

```
fun append (xs,ys) =  
  if xs=[]  
  then ys  
  else (hd xs)::append(tl xs,ys)  
  
fun map (f,xs) =  
  case xs of  
    [] => []  
  | x::xs' => (f x)::(map(f,xs'))  
  
val a = map (increment, [4,8,12,16])  
val b = map (hd, [[8,6],[7,5],[3,0,9]])
```

Programming Languages

Dan Grossman
2013

List Functions

Functions over lists

Gain experience with lists and recursion by writing several functions that process and/or produce lists...

Example list functions

```
fun sum_list (xs : int list) =  
  if null xs  
  then 0  
  else hd(xs) + sum_list(tl(xs))  
  
fun countdown (x : int) =  
  if x=0  
  then []  
  else x :: countdown (x-1)  
  
fun append (xs : int list, ys : int list) =  
  if null xs  
  then ys  
  else hd (xs) :: append (tl(xs), ys)
```

Recursion again

Functions over lists are usually recursive

- Only way to “get to all the elements”
- What should the answer be for the empty list?
- What should the answer be for a non-empty list?
 - Typically in terms of the answer for the tail of the list!

Similarly, functions that produce lists of potentially any size will be recursive

- You create a list out of smaller lists

Lists of pairs

Processing lists of pairs requires no new features. Examples:

```
fun sum_pair_list (xs : (int*int) list) =  
  if null xs  
  then 0  
  else #1(hd xs) + #2(hd xs) + sum_pair_list(tl xs)  
  
fun firsts (xs : (int*int) list) =  
  if null xs  
  then []  
  else #1(hd xs) :: firsts(tl xs)  
  
fun seconds (xs : (int*int) list) =  
  if null xs  
  then []  
  else #2(hd xs) :: seconds(tl xs)  
  
fun sum_pair_list2 (xs : (int*int) list) =  
  (sum_list (firsts xs)) + (sum_list (seconds xs))
```