

```
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

Currying Wrapup

More combining functions

- What if you want to curry a tupled function or vice-versa?
- What if a function's arguments are in the wrong order for the partial application you want?

Naturally, it is easy to write higher-order wrapper functions

- And their types are neat logical formulas

```
fun other_curry1 f = fn x => fn y => f y x
fun other_curry2 f x y = f y x
fun curry f x y = f (x,y)
fun uncurry f (x,y) = f x y
```

Efficiency

So which is faster: tupling or currying multiple-arguments?

- They are both constant-time operations, so it doesn't matter in most of your code – “plenty fast”
 - Don't program against an *implementation* until it matters!
- For the small (zero?) part where efficiency matters:
 - It turns out SML/NJ compiles tuples more efficiently
 - But many other functional-language implementations do better with currying (OCaml, F#, Haskell)
 - So currying is the “normal thing” and programmers read `t1 -> t2 -> t3 -> t4` as a 3-argument function that also allows partial application