\( (\text{sinh} \ a) \)  \( \text{ sinh} \ a, \text{ cosh} \ a, \text{ or tanh} \ a, \text{ respectively.} \)

\( (\text{cosh} \ a) \)

\( (\text{tanh} \ a) \)

\( (\text{asinh} \ a) \)  \( \text{ asinh} \ a, \text{ acosh} \ a, \text{ or atanh} \ a, \text{ respectively.} \)

\( (\text{acosh} \ a) \)

\( (\text{atanh} \ a) \)

\( (\text{cis} \ a) \)  \( \text{ Return } e^{i \ a} = \cos \ a + i \sin \ a. \)

\( (\text{conjugate} \ a) \)  \( \text{ Return complex conjugate of } a. \)

\( (\text{max} \ num^+) \)  \( \text{ Greatest or least, respectively, of } nums. \)

\( (\text{min} \ num^+) \)

\( (\text{round} \ n) \)  \( \text{ Return as integer or float, respectively, } n/d \text{ rounded, or rounded towards } -\infty, +\infty, \text{ or } 0, \text{ respectively; and remainder.} \)

\( (\text{mod} \ n) \)

\( (\text{rem} \ n) \)  \( \text{ Same as floor or truncate, respectively, but return remainder only.} \)

\( (\text{random limit [state]} [\text{random-state}]) \)  \( \text{ Return non-negative random number less than } limit, \text{ and of the same type.} \)

\( (\text{make-random-state } \text{ [state]} ) \)  \( \text{ Copy of random-state object state or of the current random state; or a randomly initialized fresh random state.} \)

\( *\text{random-state}* \)  \( \text{ Current random state.} \)

\( (\text{float-sign num-a [num-b]} ) \)  \( \text{ num-b with num-a's sign.} \)

\( (\text{signum} \ n) \)  \( \text{ Number of magnitude 1 representing sign or phase of } n. \)

\( (\text{numerator rational}) \)

\( (\text{denominator rational}) \)  \( \text{ Numerator or denominator, respectively, of rational's canonical form.} \)

\( (\text{realpart number}) \)

\( (\text{imagpart number}) \)  \( \text{ Real part or imaginary part, respectively, of number.} \)

\( (\text{complex real [imag]}) \)  \( \text{ Make a complex number.} \)

\( (\text{phase num}) \)  \( \text{ Angle of num's polar representation.} \)

\( (\text{abs n}) \)  \( \text{ Return } |n|. \)

\( (\text{rational real}) \)

\( (\text{rationalize real}) \)  \( \text{ Convert real to rational. Assume complete/limited accuracy for real.} \)

\( (\text{float real [prototype]} ) \)  \( \text{ Convert real into float with type of prototype.} \)
1 Numbers

1.1 Predicates

\( (\equiv \text{number}^+) \)
\( (\equiv \text{number}^+) \)
\( (\equiv \text{number}^+) \)
\( (\equiv \text{number}^+) \)
\( (\equiv \text{number}^+) \)
\( (\equiv \text{number}^+) \)
\( (\equiv \text{number}^+) \)
\( (\equiv \text{number}^+) \)

\( (\Rightarrow \text{number}^+) \)
\( (\Rightarrow \text{number}^+) \)
\( (\Rightarrow \text{number}^+) \)
\( (\Rightarrow \text{number}^+) \)
\( (\Rightarrow \text{number}^+) \)

\( \Rightarrow \) Return \( \text{T} \) if numbers are monotonically decreasing, monotonically non-increasing, monotonically increasing, or monotonically non-decreasing, respectively.

\( (\text{minusp} \ a) \)
\( (\text{zerop} \ a) \)
\( (\text{plusp} \ a) \)

\( (\text{evenp} \ \text{int}) \)
\( (\text{oddp} \ \text{int}) \)

\( (\text{numberp} \ \text{foo}) \)
\( (\text{realp} \ \text{foo}) \)
\( (\text{rationalp} \ \text{foo}) \)
\( (\text{floatp} \ \text{foo}) \)
\( (\text{integerp} \ \text{foo}) \)
\( (\text{complexp} \ \text{foo}) \)
\( (\text{random-state-p} \ \text{foo}) \)

1.2 Numeric Functions

\( (\text{+} \ a \ b^+) \)
\( (\ast \ a \ b^+) \)

\( \Rightarrow \) Return \( \sum a \) or \( \Pi a \), respectively.

