femptolisp

Programming Language
Quick Reference Card

Man code.google.com/p/femtolisp/wiki/Manual
API code.google.com/p/femtolisp/wiki/APIReference

```
;; comments
; text between ; and eol is skipped
#| this text is also skipped \#
#; skips next single s-expression

;;quoting
'{e}    (quote{e})
`{e}    (quasiquote{e})
,{e}    (unquote{e})
,@{e} (unquote-splicing{e})

(set! {sym} {e})

;; data types
(boolean? {e})       (pair? {e})
(symbol? {e})        (number? {e})
(char? {e})          (vector? {e})
(atom? {e})          (fixnum? {e})
procedure?  builtin?  bound?
negative?  zero?  positive?
even?  odd?  null? identity

;equality
(eq? {a} {b})
(eqv? {a} {b})       ;number, string
(equal? {a} {b})     ;list contents

;; operators
+  -  *  /  >  <
(quotient a b)    ;integer division
(= {nums..})      ;numeric equality
(lognot a)        (logand a b)
(logior a b)      (logxor a b)
(ash a)  ; bit shift
mod mod0  div  abs  max  min

;; logic
#t   #f
(and {expr..})    ;short circuit
(or {expr..})     ;short circuit
(not {expr})      (compare? {e} {e})
```

```
;;characters
#\a  #\1
#\newline  #\space

;;strings
"hello"

;;list/pair
(012)   ()
(cons{h}{t})
(car{p})   (cdr{p})
(set-car!{p}{i})
(set-cdr!{p}{i})
(list?{o})
(length{p})
(list{expr..})
(append{lst..})
(reverse{lst})
(list-ref{lst}{i})

;;vector
#(012)#()
(vector{expr..})
(vector.alloc {n} {x})
(aref{v})
(aset!{v} {i} {x})
(vector->list{v})
(list->vector{lst})
```

```
;; r5rs load module
(load{filename-string})

;; variables
(define {var} {expr..})
(let (({var} {expr})..) {expr..})
(let* ..)          ;in sequence
(letrec ..)        ;recursive procs

;; procedures
(define ({proc} {args..}) {body..})
(lambda ({args..}) {body..})

;; 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-body..})
  {body..})
(for x y (lambda ({args})
   {body}))
(while {test} . {body..})
```

```
;named let:
(let {name} (({v} {e})..) {e..})

(yeild x) return a value in generator
(prog1 {expr}..) ;eval & return 1st
(trycatch {expr} {function})
(raise {expr})
(return {expr})

;;control functions
(force{promise})
(with-delimited-continuations
  {proc})
(map{proc}{lst..})
(for-each{proc}{lst..})

;;macros
(let-syntax
  (({keyword}{transformer})..)
  {body..})
(define-syntax
  {keyword} {transformer})
;transformer
(syntax-rules({literals..})
  ({pattern}{template})..)
;patterns
x               ;variable
x...            ;repetition
{pat}...        ;repeated pattern

;; other
(table k v k v ...)

append!, assoc, assv, assq,
member, memv, memq, every, any,
list-tail, list-ref, list*,
last-pair, lastcdr, length=,
length>, map!, mapcar, for-each,
filter, count, foldr, foldl,
reverse!, copy-list, copy-tree,
map-int, iota, revappend, nreconc,
delete-duplicates
```