

```
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

Mutable References

ML has (separate) mutation

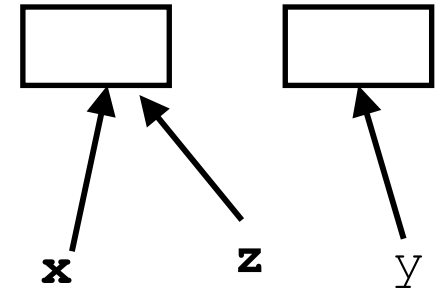
- Mutable data structures are okay in some situations
 - When “update to state of world” is appropriate model
 - But want most language constructs truly immutable
- ML does this with a separate construct: references
- Introducing now because will use them for next closure idiom
- Do not use references on your homework
 - You need practice with mutation-free programming
 - They will lead to less elegant solutions

References

- New types: **t ref** where **t** is a type
- New expressions:
 - **ref e** to create a reference with initial contents **e**
 - **e1 := e2** to update contents
 - **!e** to retrieve contents (not negation)

References example

```
val x = ref 42
val y = ref 42
val z = x
val _ = x := 43
val w = (!y) + (!z) (* 85 *)
(* x + 1 does not type-check *)
```



- A variable bound to a reference (e.g., **x**) is still immutable: it will always refer to the same reference
- But the contents of the reference may change via `:=`
- And there may be aliases to the reference, which matter a lot
- References are first-class values
- Like a one-field mutable object, so `:=` and `!` don't specify the field