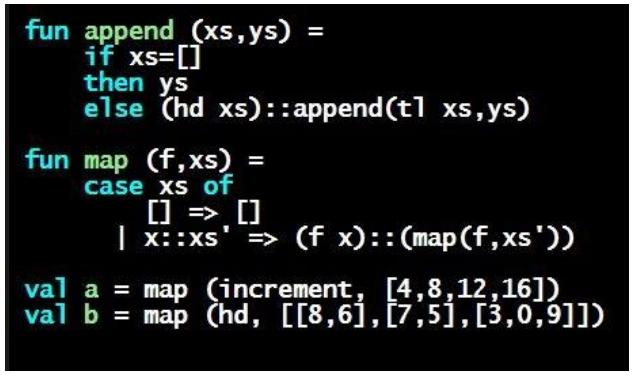

```
fun append (xs,ys) =
    if xs=[]
    then ys
    else (hd xs)::append(tl xs,ys)

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

Ways to Build New Types

How to build bigger types

- Already know:
  - Have various *base types* like **`int bool unit char`**
  - Ways to build (nested) *compound types*: tuples, lists, options
- Coming soon: more ways to build compound types
- First: 3 most important type building-blocks in *any* language
  - "Each of": A **`t`** value contains *values of each of* **`t1 t2 … tn`**
  - "One of": A **`t`** value contains *values of one of* **`t1 t2 … tn`**
  - "Self reference": A **`t`** value can refer to other **`t`** values

  Remarkable: A lot of data can be described with just these building blocks

  Note: These are not the common names for these concepts

Examples

- Tuples build each-of types
  - **`int * bool`** contains an **`int`** *and* a **`bool`**
- Options build one-of types
  - **`int option`** contains an **`int`** *or* it contains no data
- Lists use all three building blocks
  - **`int list`** contains an **`int`** *and* another **`int list`** *or* it contains no data
- And of course we can nest compound types
  - **`((int * int) option * (int list list)) option`**

Coming soon

- Another way to build each-of types in ML
  - *Records*: have named *fields*
  - Connection to tuples and idea of *syntactic sugar*

- A way to build and use our own one-of types in ML
  - For example, a type that contains an **`int`** or a **`string`**
  - Will lead to *pattern-matching*, one of ML's coolest and strangest-to-Java-programmers features

- Later in course: How OOP does one-of types
  - Key contrast with procedural and functional programming