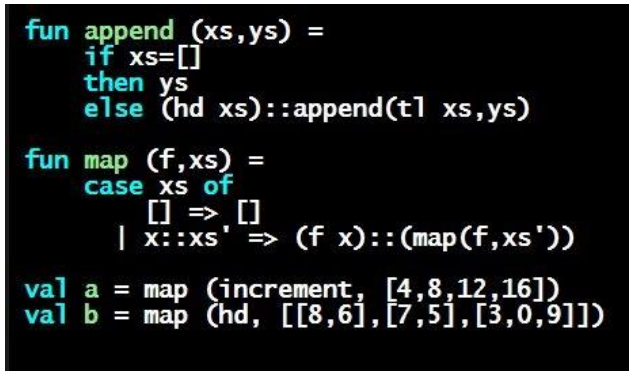

```
fun append (xs,ys) =
    if xs=[]
    then ys
    else (hd xs)::append(tl xs,ys)

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
University of Washington

Part B Wrap-Up

&

Parts C Preview

Course Overview So Far

We decided after recording this video to move Static vs. Dynamic Typing to its own "Section 7"

Part A:

1. Basics, functions, recursion, scope, variables, tuples, lists, …
2. Datatypes, pattern-matching, tail recursion
3. First-class functions, closures [and course motivation!]
4. Type inference, modules, equivalence

Part B:

5. Dynamic types, parentheses, delayed evaluation, streams, macros
6. Structs, interpreters, closures, static checking, static vs. dynamic

Part C

7. Dynamically-typed Object-Oriented Programming
   Ruby basics, arrays, blocks, classes, methods, much more, …

8. OOP vs. Functional decomposition
   Advanced OOP topics (e.g., mixins, double dispatch)
   Generics vs. Subtyping

+ Exam focusing on Part B and C

We decided after recording this video to reorganize the material in Part C into 3 sections:

- What this video calls Section 7 is now called Section 8
- What this video calls Section 8 is now divided into Section 9 and Section 10

The content is the same.

"Finishing the story"

Highly recommended to continue even if you "already know OOP"

- The (whole) point is to study OOP topics in an analogous way *and* compare/contrast with FP
- Some with OOP background find 7 "less interesting" but stay tuned for 8, which builds on 7
  - Also Homework 6 appeals to different experience levels
- Some find Part C "anti-OOP", which is mostly ☺ not true
- Will learn more about FP in Part C…

"Finishing the story"

- Implementing OOP's dynamic dispatch as an idiom in Racket
- FP vs. OOP decomposition is a big "aha" moment
- Homework 7 involves porting ML to Ruby while changing the decomposition
  - It's also another "interpreter" but for a "geometry language" without closures
- ML's type variables relation to (Java) generics and subtyping

So… congratulations on Part B and see you in Part C! ☺