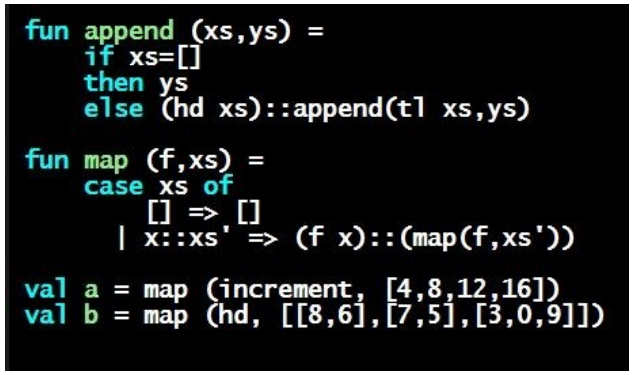

```
fun append (xs,ys) =
    if xs=[]
    then ys
    else (hd xs)::append(tl xs,ys)

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

Delay and Force

Best of both worlds

Assuming some expensive computation has no side effects, ideally we would:

- Not compute it *until needed*
- *Remember the answer* so future uses complete immediately

Called *lazy evaluation*

Delay and force

```
(define (my-delay th)
  (mcons #f th))

(define (my-force p)
      (if (mcar p)
       (mcdr p)
          (begin (set-mcar! p #t)
                 (set-mcdr! p ((mcdr p)))
                 (mcdr p))))
```

An ADT represented by a mutable pair

- `#f` in *car* means *cdr* is unevaluated thunk
  - Really a one-of type: thunk or result-of-thunk
- Ideally hide representation in a module

Using promises

```
(define (f p)
  (… (if (…) 0 (… (my-force p) …))
     (if (…) 0 (… (my-force p) …))
     …
     (if (…) 0 (… (my-force p) …))))
```

```
(f (my-delay (lambda () e)))
```

Lessons From Example

See code file for example that does multiplication using a very slow addition helper function

- With thunking second argument:
  - *Great* if first argument 0
  - Okay if first argument 1
  - *Worse* otherwise
- With precomputing second argument:
  - *Okay* in all cases
- With thunk that uses a promise for second argument:
  - *Great* if first argument 0
  - *Okay* otherwise