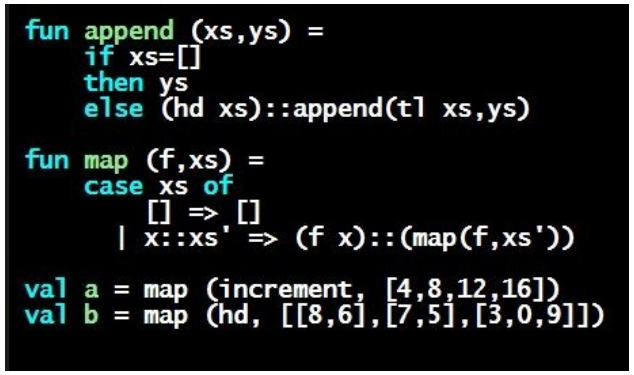

```
fun append (xs,ys) =
    if xs=[]
    then ys
    else (hd xs)::append(tl xs,ys)

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

Anonymous Functions

Toward anonymous functions

- Definitions unnecessarily at top-level are still poor style:

```
fun triple x = 3*x
fun triple_n_times (f,x) = n_times(triple,n,x)
```

- So this is better (but not the best):

```
fun triple_n_times (f,x) =
  let fun trip y = 3*y
  in
     n_times(trip,n,x)
  end
```

- And this is even smaller scope
  - It makes sense but looks weird (poor style; see next slide)

```
fun triple_n_times (f,x) =
  n_times(let fun trip y = 3*y in trip end, n, x)
```

Anonymous functions

- This does not work: A function *binding* is not an *expression*

```
fun triple_n_times (f,x) =
  n_times((fun trip y = 3*y), n, x)
```

- This is the best way we were building up to: an expression form for *anonymous functions*

```
fun triple_n_times (f,x) =
  n_times((fn y => 3*y), n, x)
```

  - Like all expression forms, can appear anywhere
  - Syntax:
    - **`fn`** not **`fun`**
    - **`=>`** not **`=`**
    - no function name, just an argument pattern

Using anonymous functions

- Most common use: Argument to a higher-order function
  - Don't need a name just to pass a function

- But: Cannot use an anonymous function for a recursive function
  - Because there is no name for making recursive calls
  - If not for recursion, **`fun`** bindings would be syntactic sugar for **`val`** bindings and anonymous functions

```
fun triple x = 3*x

val triple = fn y => 3*y
```