\( (\text{+}^+ \ a \ b^+) \)
\( (\ast^+ \ a \ b^+) \)

\( \Rightarrow \) Return \( a - \sum b \) or \( a/\Pi b \), respectively. Without any bs, return \( -a \) or \( 1/a \), respectively.

\( (\text{+} \ a \ b^+) \)
\( (\ast \ a \ b^+) \)

\( \Rightarrow \) Return \( a + 1 \) or \( a - 1 \), respectively.

\( (\text{inf} \ \text{delta}) \)
\( (\text{decf} \ \text{delta}) \)

\( \Rightarrow \) Placeholder for actual code.

\( \text{me} \)

\( \Rightarrow \) Either one \( \text{foo} \) or nothing; defaults to \( \text{bar} \).

\( \text{foo}^+ \)\{ \text{foo}\}^+ \)
\( \Rightarrow \) Zero or more \( \text{foos} \).

\( \text{foo}^+ \)\{ \text{foo}\}^+ \)
\( \Rightarrow \) One or more \( \text{foos} \).

\( \text{foos} \)
\( \Rightarrow \) English plural denotes a list argument.

\( \{\text{foo}| \text{bar}| \text{baz}\} \)
\( \Rightarrow \) Either \( \text{foo} \), \( \text{bar} \), or \( \text{baz} \).

\( \{\text{foo}| \text{bar}| \text{baz}\} \)
\( \Rightarrow \) Anything from none to each of \( \text{foo} \), \( \text{bar} \), and \( \text{baz} \).

\( \text{foo} \)
\( \Rightarrow \) Argument \( \text{foo} \) is not evaluated.

\( \text{bar} \)
\( \Rightarrow \) Argument \( \text{bar} \) is possibly modified.

\( \text{foo}^b \)
\( \Rightarrow \) \( \text{foo}^+ \) is evaluated as in \( \text{progn} \); see page 21.

\( \text{foo}; \text{bar}; \text{baz} \)
\( \Rightarrow \) Primary, secondary, and nth return value.

\( \text{T}; \text{NIL} \)
\( \Rightarrow \) \text{T}, or truth in general; and \text{NIL} or \text{()}.
3 Strings

Strings can as well be manipulated by array and sequence functions; see pages 11 and 12.

(\text{string foo}) \triangleright \text{T} if foo is of indicated type.

(\text{string \_p foo}) \triangleright \text{T} if foo is of indicated type.

\begin{align*}
\{ & \text{string=} \} \quad \text{foo bar} \\
\{ & \text{string= \_p} \} \quad \text{foo bar}
\end{align*}

\triangleright \text{Return } \text{T} if \text{subsequences of } \text{foo and } \text{bar} are equal. Obey/ignore, respectively, case.

\begin{align*}
\{ & \text{string\_p} \} \quad \text{not-equal} \\
\{ & \text{string\_p} \} \quad \text{greaterp} \\
\{ & \text{string\_p} \} \quad \text{lessp} \\
\{ & \text{string\_p} \} \quad \text{not-greaterp}
\end{align*}

\triangleright \text{If } \text{foo} is lexicographically not equal, greater, not less, less, or not greater, respectively, then return position of first mismatching character in \text{foo}. Otherwise return \text{NIL}. Obey/ignore, respectively, case.

(\text{make-string size \{initial-element char | element-type type\} character})

\triangleright \text{Return string of length } \text{size}.

(\text{string x})

(\text{string\_upcase x})

(\text{string-downcase x})

\triangleright \text{Convert } x (\text{symbol, string, or character}) \text{ into a string, a string with capitalized words, an all-upper case string, or an all-lowercase string, respectively.}

\begin{align*}
\{ & \text{string\_capitalize \_p} \} \\
\{ & \text{string\_upcase \_p} \} \\
\{ & \text{string\_downcase \_p} \}
\end{align*}

\triangleright \text{Convert string into a string with capitalized words, an all-upper case string, or an all-lowercase string, respectively.}

\begin{align*}
\{ & \text{string\_trim \_p} \} \\
\{ & \text{string\_left\_trim \_p} \} \\
\{ & \text{string\_right\_trim \_p} \}
\end{align*}

