

```
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
2013

Case Expressions

Case

ML combines the two aspects of accessing a one-of value with a *case expression* and *pattern-matching*

- Pattern-matching much more general/powerful (soon!)

Example:

```
fun f x = (* f has type mytype -> int *)
  case x of
    Pizza => 3
  | TwoInts(i1,i2) => i1+i2
  | Str s => String.size s
```

- A multi-branch conditional to pick branch based on variant
- Extracts data and binds to variables local to that branch
- Type-checking: all branches must have same type
- Evaluation: evaluate between case ... of and the right branch

Patterns

In general the syntax is:

```
case e0 of
  p1 => e1
  | p2 => e2
  ...
  | pn => en
```

For today, each *pattern* is a constructor name followed by the right number of variables (i.e., `C` or `C x` or `C (x, y)` or ...)

- Syntactically most patterns (all today) look like expressions
- But patterns are not expressions
 - We do not evaluate them
 - We see if the result of `e0` *matches* them

Why this way is better

0. You can use pattern-matching to write your own testing and data-extractions functions if you must
 - But do not do that on your homework
1. You cannot forget a case (inexhaustive pattern-match warning)
2. You cannot duplicate a case (a type-checking error)
3. You will not forget to test the variant correctly and get an exception (like `hd []`)
4. Pattern-matching can be generalized and made more powerful, leading to elegant and concise code