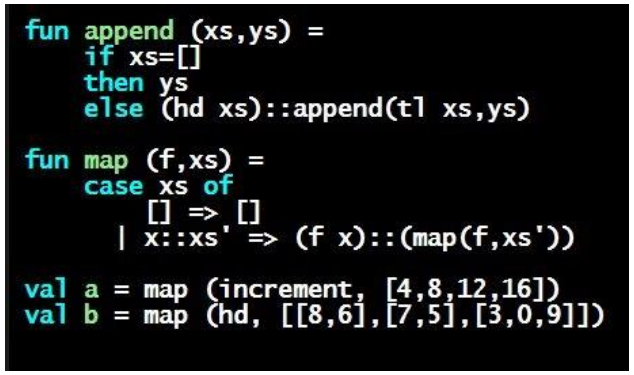

```
fun append (xs,ys) =
    if xs=[]
    then ys
    else (hd xs)::append(tl xs,ys)

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

Racket Definitions, Functions, Conditionals

Example

```
#lang racket

(define x 3)
(define y (+ x 2))

(define cube ; function
  (lambda (x)
    (* x (* x x))))

(define pow ; recursive function
  (lambda (x y)
    (if (= y 0)
        1
        (* x (pow x (- y 1))))))
```

Some niceties

Many built-in functions (a.k.a. procedures) take any number of args

- Yes `*` is just a function
- Yes you can define your own *variable-arity* functions (not shown here)

```
(define cube
  (lambda (x)
    (* x x x)))
```

Better style for non-anonymous function definitions (just sugar):

```
(define (cube x)
  (* x x x))

(define (pow x y)
    (if (= y 0)
        1
        (* x (pow x (- y 1)))))
```

An old friend: currying

Currying is an idiom that works in any language with closures

- Less common in Racket because it has real multiple args

```
(define pow
  (lambda (x)
    (lambda (y)
      (if (= y 0)
          1
          (* x ((pow x) (- y 1)))))))

(define three-to-the (pow 3))
(define eightyone (three-to-the 4))
(define sixteen ((pow 2) 4))
```

Sugar for defining curried functions: `(define ((pow x) y) (if …`

(No sugar for calling curried functions)