\triangleright \text{Return } \text{string} \text{ with all characters in sequence } \text{char-bag} \text{ removed from both ends, from the beginning, or from the end, respectively.}

(\text{char string i})

(\text{schar string i})

\triangleright \text{Return zero-indexed } i \text{th character of string ignoring/obeying, respectively, fill pointer. set\_table.}

(\text{parse\_integer string \{start end \_p \} \_radix int \_junk\_allowed bool \_p})

\triangleright \text{Return integer parsed from } \text{string} \text{ and index of parse end.}

4 Conses

4.1 Predicates

(\text{consp foo}) \triangleright \text{T} if foo is of indicated type.

(\text{listp foo}) \triangleright \text{T} if foo is of indicated type.

(\text{endp list}) \triangleright \text{T} if list/foo is \text{NIL}.

(\text{null foo}) \triangleright \text{T} if list/foo is \text{NIL}.

1.3 Logic Functions

Negative integers are used in two’s complement representation.

(\text{boole operation int-a int-b})

\triangleright \text{Return value of bitwise logical } \text{operation}. \text{operations are}

- \text{boole-1} \quad \triangleright \text{int-a.}
- \text{boole-2} \quad \triangleright \text{int-b.}
- \text{boole-c1} \quad \triangleright \text{not-int-a.}
- \text{boole-c2} \quad \triangleright \text{not-int-b.}
- \text{boole-set} \quad \triangleright \text{All bits set.}
- \text{boole-clr} \quad \triangleright \text{All bits zero.}
- \text{boole-eqv} \quad \triangleright \text{int-a} \equiv \text{int-b.}
- \text{boole-and} \quad \triangleright \text{int-a} \wedge \text{int-b.}
- \text{boole-andc1} \quad \triangleright \text{not-int-a} \wedge \text{int-b.}
- \text{boole-andc2} \quad \triangleright \text{int-a} \wedge \text{not-int-b.}
- \text{boole-nand} \quad \triangleright \text{not-int-a} \wedge \text{not-int-b.}
- \text{boole-ior} \quad \triangleright \text{int-a} \lor \text{int-b.}
- \text{boole-ocl} \quad \triangleright \text{not-int-a} \lor \text{int-b.}
- \text{boole-orc2} \quad \triangleright \text{int-a} \lor \text{not-int-b.}
- \text{boole-nor} \quad \triangleright \text{not-int-a} \lor \text{not-int-b.}

(\text{lognot integer}) \quad \triangleright \text{not-integer.}

(\text{logeqv integer\_p})

(\text{logand integer\_p})

\triangleright \text{Return value of exclusive-ored or anded integers, respectively. Without any integer, return 0.}

(\text{logandc1 int-a int-b}) \quad \triangleright \text{not-int-a} \wedge \text{int-b.}

(\text{logandc2 int-a int-b}) \quad \triangleright \text{int-a} \wedge \text{not-int-b.}

(\text{lognot int-a int-b}) \quad \triangleright \text{not-int-a} \wedge \text{int-b.}

(\text{logxor integer\_p})

(\text{logior integer\_p})

\triangleright \text{Return value of exclusive-ored or ored integers, respectively. Without any integer, return 0.}

(\text{logorcl int-a int-b}) \quad \triangleright \text{not-int-a} \lor \text{int-b.}

(\text{logorcl int-a int-b}) \quad \triangleright \text{int-a} \lor \text{not-int-b.}

(\text{lognor int-a int-b}) \quad \triangleright \text{not-int-a} \lor \text{not-int-b.}

(\text{logbitp i int}) \quad \triangleright \text{T} if zero-indexed } i \text{th bit of } \text{int} \text{ is set.}

(\text{logtest int-a int-b})

\triangleright \text{Return } \text{T} \text{ if there is any bit set in } \text{int-a} \text{ which is set in } \text{int-b} \text{ as well.}

(\text{logcount int})

\triangleright \text{Number of 1 bits in } \text{int} \geq 0, \text{number of 0 bits in } \text{int} < 0.
1.4 Integer Functions

(\texttt{integer-length integer})
\quad \rightarrow \text{Number of bits necessary to represent } integer.

