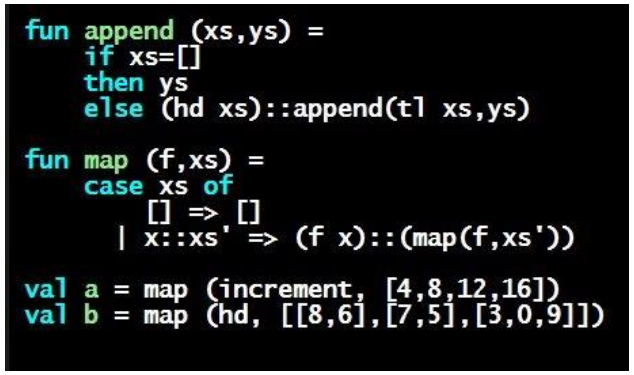

```
fun append (xs,ys) =
    if xs=[]
    then ys
    else (hd xs)::append(tl xs,ys)

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

Why Lexical Scope

Why lexical scope

- *Lexical scope*: use environment where function is defined

- *Dynamic scope*: use environment where function is called

Decades ago, both might have been considered reasonable, but now we know lexical scope makes much more sense

Here are three precise, technical reasons

  – Not a matter of opinion

Why lexical scope?

1. Function meaning does not depend on variable names used

Example: Can change body of **`f`** to use **`q`** everywhere instead of **`x`**

- Lexical scope: it cannot matter
- Dynamic scope: depends how result is used

```
fun f y =
    let val x = y+1
    in fn z => x+y+z end
```

Example: Can remove unused variables

- Dynamic scope: but maybe some **`g`** uses it (weird)

```
fun f g =
    let val x = 3
    in g 2 end
```

Why lexical scope?

2. Functions can be type-checked and reasoned about where defined

Example: Dynamic scope tries to add a string and an unbound variable to **6**

```
val x = 1
fun f y =
    let val x = y+1
    in fn z => x+y+z end
val x = "hi"
val g = f 7
val z = g 4
```

Why lexical scope?

3. Closures can easily store the data they need
   - Many more examples and idioms to come

```
fun greaterThanX x = fn y => y > x

fun filter (f,xs) =
    case xs of
      [] => []
    | x::xs => if f x
               then x::(filter(f,xs))
               else filter(f,xs)

fun noNegatives xs = filter(greaterThanX ~1, xs)
fun allGreater (xs,n) = filter(fn x => x > n, xs)
```

Does dynamic scope exist?

- Lexical scope for variables is definitely the right default
  - Very common across languages

- Dynamic scope is occasionally convenient in some situations
  - So some languages (e.g., Racket) have special ways to do it
  - But most do not bother

- If you squint some, exception handling is more like dynamic scope:
  - **`raise e`** transfers control to the current innermost handler
  - Does not have to be syntactically inside a handle expression (and usually is not)