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 each formal parameter replaced by the corresponding actual argument value.
Taxonomy of Expressions

- **Primitive Expressions**
  
  - Constants (self-evaluating)
    
    * Numerals, e.g. \(7 == \text{Sch-Num} 7\)
    * Strings, e.g. "hello" == \text{Sch-String} hello
    * Booleans, e.g. \#f #t == \text{Sch-Bools}
  
  - Names
    
    * e.g. \(+ == \text{primitive procedure}\)
    * e.g. \(x == \text{variable created by define or procedure application}\)

- **Compound Expressions**
  
  - Combinations: \(<\text{operator}> <\text{operand}> <\text{operand}> ...\)
  
  - Special Forms
    
    * define: \(\text{define} <\text{name}> <\text{exp}>\)
    * lambda: \(\lambda ( <\text{formal1}> <\text{formal2}> ...) <\text{body}>\)
    * if: \(\text{if} <\text{predicate}> <\text{consequent}> <\text{alternative}>\)
    * ... more to come! ...

**Approximations for Square Root**

```scheme
(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)))
```