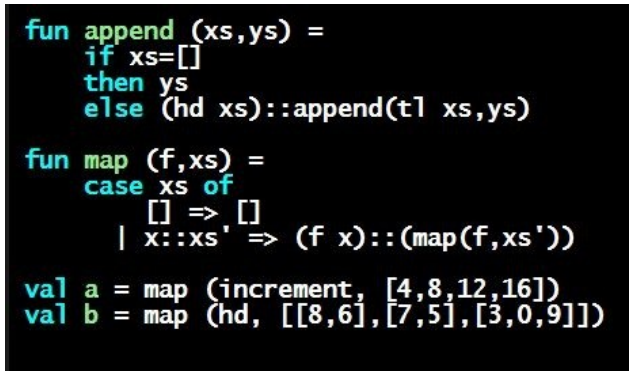

```
fun append (xs,ys) =
    if xs=[]
    then ys
    else (hd xs)::append(tl xs,ys)

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

Passing Blocks

Blocks

Blocks are probably Ruby's strangest feature compared to other PLs

But *almost* just closures

- Normal: easy way to pass anonymous functions to methods for all the usual reasons
- Normal: Blocks can take 0 or more arguments
- Normal: Blocks use lexical scope: block body uses environment where block was defined

Exam

```
3.times { puts "hi" }
[4,6,8].each { puts "hi" }
i = 7
[4,6,8].each {|x| if i > x then puts (x+1) end }
```

Some strange things

- Can pass 0 or 1 block with *any* message
  - Callee might ignore it
  - Callee might give an error if you do not send one
  - Callee might do different things if you do/don't send one
    - Also number-of-block-arguments can matter

- Just put the block "next to" the "other" arguments (if any)
  - Syntax: `{e}`, `{|x| e}`, `{|x,y| e}`, etc. (plus variations)
    - Can also replace `{` and `}` with **`do`** and **`end`**
      - Often preferred for blocks > 1 line

Blocks everywhere

- Rampant use of great block-taking methods in standard libraray
- Ruby has loops but very rarely used
  - Can write `(0..i).each {|j| e}`, but often better options
- Examples (consult documentation for many more)

```
a = Array.new(5) {|i| 4*(i+1)}
a.each { puts "hi" }
a.each {|x| puts (x * 2) }
a.map  {|x| x * 2 } #synonym: collect
a.any? {|x| x > 7 }
a.all? {|x| x > 7 }
a.inject(0) {|acc,elt| acc+elt }
a.select {|x| x > 7 } #non-synonym: filter
```