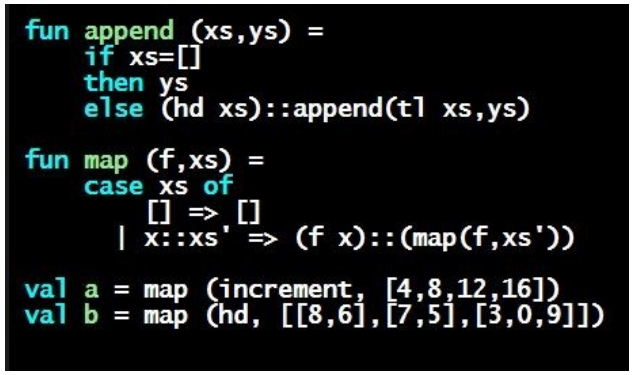

```
fun append (xs,ys) =
    if xs=[]
    then ys
    else (hd xs)::append(tl xs,ys)

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

[Placeholder for] Course Motivation

What this course is about

- Many essential concepts relevant in any programming language
  - And how these pieces fit together
- Use ML, Racket, and Ruby languages:
  - They let many of the concepts "shine"
  - Using multiple languages shows how the same concept can "look different" or actually be slightly different
  - In many ways simpler than Java, C#, Python, …
- Big focus on *functional programming*
  - Not using *mutation* (assignment statements) (!)
  - Using *first-class functions* (can't explain that yet)
  - But many other topics too

Assumed background

- *Not* an introductory programming course
  - Assume you have at least 1-2 programming courses

- *Not* an advanced course on programming langauges
  - Won't assume you are an experienced programmer

Somewhere in the middle…

Things I assume you know (a little)

- Variables, conditionals (if), loops, arrays
- Recursion (okay if lack a little confidence for now)
- Implementation vs. interface (abstraction, modularity)
  - Possibly/probably using objects-oriented programming
- Basic data structures: linked lists, binary trees
- Dynamic-dispatch
  - Also known as method overriding, subclassing, …
  - But not needed for first 2/3 of course and then will review

Any particular language

- Occasionally compare to Java in optional videos (not on homework)
  - If know C#, can surely follow right along
- Will more rarely compare to C in optional videos (not on homework)
  - Can be very useful if you understand some C (or if you learn C later)
- It is okay if you mostly know Python or Javascript or…
  - What matters are the concepts on the previous slide

Really will "start programming over from the beginning"

- Moving way too fast unless you have programmed some before

Why learn this?

This is the “normal” place for course motivation

- Why learn this material?

But in my experience, we don’t have enough shared vocabulary

- So 2-week delay on motivation for functional programming
- I promise full motivation: delay is worth it
- (Will motivate immutable data at end of section 1)

My claim

Learning to think about software in this "PL" way will make you a better programmer even if/when you go back to old ways

It will also give you the mental tools and experience you need for a lifetime of confidently picking up new languages and ideas

[Somewhat in the style of *The Karate Kid* movies (1984, 2010)

- http://www.imdb.com/title/tt0087538/
- http://www.imdb.com/title/tt1155076/

]

A strange environment

- Next 4-5 weeks will use
  - ML language
  - Emacs editor
  - Read-eval-print-loop (REPL) for evaluating programs
- *You* need to get things installed and configured
  - See written instructions (read carefully; feedback welcome)
  - Optional: videos showing Windows installation
- Only then can you focus on the content of Homework 1
- Working in strange environments is a CS life skill

Enough text already

- Subsequent lectures will write code
  - Plus switch to slides for key concepts
  - Plus “in-video questions”

- Much better than these “introduction” videos