***Core*** ***Reference e***

***core* name space, version *0.0.14***

***type identifiers s***

%lambda closure lambda

%exception exception

%vector vector

%closure lexical closure

*bool* false if *()*, otherwise true

*char*

*cons*

*env*

*fixnum fix*

*float*

*function fn*

*keyword key*

*namespace ns*

*null*

*stream*

*string str*

*struct*

*symbol sym*

*vector vec*

***core s***

**load** *string bool* load file through

core reader

**eval** *T*  *T*eval form

**apply** *fn* *list* *T*apply *fn* to *list*

**compile** *T T* compile *T* in null

environment

**identity** *T T*identity function

**type-of** *T symbol*object type

**eql** *T T* *bool* eql predicate

***special*** ***forms s***

**%defmacro** *sym list . body*

*sym* define macro

**%lambda** *list* . body *fn* define closure

**%if** *T 'T T* conditional

**%if** *T 'T ''T T* conditional

***lists s***

**assq** *T* *list* *list* assoc

**rassq** *T* *list*  *list* reverse assoc

**find-if** *fn* *list T* element if applied

*fn* returns

an *atom*, else *()*

**position-if** *fn* *list T* index of element

if *fn* returns an

*atom*, else *()*

**dropl** *list* *fixnum* *list* drop left

**dropr** *list* *fixnum list* drop right

**foldl** *fn T list list* left fold

**foldr** *fn T list list* right fold

**mapc** *fn list* *list* apply *fn* to *list*

cars, return *list*

**mapcar** *fn list list* new list from

applying *fn* to

*list* cars

**mapl** *fn list* *list* apply *fn* to *list*

cdrs, return *list*

**maplist** *fn* *list list* new list from

applying *fn* to

*list* cdrs

**append** *list*  *list* append lists

**reverse** *list list* reverse *list*

***vectors*** ***s***

**make-vector** *list list* reverse *list*

**bit-vector-p** *vec bool* a bit vector?

**vector-displaced-p** *vec bool* a displaced vector?

**vector-ref** *vec* *fixnum T* index *vec*

**vector-slice** *vec fix ‘fix vec* displaced vector -

start, length

**vector-type** *vec symbol* specialized vector

type

***macros xu***

**define-symbol-macro** *symbol* *T*

*symbol* define symbol macro

**get-macro-character** *char*

*T* expand character

macro

**set-macro-character** *char* *fn* *bool*

*symbol* create character

macro

**macro-function** *symbol* *env*

*fn* macro expander

function or *()*

**macroexpand** *T env T* expand macro

completely

**macroexpand-1** *T env T* expand macro once

***symbols xu***

**gensym** *sym* create unique

uninterned symbol

**gentemp** *sym* create unique

temp symbol

**s*treams*** ***s***

**read** *stream bool T* *T* read from stream

with EOF handling

**write** *T bool stream* *T* write escaped

object to stream

***predicates s***

**minusp** *fix* *bool* negative value

**numberp** *T* *bool* float or fixnum

**charp** *T bool* char

**consp** *T* *bool* cons

**fixnump** *T* *bool* fixnum

**floatp** *T* *bool* float

**functionp** *T* *bool* function

**keywordp** *T* *bool* keyword

**listp** *T* *bool* cons or ()

**namespacep** *T* *bool* namespace

**null** *T* *bool* :nil or ()

**streamp** *T* *bool* stream

**stringp** *T* *bool* char vector

**structp** *T* *bool* struct

**symbolp** *T* *bool* symbol

**vectorp** *T* *bool* vector

***streams xu***

**read** *stream bool T* *T* read from stream

with EOF handling

**write** *T bool stream* *T* write escaped

object to stream

**e*xceptions n***

**error** *T symbol list string* error format

**exceptionp** *struct* *bool* predicate

**raise** *T sym str* raise exception

**raise-env** *T sym str*  raise exception

**warn** *T string* *T* warning

**with-exception** *fn fn T* catch exception

***macro definitions***  *s*

**and** ... *T* logical *and* of ...

**cond** ... *T* cond switch

**let** *list* ... *T* lexical bindings

**let\*** *list* ... *T* dependent list

of bindings

**or** ... *T* logical *or* of ...

**progn** ... *T* evaluate rest list,

return final

evaluation

**unless** *T* ... *T* if *T* is *()*, *(***progn** *…)*

else *()*

**when** *T* ... *T* if *T* is an *atom*,

*(***progn** *...)* else ()

***rest functions*** *s*

**append** *…*  *list* append lists

**apply** *fn* … *T* apply *fn to ...*

**format** *T* *string* ... *T* formatted output

**funcall** *fn* … *T* apply *fn to ...*

**list** … *list list of ...*

**list\*** ... *list list dot* ...

**mapc** *fn* ... *list* mapcof ...

**mapcar** *fn* ... *list* mapcarof …

**mapl** *fn* ... *list* maplof ...

**maplist** *fn* ... *list* maplistof …

**vector** ... *vec* make general vector

of ...

***Reader Syntax x***

; comment to end of line

#|...|# block comment

‘*form* quoted form

`*form* backquoted form

`(*...)* backquoted list (proper lists)

,*form* eval backquoted form

,@*form* eval-splice backquoted form

(…) constant *list*

() empty *list*, prints as :nil

(… . .) dotted *list*

“…” *string, char vector*

*\* single escape in strings

#\*... bit vector

#x... hexadecimal *fixnum*

#. read-time eval

#\. *char*

#(:type …) *vector*

#s(:type …) *struct*

#:symbol uninterned *symbol*

“`,; terminating macro char

# non-terminating macro char

!$%&\*+-. symbol constituents

<>=?@[]|

:^\_{}~/

A..Za..z

0..9

0x09 #\tab whitespace

0x0a #\linefeed

0x0c #\page

0x0d #\return

0x20 #\space