

MASSACHUSETTS INSTITUTE OF TECHNOLOGY  
Department of Electrical Engineering and Computer Science  
6.001—Structure and Interpretation of Computer Programs  
Spring 2004

**Recitation 1**  
**Introduction**

## Glossary

**program** collection of procedures and static data that accomplishes a specific task.

**procedure** a piece of code that when called with arguments computes and returns a result; possibly with some side-effects. In scheme, procedures are normal values like numbers.

**function** see procedure; they're equivalent in scheme. Some other languages make a distinction.

**argument** An input variable to a procedure. A new version of the variable is created every time the procedure is called.

**parameter** see argument.

**expression** A single valid scheme statement. `5`, `(+ 3 4)`, and `(if (lambda (x) x) 5 (+ 3 4))` are expressions.

**value** The result of evaluating an expression. `5`, `7`, and `5` respectively.

**type** Values are classified into types. Some types: numbers, booleans, strings, lists, and procedures. Generally, types are disjoint (any value falls into exactly one type class).

**call** Verb, the action of invoking, jumping to, or using a procedure.

**apply** Calling a procedure. Often used as “apply procedure `p` to arguments `a1` and `a2`.”

**pass** Usage “pass `X` to `Y`.” When calling procedure `Y`, supply `X` as one of the arguments.

**side-effect** In relation to an expression or procedure, some change to the system that does not involve the expression's value.

**iterate** To loop, or “do” the same code multiple times.

**variable** A name that refers to a exactly one value.

**binding** Also verb “to bind”. The pairing of a name with a value to make a variable.

**recurse** In a procedure, to call that same procedure again.

## Scheme

### 1. Basic Elements

- (a) *self-evaluating* - expressions whose value is the same as the expression.
  
  
  
  
  
  
  
  
  
  
- (b) *names* - Name is looked up in the symbol table to find the value associated with it. Names may be made of any collection of characters that doesn't start with a number.

### 2. Combination

( *procedure arguments-separated-by-spaces* )

Value is determined by evaluating the expression for the procedure and applying the resulting value to the value of the arguments.

### 3. Special Forms

(a) *define* - (**define** *name value*)

The name is bound to the result of evaluating the the value. Return value is *unspecified*.

(b) *if* - (**if** *test consequent alternative*)

If the value of the test is not false (#f), evaluate the consequent, otherwise evaluate the alternative.

## Problems

1. *Evaluation* - give values for the following expressions assuming they were evaluated in order. If the expression is erroneous simply indicate “error” for the value. If the expression returns an unspecified value, write whatever you want!

4

5.5

4.2e1

(+ 1 2)

(7)

(\* (+ 7 8) (- 5 6))

(define one 1)

(define two (+ 1 one))

(define five 3)

(+ five two)

(define biggie-size \*)

(define hamburger 1)

(biggie-size hamburger five)

(= 7 (+ 3 4))

(= #t #f)

((+ 5 6))

biggie-size

2. *Five* - write an expression that evaluates to 3.
3. *Five* - write a *more interesting* expression that evaluates to 3.