```
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

Datatype Bindings

Datatype bindings

A “strange” (?) and totally awesome (!) way to make one-of types:

- A **`datatype`** binding

```
datatype mytype = TwoInts of int * int
                | Str of string
                | Pizza
```

- Adds a new type **`mytype`** to the environment
- Adds *constructors* to the environment: **`TwoInts`**, **`Str`**, and **`Pizza`**
- A constructor is (among other things), a function that makes values of the new type (or is a value of the new type):
  - **`TwoInts : int * int -> mytype`**
  - **`Str : string -> mytype`**
  - **`Pizza : mytype`**

The values we make

```
datatype mytype = TwoInts of int * int
                | Str of string
                | Pizza
```

- Any value of type **`mytype`** is made from *one of* the constructors
- The value contains:
  - A "tag" for "which constructor" (e.g., **`TwoInts`**)
  - The corresponding data (e.g., **`(7,9)`**)
- Examples:
  - **`TwoInts(3+4,5+4)`** evaluates to **`TwoInts(7,9)`**
  - **`Str(if true then "hi" else "bye")`** evaluates to **`Str("hi")`**
  - **`Pizza`** is a value

Using them

So we know how to *build* datatype values; need to *access* them

There are *two* aspects to accessing a datatype value

1. Check what *variant* it is (what constructor made it)
2. Extract the *data* (if that variant has any)

Notice how our other one-of types used functions for this:

- **`null`** and **`isSome`** check variants
- **`hd`**, **`tl`**, and **`valOf`** extract data (raise exception on wrong variant)

ML *could* have done the same for datatype bindings

- For example, functions like "**`isStr`**" and "**`getStrData`**"
- Instead it did something better