(\texttt{ldb-test byte-spec integer})
\quad \rightarrow \text{Return } \top \text{ if any bit specified by } byte-spec \text{ in } integer \text{ is set.}

(\texttt{ash integer count})
\quad \rightarrow \text{Return copy of } integer \text{ arithmetically shifted left by } count \text{ adding zeros at the right, or, for } count < 0, \text{ shifted right discarding bits.}

(\texttt{ldb byte-spec integer})
\quad \rightarrow \text{Extract byte denoted by } byte-spec \text{ from } integer. \textsf{setf}a\text{ble.}

(\texttt{(deposit-field int-a byte-spec int-b)})
\quad \rightarrow \text{Return int-b with bits denoted by } byte-spec \text{ replaced by corresponding bits of int-a, or by the low } (\texttt{byte-size byte-spec}) \text{ bits of int-a, respectively.}

(\texttt{(mask-field byte-spec integer)})
\quad \rightarrow \text{Return copy of integer with all bits unset but those denoted by } byte-spec. \textsf{setf}able.

(\texttt{byte size position})
\quad \rightarrow \text{Byte specifier for a byte of size bits starting at a weight of } 2^{\text{position}}.

(\texttt{byte-size byte-spec})
(\texttt{byte-position byte-spec})
\quad \rightarrow \text{Size or position, respectively, of } byte-spec.

1.5 Implementation-Dependent

\begin{itemize}
\item \texttt{short-float}
\item \texttt{single-float}
\item \texttt{double-float}
\item \texttt{long-float}
\end{itemize}
\quad \rightarrow \text{Smallest possible number making a difference when added or subtracted, respectively.}

\begin{itemize}
\item \texttt{least-negative}
\item \texttt{least-negative-normalized}
\item \texttt{least-positive}
\item \texttt{least-positive-normalized}
\end{itemize}
\quad \rightarrow \text{Available numbers closest to } -0 \text{ or } +0, \text{ respectively.}

\begin{itemize}
\item \texttt{most-negative}
\item \texttt{most-positive}
\end{itemize}
\quad \rightarrow \text{Available numbers closest to } -\infty \text{ or } +\infty, \text{ respectively.}

(\texttt{decode-float n})
(\texttt{integer-decode-float n})
\quad \rightarrow \text{Return significand, exponent, and sign of } float \text{ } n.

(\texttt{scale-float n \{i\}})
\quad \rightarrow \text{With } n \text{’s radix } b, \text{ return } nb^i.

(\texttt{float-radix n})
(\texttt{float-digits n})
(\texttt{float-precision n})
\quad \rightarrow \text{Radix, number of digits in that radix, or precision in that radix, respectively, of } float \text{ } n.

(\texttt{upgraded-complex-part-type foo \{environment\}})
\quad \rightarrow \text{Type of most specialized } complex \text{ number able to hold parts of type } foo.

2 Characters

The \texttt{standard-char} type comprises a-z, A-Z, 0-9, Newline, Space, and !"#$%'&()*+,-./:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~.

(\texttt{characterp foo})
(\texttt{standard-char-p char})
\quad \rightarrow \top \text{ if argument is of indicated type.}

(\texttt{graphic-char-p character})
(\texttt{alpha-char-p character})
(\texttt{alphanumericp character})
\quad \rightarrow \top \text{ if character is visible, alphabetic, or alphanumeric, respectively.}

(\texttt{upper-case-p character})
(\texttt{lower-case-p character})
(\texttt{both-case-p character})
\quad \rightarrow \top \text{ if character is uppercase, lowercase, or able to be in another case, respectively.}

(\texttt{digit-char-p character \{radix\}})
\quad \rightarrow \text{Return } \top \text{ if all characters, or none, respectively, are equal.}

(\texttt{char= character+})
(\texttt{char/= character+})
(\texttt{char< character+})
(\texttt{char<= character+})
\quad \rightarrow \top \text{ if all characters, or none, respectively, are equal ignoring case.}

(\texttt{char> character+})
(\texttt{char>= character+})
(\texttt{char< character+})
(\texttt{char<= character+})
\quad \rightarrow \top \text{ if characters are monotonically decreasing, monotonically non-increasing, monotonically increasing, or monotonically non-decreasing, respectively.}

