```
;; logic
#t  #f
(and _expr.._) ;short circuit
(or _expr.._)  ;short circuit
(not _expr_)

;; characters
#\a  #\1  #\newline  #\space
(char=? _a_ _b_) ; <, >, <=, >=
(char-ci=? _a_ _b_)
(char->integer _ch_)
(integer->char _i_)

;; strings
"hello"
(string-length _s_)
(string-ref _s_ _i_)
(string=? _a_ _b_) ; <, >, <=, >=
(string-ci=? _a_ _b_)
(substring _s_ _start_ _end_)
(string-append _s.._)
(string-set! _s_ _i_ _ch_)
(string-copy _s_)
```

```
;; symbols
(symbol->string _sym_)
(string->symbol _s_)

;; list/pair
()  (0 1 2)  (a . b)
(cons _h_ _t_)
(car _p_)  (cdr _p_)
(set-car! _p_ _h_)
(set-cdr! _p_ _t_)
(null? _o_)
(list? _o_)
(list _expr.._)
(length _o_)
(append _lst.._)
(reverse _lst_)
(list-ref _lst_ _i_)

;; vector
#() #(0 1 2)
(make-vector _len_)
(vector _expr.._)
(vector-length _v_)
(vector-ref _v_ _i_)
(vector-set! _v_ _i_ _expr_)
(vector->list _v_)
(list->vector _lst_)
```

```
;; input
(input-port? _obj_)
(current-input-port)
(open-input-file _filename_)
(close-input-port _port_)
(call-with-input-file
  _filename_ _thunk_)
(read _?port_)
(read-char _?port_)
(peek-char _?port_)
(eof-object? _obj_)
(char-read? _?port_)

;; output
(output-port? _obj_)
(current-output-port)
(open-output-file _filename_)
(close-output-port _port_)
(call-with-output-file
  _filename_ _thunk_)
(write _obj_ _?port_)
(display _obj_ _?port_)
(write-char _ch_ _?port_)
(newline _?port_)
```

```
;; quoting
'_e_   (quote _e_)
`_e_   (quasiquote _e_)
,_e_   (unquote _e_)
,@_e_  (unquote-splicing _e_)

;; control functions
(delay _e_)
(force _promise_)
(call-with-current-continuation
  _proc_)
(map _proc_ _lst.._)
(for-each _proc_ _lst.._)

;; macros
(let-syntax
  ((_keyword_ _transformer_) ..)
  _body_)
(define-syntax _keyword_
  _transformer_)
(syntax-rules (_literals.._)
  (_pattern_ _template_) ..)
;transformer
;patterns
x          ;variable
x ...      ;repetition
_pat_ ...  ;repeated pattern
```

Scheme

Programming Language
Quick Reference Booklet

Community References:
http://community.schemewiki.org/
http://schemers.org/

Standards:
http://r6rs.org/

Other Books:
http://www.scheme.com/tspl4/
http://mitpress.mit.edu/sicp/
http://www.htdp.org/

draft 3 - 1/17/2011

© 2011 Aaron Lahman
http://ra3s.com/

```
;; r5rs load module
(load _filename_)

;; variables
(define _var_ _expr.._)
(let ((_var_ _expr_)..) _expr.._)
(let* ..)   ;in sequence
(letrec ..) ;recursive procs

;; procedures
(define (_name _args.._) _e.._)
(lambda (_args.._) _e.._)

;; control flow
(if _test_ _true-expr_
  _false-expr_)
(cond (_test_ _body.._)..
      (_test_ => _thunk_)..
      (else _body.._))
(case _expr_
  ((_keys.._) _body.._)..
  (else _body.._))
(do ((_var_ _init_ _step_)..)
    (_test_ _exit-expr_..)
  _body.._)
;named let:
(let _name_ ((_v_ _e_)..) _e.._)
```

```
;; data types
(boolean? _e_)  (pair? _e_)
(symbol? _e_)   (number? _e_)
(char? _e_)     (vector? _e_)
(port? _e_)     (procedure? _e_)

;equality
(eq? _a_ _b_)     ;ref equality
(eqv? _a_ _b_)    ;number, string
(equal? _a_ _b_) ;list contents

;; numbers
2  3.5  1/3  6.02e23
+  -  *  /
= < > <= >=
(quotient _a_ _b_) ;integer
division
(modulo _a_ _b_)    ;unsigned
(remainer _a_ _b_) ;signed
(abs _n_)   (max _n.._)  (min
_n.._)
(zero? _n_) (even? _n_) (odd? _n_)
(positive? _n_ (negative? _n_)
```