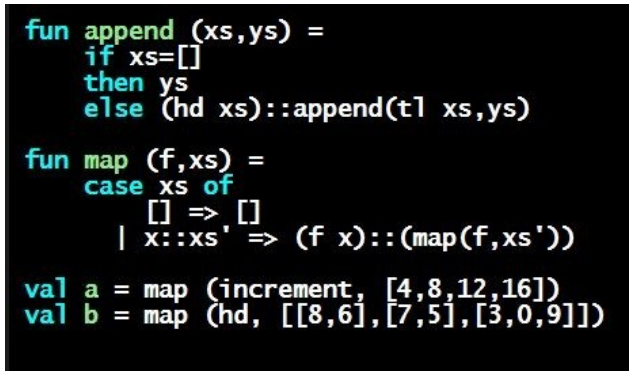

```
fun append (xs,ys) =
    if xs=[]
    then ys
    else (hd xs)::append(tl xs,ys)

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

What is Type Inference?

Type-checking

- (Static) type-checking can reject a program before it runs to prevent the possibility of some errors
  - A feature of statically typed languages
- Dynamically typed languages do little (none?) such checking
  - So might try to treat a number as a function at run-time
- Will study relative advantages after some Racket
  - Racket, Ruby (and Python, Javascript, …) dynamically typed
- ML (and Java, C#, Scala, C, C++) is statically typed
  - Every binding has one type, determined "at compile-time"

Implicitly typed

- ML is statically typed
- ML is implicitly typed: rarely need to write down types

```
fun f x = (* infer val f : int -> int *)
    if x > 3
    then 42
    else x * 2

fun g x = (* report type error *)
    if x > 3
    then true
    else x * 2
```

- Statically typed: Much more like Java than Javascript!

Type inference

- Type inference problem: Give every binding/expression a type such that type-checking succeeds
  - Fail if and only if no solution exists

- In principle, could be a pass before the type-checker
  - But often implemented together

- Type inference can be easy, difficult, or *impossible*
  - Easy: Accept all programs
  - Easy: Reject all programs
  - Subtle, elegant, and *not magic*: ML