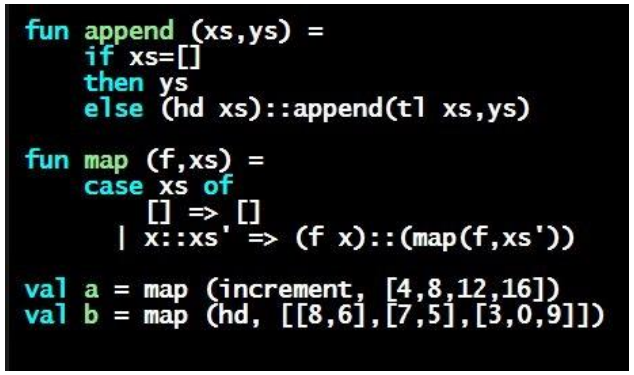

```
fun append (xs,ys) =
    if xs=[]
    then ys
    else (hd xs)::append(tl xs,ys)

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

Introduction to Ruby

# *Ruby logistics*

- Next two sections use the Ruby language
  - http://www.ruby-lang.org/
  - Installation / basic usage instructions on course website
    - Emacs in lectures, but many editors support Ruby well
    - We support a few versions; differences not so relevant

- Excellent documentation available, much of it free
  - So may not cover every language detail in course materials
  - http://ruby-doc.org/
  - http://www.ruby-lang.org/en/documentation/
  - Particularly recommend “Programming Ruby 1.9 & 2.0, The Pragmatic Programmers’ Guide”
    - Not required and not free

# *Ruby: Our focus*

- *Pure object-oriented*: *all* values are *objects* (even numbers)
- *Class-based*: Every object has a class that determines behavior
  - Like Java, unlike Javascript
  - *Mixins* (neither Java interfaces nor C++ multiple inheritance)
- *Dynamically typed*
- Convenient *reflection*: Run-time inspection of objects
- Very *dynamic*: Can change classes during execution
- *Blocks* and libraries encourage lots of closure idioms
- Syntax, scoping rules, semantics of a "*scripting language*"
  - Variables "spring to life" on use
  - Very flexible arrays

# *Ruby: Not our focus*

- Lots of support for string manipulation and regular expressions
- Popular for server-side web applications
  - Ruby on Rails
- Often many ways to do the same thing
  - More of a “why not add that too?” approach

# *Where Ruby fits*

| | dynamically typed | statically typed |
|---|---|---|
| functional | Racket | SML |
| object-oriented (OOP) | Ruby | Java, etc. |

Note: Racket also has classes and objects when you want them

- In Ruby everything uses them (at least implicitly)

Historical note: *Smalltalk* also a dynamically typed, class-based, pure OOP language with blocks and convenient reflection

- Smaller just-as-powerful language
- Ruby less simple, more “modern and useful”

Dynamically typed OOP helps identify OOP's essence by not having to discuss types

# *A note on the homework*

Next homework is about understanding and extending an *existing* program in an *unfamiliar* language

- Good practice
- Quite different feel than previous homeworks
- *Read* code: determine what you do and do not (!) need to understand

Homework requires the Tk graphics library to be installed such that the provided Ruby code can use it

# *Getting started*

- See **`.rb`** file for our first program
  - (There are much shorter ways to write the same thing)

- Can run file **`foo.rb`** at the command-line with **`ruby foo.rb`**

- Or can use **`irb`**, which is a REPL
  - Run file **`foo.rb`** with **`load "foo.rb"`**