(\texttt{char-greaterp character+})
(\texttt{char-not-lessp character+})
(\texttt{char-lessp character+})
(\texttt{char-not-greaterp character+})
\quad \rightarrow \top \text{ if characters are monotonically decreasing, monotonically non-increasing, monotonically increasing, or monotonically non-decreasing, respectively, ignoring case.}

(\texttt{char-upcase character})
(\texttt{char-downcase character})
\quad \rightarrow \text{Return corresponding uppercase/lowercase } character, \text{ respectively.}

(\texttt{digit-char i \{radix\}})
\quad \rightarrow \text{Character representing digit } i.

(\texttt{char-name char})
\quad \rightarrow \text{char’s name if any, or } \texttt{NIL}.

(\texttt{name-char foo})
\quad \rightarrow \text{Character named } foo \text{ if any, or } \texttt{NIL}.

(\texttt{char-int character})
(\texttt{char-code character})
\quad \rightarrow \text{Code of character.}

(\texttt{code-char code})
\quad \rightarrow \text{Character with code.}

\texttt{char-code-limit} \quad \rightarrow \text{Upper bound of } (\texttt{char-code char}); \geq 96.

(\texttt{character c})
\quad \rightarrow \text{Return } \#\backslash c.
(\text{atom} \ foo) \ \triangleright \ \text{Return } \top \text{ if } \ foo \text{ is not a cons.}

(\text{tails} \ foo \ list) \ \triangleright \ \text{Return } \top \text{ if } \ foo \text{ is a tail of } \ list.

(\text{member} \ foo \ list \ \{ \begin{array}{ll} \text{test function} & \#R \ \\ \text{test-not function} & \#R \ {\text{key function}} \end{array} \})
\triangleright \ \text{Return tail of } \ list \text{ starting with its first element matching } \ foo. \ \text{Return } \top \text{ if there is no such element.}

(\text{member-if} \ \{ \begin{array}{ll} \text{test list} & \#R \ \{ \begin{array}{l} \text{key function} \end{array} \} \end{array} \})
\triangleright \ \text{Return tail of } \ list \text{ starting with its first element satisfying } \ test. \ \text{Return } \top \text{ if there is no such element.}

(\text{subsetp} \ list-a \ list-b \ \{ \begin{array}{ll} \text{test function} & \#R \ \\ \text{test-not function} & \#R \ \{ \begin{array}{l} \text{key function} \end{array} \} \end{array} \})
\triangleright \ \text{Return } \top \text{ if } \ list-a \text{ is a subset of } \ list-b.

4.2 \ Lists

(\text{cons} \ foo \ bar) \ \triangleright \ \text{Return new cons } (\ foo, \ bar).\

(\text{list} \ foo^+) \ \triangleright \ \text{Return list of } \ foo.

(\text{list} \ foo^+) \ \triangleright \ \text{Return list of } \ foo.\

(\text{make-list} \ num \ \{ \begin{array}{l} \text{initial-element} \ end \} \})
\triangleright \ \text{New list with } \ num \text{ elements set to } \ foo.

(\text{list-length} \ list) \ \triangleright \ \text{Length of } \ list; \ \text{NIL for circular } \ list.

(\text{car} \ list) \ \triangleright \ \text{Car of } \ list \text{ or } \text{NIL if } \ list \text{ is } \text{NIL setf}.

(\text{cdr} \ list) \ \triangleright \ \text{Cdr of } \ list \text{ or } \text{NIL if } \ list \text{ is } \text{NIL setf}.

(\text{rest} \ list) \ \triangleright \ \text{Cdr of } \ list \text{ or } \text{NIL if } \ list \text{ is } \text{NIL setf}.

(\text{nthcdr} \ n \ list) \ \triangleright \ \text{Return tail of } \ list \text{ after calling } \text{cdr} \ n \text{ times}.

(\text{first} \ \{ \begin{array}{l} \text{second} \ \{ \ \ddots \ {\text{list}} \end{array} \})
\triangleright \ \text{Return } \text{nth element of } \ list \text{ if any, or } \text{NIL otherwise setf}.

(\text{nth} \ n \ list) \ \triangleright \ \text{Zero-indexed } \text{nth element of } \ list. \ \text{setf}.

