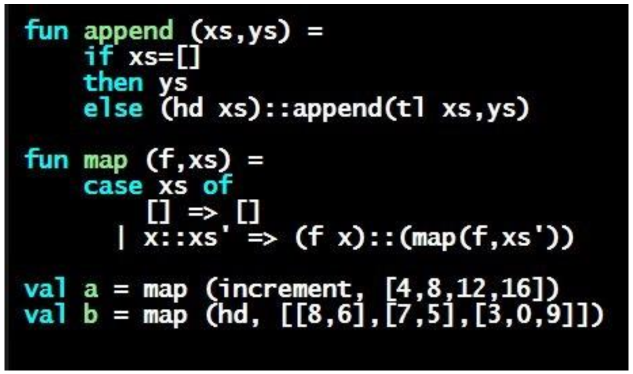

```
fun append (xs,ys) =
    if xs=[]
    then ys
    else (hd xs)::append(tl xs,ys)

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

Macros: The Key Points

Rest of Section

- This segment: High-level idea of macros
  - Not needed for this section's homework
  - Needed to build on a little in next section
- Then: Racket's macro system and pitfalls of macros
  - Defining a macro is this section's homework challenge
  - "Macros done right" a key feature of Racket compared to other macro systems (e.g., C/C++)
- Segments *after* this one optional
  - Conserving time and will not build on it

What is a macro

- A *macro definition* describes how to transform some new syntax into different syntax in the source language
- A macro is one way to implement syntactic sugar
  - "Replace any syntax of the form **`e1 andalso e2`** with **`if e1 then e2 else false`**"
- A *macro system* is a language (or part of a larger language) for defining macros
- *Macro expansion* is the process of rewriting the syntax for each *macro use*
  - Before a program is run (or even compiled)

Using Racket Macros

- If you define a macro **`m`** in Racket, then **`m`** becomes a new special form:
  - Use **`(m …)`** gets expanded according to definition

- Example definitions (actual definitions in optional segment):
  - Expand **`(my-if e1 then e2 else e3)`**

    to **`(if e1 e2 e3)`**
  - Expand **`(comment-out e1 e2)`**

    to **`e2`**
  - Expand **`(my-delay e)`**

    to **`(mcons #f (lambda () e))`**

Example uses

It is like we added keywords to our language

- Other keywords only keywords in uses of that macro
- Syntax error if keywords misused
- Rewriting ("expansion") happens before execution

```
(my-if x then y else z)   ; (if x y z)
(my-if x then y then z)   ; syntax error

(comment-out (car null) #f)

(my-delay (begin (print "hi") (foo 15)))
```

Overuse

Macros often deserve a bad reputation because they are often overused or used when functions would be better

When in doubt, resist defining a macro?

But they can be used well and the optional material should help

Optional stuff ahead

- How any macro system must deal with tokens, parentheses, and scope
- How to define macros in Racket
- How macro definitions must deal with expression evaluation carefully
  - Order expressions evaluate and how many times
- The key issue of variable bindings in macros and the notion of *hygiene*
  - Racket is superior to most languages here