

```
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

Pattern-Matching So Far: Precisely

Careful definitions

When a language construct is “new and strange,” there is *more* reason to define the evaluation rules precisely...

- ... so let's review datatype bindings and case expressions “so far”
 - *Extensions* to come but won't invalidate the “so far”

Datatype bindings

```
datatype t = C1 of t1 | C2 of t2 | ... | Cn of tn
```

Adds type t and constructors C_i of type $t_i \rightarrow t$

- $C_i \ v$ is a value, i.e., the result “includes the tag”

Omit “of t ” for constructors that are just tags, no underlying data

- Such a C_i is a value of type t

Given an expression of type t , use *case expressions* to:

- See which variant (tag) it has
- Extract underlying data once you know which variant

Datatype bindings

```
case e of p1 => e1 | p2 => e2 | ... | pn => en
```

- As usual, can use a case expressions anywhere an expression goes
 - Does not need to be whole function body, but often is
- Evaluate **e** to a value, call it **v**
- If **p_i** is the first *pattern* to *match* **v**, then result is evaluation of **e_i** in environment “extended by the match”
- Pattern **C_i (x₁ , ... , x_n)** matches value **C_i (v₁ , ... , v_n)** and extends the environment with **x₁** to **v₁** ... **x_n** to **v_n**
 - For “no data” constructors, pattern **C_i** matches value **C_i**