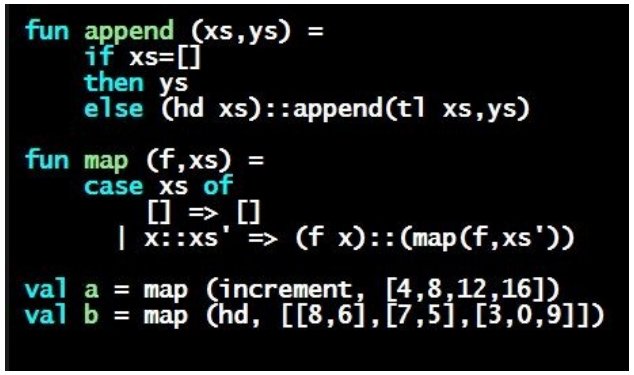

```
fun append (xs,ys) =
    if xs=[]
    then ys
    else (hd xs)::append(tl xs,ys)

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

Generalizing Prior Topics

Generalizing

Our examples of first-class functions so far have all:

- Taken one function as an argument to another function
- Processed a number or a list

But first-class functions are useful anywhere for any kind of data

- Can pass several functions as arguments
- Can put functions in data structures (tuples, lists, etc.)
- Can return functions as results
- Can write higher-order functions that traverse your own data structures

Useful whenever you want to abstract over "what to compute with"

- No new language features

Returning functions

- Remember: Functions are first-class values
  - For example, can return them from functions

- Silly example:

```
fun double_or_triple f =
    if f 7
    then fn x => 2*x
    else fn x => 3*x
```

Has type `(int -> bool) -> (int -> int)`

But the REPL prints `(int -> bool) -> int -> int`
because it never prints unnecessary parentheses and
`t1 -> t2 -> t3 -> t4` means `t1->(t2->(t3->t4))`

Other data structures

- Higher-order functions are not just for numbers and lists
- They work great for common recursive traversals over your own data structures (datatype bindings) too
- Example of a higher-order *predicate*:
  - Are all constants in an arithmetic expression even numbers?
  - Use a more general function of type
    `(int -> bool) * exp -> bool`
  - And call it with **`(fn x => x mod 2 = 0)`**