

```
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

*Optional:* Racket Macros with **define-syntax**

# *Example Racket macro definitions*

Two simple macros

```
(define-syntax my-if                ; macro name
  (syntax-rules (then else)        ; other keywords
    [(my-if e1 then e2 else e3)    ; macro use
     (if e1 e2 e3)]))              ; form of expansion
```

```
(define-syntax comment-out          ; macro name
  (syntax-rules ()                  ; other keywords
    [(comment-out ignore instead)   ; macro use
     instead]))                     ; form of expansion
```

If the form of the use matches, do the corresponding expansion

- In these examples, list of possible use forms has length 1
- Else syntax error

# Revisiting delay and force

Recall our definition of promises from earlier

- Should we use a macro instead to avoid clients' explicit thunk?

```
(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))))
```

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

```
(define (f p)
  (... (my-force p) ...))
```

# *A delay macro*

- A macro can put an expression under a thunk
  - Delays evaluation without explicit thunk
  - Cannot implement this with a function
- Now client should *not* use a thunk (that would double-thunk)
  - Racket's pre-defined `delay` is a similar macro

```
(define-syntax my-delay
  (syntax-rules ()
    [ (my-delay e)
      (mcons #f (lambda () e)) ]))
```

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

# *What about a force macro?*

We could define `my-force` with a macro too

- Good macro style would be to evaluate the argument exactly once (use `x` below, not multiple evaluations of `e`)
- Which shows it is *bad style to use a macro at all here!*
- *Do not use macros when functions do what you want*

```
(define-syntax my-force
  (syntax-rules ()
    [ (my-force e)
      (let ([x e])
        (if (mcar x)
            (mcdr x)
            (begin (set-mcar! x #t)
                    (set-mcdr! p ((mcdr p)))
                    (mcdr p))))]))
```