```
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

Useful Datatypes

Useful examples

Let's fix the fact that our only example datatype so far was silly…

- Enumerations, including carrying other data

```
datatype suit = Club | Diamond | Heart | Spade
datatype rank = Jack | Queen | King
              | Ace | Num of int
```

- Alternate ways of identifying real-world things/people

```
datatype id = StudentNum of int
            | Name of string
                    * (string option)
                    * string
```

Don't do this

Unfortunately, bad training and languages that make one-of types inconvenient lead to common *bad style* where each-of types are used where one-of types are the right tool

```
(* use the student_num and ignore other
   fields unless the student_num is ~1 *)
{ student_num : int,
  first       : string,
  middle      : string option,
  last        : string }
```

- Approach gives up all the benefits of the language enforcing every value is one variant, you don't forget branches, etc.
- And it makes it less clear what you are doing

That said…

But if instead, the point is that every "person" in your program has a name and maybe a student number, then each-of is the way to go:

```
{ student_num : int option,
  first       : string,
  middle      : string option,
  last        : string }
```

Expression Trees

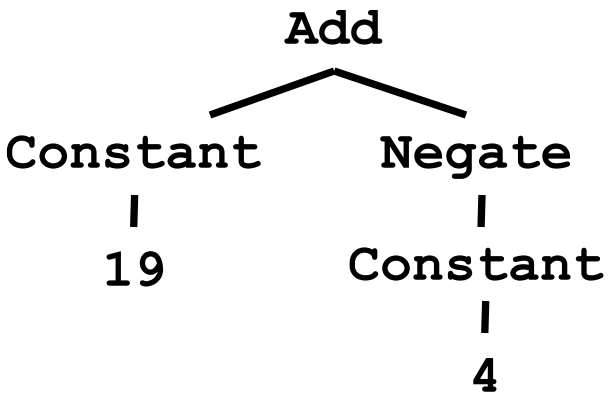
A more exciting (?) example of a datatype, using self-reference

```
datatype exp = Constant of int
             | Negate   of exp
             | Add      of exp * exp
             | Multiply of exp * exp
```

An expression in ML of type **`exp`**:

```
Add (Constant (10+9), Negate (Constant 4))
```

How to picture the resulting value in your head:



Recursion

Not surprising:

Functions over recursive datatypes are usually recursive

```
fun eval e =
    case e of
         Constant i       => i
       | Negate e2        => ~ (eval e2)
       | Add(e1,e2)       => (eval e1) + (eval e2)
       | Multiply(e1,e2)  => (eval e1) * (eval e2)
```