61A Extra Lecture 8
Announcements
Homoiconicity
A Scheme Expression is a Scheme List

Scheme programs consist of expressions, which can be:

- Primitive expressions: 2 3.3 true + quotient
- Combinations: (quotient 10 2) (not true)

The built-in Scheme list data structure (which is a linked list) can represent combinations

```scheme
(scm> (list 'quotient 10 2))
(quotient 10 2)

(scm> (eval (list 'quotient 10 2)))
5
```

In such a language, it is straightforward to write a program that writes a program

(Demo)
Homoiconic Languages

Languages have both a concrete syntax and an abstract syntax

(Python Demo)

A language is *homoiconic* if the abstract syntax can be read from the concrete syntax

(Scheme Demo)

Quotation is actually a combination in disguise

(Quote Demo)
Macros
Macros Perform Code Transformations

A macro is an operation performed on the source code of a program before evaluation.

Macros exist in many languages, but are easiest to define correctly in homoiconic languages.

Scheme has a `define-macro` special form that defines a source code transformation:

```
(define-macro (twice expr)
  (list 'begin expr expr))
```

Example:

```
> (twice (print 2))
2
2
```

Evaluation procedure of a macro call expression:

- Evaluate the operator sub-expression, which evaluates to a macro.
- Call the macro procedure on the operand expressions \textit{without evaluating them first}.
- Evaluate the expression returned from the macro procedure.

(Demo)
Problem 1

Define a macro that evaluates an expression for each value in a sequence

```
(define (map fn vals)
  (if (null? vals)
      ()
      (cons (fn (car vals))
            (map fn (cdr vals))))

scm> (map (lambda (x) (* x x)) '(2 3 4 5))
(4 9 16 25)

(define-macro (for sym expr vals)
  (list 'map ________________________
        (list 'lambda (list sym) expr) vals)
_______________________________)

scm> (for x (* x x) '(2 3 4 5))
(4 9 16 25)
```
Quasi-Quoting

(Demo)
Variable-Length Parameter Lists

(Demo)
Problem 2

Define a function `nest` that builds a nested list containing its arguments

```scheme
(define (nest first . rest)
  (if (null? rest)
      (list first)
      (list first (apply nest rest))))

scm> (nest 3)
(3)

scm> (nest 3 4 5 6)
(3 (4 (5 (6) )))
```
Temporary Symbols

(Demo)