Scheme Reference Guide

(define var expr)
Evaluates expr. Binds the variable name var to the value in the global environment.

(quote object)
Returns object. E.g., '(1 2 3) returns (1 2 3).

(lambda (param1 ... paramn) body)
Returns a procedure that takes n expressions as its arguments, and when invoked evaluates the body. For example, the value of (lambda (x) (* x x)) is a one-argument procedure (which squares its input). To invoke this procedure, put it in parentheses with its arguments. Thus ((lambda (x) (* x x) 5) evaluates to 25.

(l lambda args body)
Syntax for variable-arity functions. If a single symbol is in the place of the arguments, the function takes any number of arguments (when the function is invoked, that symbol is bound to the list of arguments). A lambda expression with a dotted pair (e.g., (a b . rest)) requires one argument for each named symbol, aside from the last; when invoked, the list of remaining arguments are bound to the last symbol. (E.g., ((lambda (a . others) body) 1 2 3) has a bound to 1 and others to (2 3).)

(if test conseq alt)
Evaluates test. If the value is true (anything other than #f), then it evaluates conseq, whose value is the value of the if; otherwise it evaluates alt, whose value is the value of the if.

(cond (t1 v1) ... (tn vn))
Evaluates t1, t2, ... until one is true; then evaluates the corresponding vi, which is the value of the entire cond. The last test tn can be replaced by else, in which case the value of the last body is returned whenever all previous tis are false.

(let ((v1 e1) ... (vn en)) body)
Evaluates e1, e2, ...; then binds corresponding vis to their values. The value of the expression is the evaluation of body.

(let* ((v1 e1) ... (vn en)) body)
Similar, but when ek is being evaluated, the first k – 1 values are bound.

(letrec ((v1 e1) ... (vn en)) body)
Similar, but all of e1, e2, ... are within the scope of all of the variables v1, v2, ... so letrec allows the definition of mutually recursive procedures.

(apply f args)
Invoke the function f with the arguments as args.

(map f L)
Apply the function f to each element of L, and collect the results in a list.

(filter f L)
Apply the function f to each element of L, and collect all elements x of L for which (f x) is true in a list.