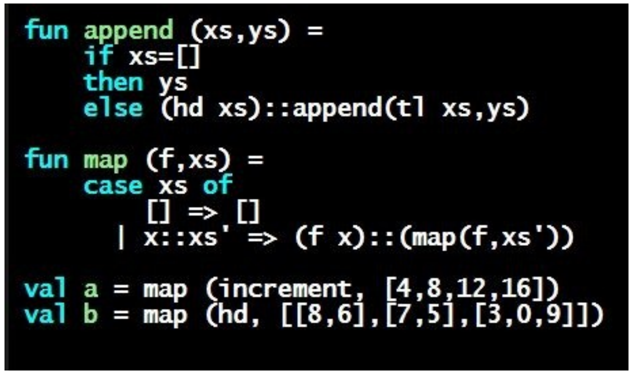

```
fun append (xs,ys) =
    if xs=[]
    then ys
    else (hd xs)::append(tl xs,ys)

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

Records

Records

Record values have fields (any name) holding values

```
{f1 = v1, …, fn = vn}
```

Record types have fields (and name) holding types

```
{f1 : t1, …, fn : tn}
```

The order of fields in a record value or type never matters

- REPL alphabetizes fields just for consistency

Building records:

```
{f1 = e1, …, fn = en}
```

Accessing pieces:

```
#myfieldname e
```

(Evaluation rules and type-checking as expected)

Example

```
{name = “Amelia”, id = 41123 - 12}
```

Evaluates to

```
{id = 41111, name = “Amelia”}
```

And has type

```
{id : int, name : string}
```

If some expression such as a variable **`x`** has this type, then get fields with:

```
#id x      #name x
```

Note we did not have to declare any record types

- The same program could also make a **`{id=true,ego=false}`** of type **`{id:bool,ego:bool}`**

By name vs. by position

- Little difference between **`(4,7,9)`** and **`{f=4,g=7,h=9}`**
  - Tuples a little shorter
  - Records a little easier to remember “what is where”
  - Generally a matter of taste, but for many (6? 8? 12?) fields, a record is usually a better choice

- A common decision for a construct’s syntax is whether to refer to things *by position* (as in tuples) or *by some (field) name* (as with records)
  - A common hybrid is like with Java method arguments (and ML functions as used so far):
    - Caller uses *position*
    - Callee uses *variables*
    - Could do it differently; some languages have