MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.001—Structure and Interpretation of Computer Programs
Fall Semester, 1996

Lecture Notes, September 10

Scheme Basics

Language Components

- Primitives
- Means of combination
  - procedure application
  - compound data structures
- Means of abstraction
  - naming
  - procedures
  - data abstractions

Rules for Scheme

1. (Almost) Every *expression* has a *value* (which is returned when the expression is *evaluated*).
2. Every value has a *type*.

Rules for Evaluation

1. If expression is *self-evaluating*, return value.
2. If a *name*, return value associated with that name in the environment.
3. If a *special form*, do something special.
4. If a *combination*, then
   (a) *Evaluate* all of the subexpressions of the combination (in any order).
   (b) *Apply* the *operator* to the values of the *operands* (the arguments) and return the result.

Rules for Application

1. If procedure is *primitive* (built-in), just do it.
2. If procedure is a *compound procedure*, then *evaluate* the *body* of the procedure with the *formal parameters* replaced by the *actual argument* values.

Taxonomy of Expressions

```
Expression
  Primitive Expression
    Constants (self-evaluating)
      Numerals - 7 == Sch-Num 7
      Strings - "hello" == Sch-String hello
      Booleans - #f #t == Sch-Bools
    Names - + == primitive procedure
            x == created by DEFINE or procedure application
  Compound Expressions
    Combinations - (<operator> <operand> <operand> ...)
    Special Forms
      define - (define <name> <exp>)
      lambda - (lambda (<formal1> <formal2> ...) <body>)
      if - (if <predicate> <consequent> <alternative>)
      ... more to come! ...
      (and ...)
      (or ..)
```

Approximations for Square Root

```
(define try
  (lambda (guess x)
    (if (good-enuf? guess x)
        guess
        (try (improve guess x) x))))

(define improve
  (lambda (guess x)
    (average guess (/ x guess))))

(define average
  (lambda (a b)
    (/ (+ a b) 2)))

(define good-enuf?
  (lambda (guess x)
    (< (abs (- (square guess) x)) 0.001)))

(define sqrt
  (lambda (x) (try 1 x)))
```