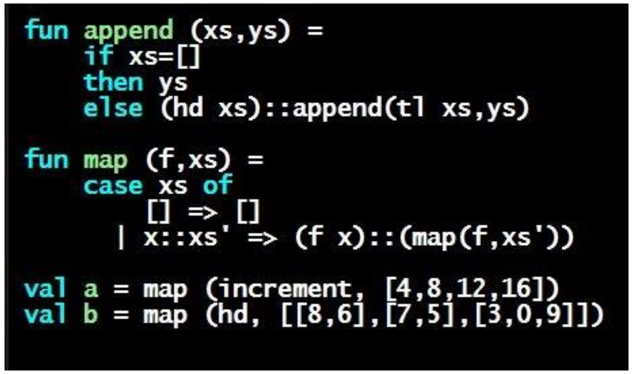

```
fun append (xs,ys) =
    if xs=[]
    then ys
    else (hd xs)::append(tl xs,ys)

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

Optional: Very High-Level Outline

Course Overview

- Very hard to describe topics we haven't done yet (!)
  - So don't be intimidated and maybe skip this to "dive in"

Part A:

0. Software Installation
1. Basics, functions, recursion, scope, variables, tuples, lists, …
   - Give extra time for Section 1
2. Datatypes, pattern-matching, tail recursion
3. First-class functions, closures [and course motivation!]
4. Type inference, modules

Overall: A *precisely specified* introduction to functional programming built up piece-by-piece

Part B

5. Quick “re-do” in a dynamically typed language;
   Delaying evaluation

6. Implementing languages with interpreters;
   Static vs. dynamic typing

Part C

7. Dynamically-typed Object-Oriented Programming

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

“Finishes the story” even if you “already know OOP”

- Some with OOP background find 7 “less interesting” but stay tuned for 8
- Some find Part C “anti-OOP”, which is mostly ☺ not true
    - About contrasting with what many “already think/know”
    - And yes, some focus on where commitment to “pure OOP” *may* be unwise even in “an OOP language”