

```
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

Different Modules Define Different Types

# *Can't mix-and-match module bindings*

Modules with the *same signatures* still define *different types*

So things like this do not type-check:

- `Rational1.toString(Rational2.make_frac(9,6))`
- `Rational3.toString(Rational2.make_frac(9,6))`

This is a crucial feature for type system and module properties:

- Different modules have different internal invariants!
- In fact, they have different type definitions
  - `Rational1.rational` looks like `Rational2.rational`, but clients and the type-checker do not know that
  - `Rational3.rational` is `int*int` not a datatype!