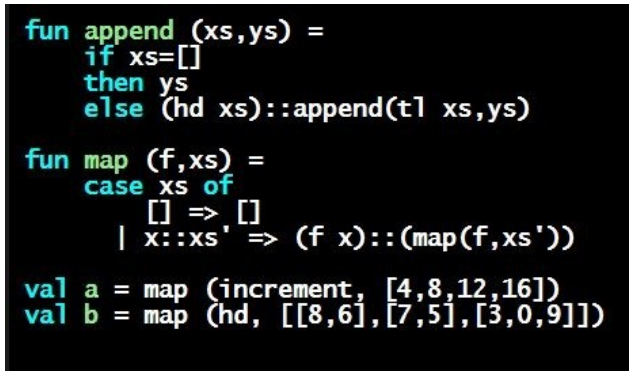

```
fun append (xs,ys) =
    if xs=[]
    then ys
    else (hd xs)::append(tl xs,ys)

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

Functions (Formally)

Function bindings: 3 questions

- Syntax: `fun x0 (x1 : t1, … , xn : tn) = e`
  - (Will generalize in later lecture)

- Evaluation: ***A function is a value!*** (No evaluation yet)
  - Adds **x0** to environment so *later* expressions can *call* it
  - (Function-call semantics will also allow recursion)

- Type-checking:
  - Adds binding `x0 : (t1 * … * tn) -> t` if:
  - Can type-check body `e` to have type `t` in the static environment containing:
    - "Enclosing" static environment (earlier bindings)
    - `x1 : t1, …, xn : tn` (arguments with their types)
    - `x0 : (t1 * … * tn) -> t` (for recursion)

More on type-checking

```
fun x0 (x1 : t1, … , xn : tn) = e
```

- New kind of type: `(t1 * … * tn) -> t`
  - Result type on right
  - The overall type-checking result is to give `x0` this type in rest of program (unlike Java, not for earlier bindings)
  - Arguments can be used only in `e` (unsurprising)

- Because evaluation of a call to `x0` will return result of evaluating `e`, the return type of `x0` is the type of `e`

- The type-checker "magically" figures out `t` if such a `t` exists
  - Later lecture: Requires some cleverness due to recursion
  - More magic after hw1: Later can omit argument types too

Function Calls

A new kind of expression: 3 questions

Syntax: ***`e0 (e1,…,en)`***

- (Will generalize later)
- Parentheses optional if there is exactly one argument

Type-checking:

If:

- **`e0`** has some type **`(t1 * … * tn) -> t`**
- **`e1`** has type **`t1`**, …, **`en`** has type **`tn`**

Then:

- **`e0(e1,…,en)`** has type **`t`**

Example: **`pow(x,y-1)`** in previous example has type **`int`**

Function-calls continued

```
e0(e1,…,en)
```

Evaluation:

1. (Under current dynamic environment,) evaluate **`e0`** to a function **`fun x0 (x1 : t1, … , xn : tn) = e`**
   - Since call type-checked, result *will be* a function

2. (Under current dynamic environment,) evaluate arguments to values **`v1, …, vn`**

3. Result is evaluation of **`e`** in an environment extended to map **`x1`** to **`v1`**, …, **`xn`** to **`vn`**
   - ("An environment" is actually the environment where the function was defined, and includes **`x0`** for recursion)