, l)
      val step2 = filter(fn x => x<8, step1)
  in
    step2
  end
```

$\boxed{\text{or}}$

```
filter(fn x => x < 8,
         map(fn x => x+1, l))
```

```
fun sum(l: int list) : int =
    case l of
        [] => 0
      | x::xs => x + sum(xs)
```

sum [1,4,5]

= 10

```
fun join(l: string list) : string =
    case l of
        [] => ""
      | x::xs => x ^ join(xs)
```

join ["a","g"]

=

"ag"

```
fun reduce( c : 'a * 'a -> 'a ,
            n : 'a ,
            l : 'a list ) : 'a =
  case l of
    [] => n
  | x::xs => c(x , reduce(c, n , xs))
fun sum(l) = reduce(fn (x,y) => x+y, 0, l)
fun join(l) = reduce(fn(x,y) => x^y, "", l)
fun max(x:int, y:int) =
    case x<y of
        true => y
      | false => x
```

$\Big]$ Int.max

minInt : int

max(minInt, a) = a
max(a, minInt) = a

```
fun maxAll (l: int list):int =
  reduce (Int.max, minInt, l)
```

maxAll [1, 8, 7, 2]
= 8

Problem:

given a string,
find the number of words in
the longest line.

$$\left.\begin{array}{l} \text{the quick brown fox} \longrightarrow 4 \\ \text{jumped over} \longrightarrow 2 \end{array}\right] 4$$

"the quick brown fox \n jumped over"

"the quick brown fox\n jumped over"

$\downarrow$ divides into lines $\boxed{\text{step 1}}$ $\boxed{\text{lines}}$

["the quick brown fox",
"jumped over"]

$\downarrow$ counting words in each string $\boxed{\text{Step 2}}$ $\boxed{\text{map}}$ $\boxed{\text{wordcount}}$

[4,
2]

$\downarrow$ $\boxed{\text{Step 3}}$ take the max:

4

maxAll

```
(* e.g. "AB\nC\nD" = ["AB", "C", "D"] *)
fun lines(s: string): string list = ...

       words
(* e.g. ("the quick brown fox") =
           ["the", "quick", "brown", "fox"]  *)
fun words(s:string): string list= ...
fun longestline (s: string): int =
  let val step1 = lines(s)
      val step2 = map (fn s => length(words s),
                       step1)
      val step3 = maxAll (step2)
in
    step3
end
```