(\text{caddr} \ list) \ \triangleright \ \text{With } X \text{ being one to four as and ds representing } \text{cars} \ \text{and } \text{cdrs}, \ \text{e.g. } (\text{caddr} \ bar) \text{ is equivalent to } (\text{car} \ (\text{cdr} \ bar)). \ \text{setf}.

(\text{last} \ list \ \{ \begin{array}{l} \text{num} \end{array} \})
\triangleright \ \text{Return last } \ num \text{ conses of } \ list.

(\text{butlast} \ list) \ \triangleright \ \text{list excluding last } \ num \text{ conses}.

(\text{rplaca} \ \text{cons} \ object) \ \triangleright \ \text{Replace car, or cdr, respectively, of } \text{cons with } \ object.

(\text{rplacd} \ object) \ \triangleright \ \text{Replace car, or cdr, respectively, of } \text{cons with } \ object.

(\text{adiff} \ list \ foo) \ \triangleright \ \text{If } \ foo \text{ is a tail of } \ list, \text{ return preceding part of } \ list. \ \text{Otherwise return } \ list.

(\text{adjoin} \ foo \ list \ \{ \begin{array}{ll} \text{test function} & \#R \ \\ \text{test-not function} & \#R \ \{ \begin{array}{l} \text{key function} \end{array} \} \end{array} \})
\triangleright \ \text{Return list if } \ foo \text{ is already member of } \ list. \ \text{If not, return } (\text{cons} \ foo \ list).

(\text{pop} \ \text{place}) \ \triangleright \ \text{Set place to } (\text{cdr} \ \text{place}), \ \text{return } (\text{car} \ \text{place}).
\textbf{5.1 Predicates}

\begin{verbatim}$\textsf{pair}\,foo\,bar$\end{verbatim} \quad $\text{Return } \mathbb{T}$ \text{if } \text{foo} \text{ and } \text{bar} \text{ have same shape and leaves satisfying test.}

\begin{verbatim}$\textsf{assoc}\,\text{key}\,\text{value}\,\text{alist}$\end{verbatim} \quad $\text{Return } \text{alist} \text{ with a } (\text{key} \, , \text{value}) \text{ pair added.}$

\begin{verbatim}$\textsf{assoc-\text{-}if}\,-\text{not}\,\text{key}\,\text{test}\,\text{alist}$\end{verbatim} \quad $\text{Return } \text{alist} \text{ with } \text{test} \text{ if } \text{key} \text{ satisfies } \text{test}.$

\begin{verbatim}$\textsf{copy\,-\text{-}alist}\,\text{alist}$\end{verbatim} \quad $\text{Return copy of } \text{alist}.$

\begin{verbatim}$\textsf{sublis}\,\text{association\,-\text{-}list}\,\text{tree}$\end{verbatim} \quad $\text{Make } \text{copy} \text{ of } \text{tree} \text{ with each subtree or leaf matching a } \text{key} \text{ in } \text{association\,-\text{-}list} \text{ replaced by that } \text{key}'s \text{ value.}$

\begin{verbatim}$\textsf{copy\,-\text{-}tree}\,\text{tree}$\end{verbatim} \quad $\text{Copy } \text{tree} \text{ with same shape and leaves.}$

\begin{verbatim}$\textsf{intersection}\,\text{set}\,-\text{-}difference\,\text{union}\,\text{set\,-\text{-}exclusive\,-\text{-}or}\,\text{nintersection}\,\text{nset\,-\text{-}difference}\,\text{union}\,\text{nset\,-\text{-}exclusive\,-\text{-}or}$\end{verbatim} \quad $\text{Return } a \cap b, a \setminus b, a \cup b, \text{ or } a \triangle b, \text{ respectively, of lists } a \text{ and } b.$

\section{Arrays}

\subsection{5.2 Array Functions}

\begin{verbatim}$\textsf{make\,-\text{-}array}\,\text{dimension\,-\text{-}sizes}\,\text{adjust\,-\text{-}array}$\end{verbatim} \quad $\text{Return } \text{fresh, or readjust, respectively, } \text{vector} \text{ or } \text{array.}$

\begin{verbatim}$\textsf{aref}\,\text{array}\,\text{subscripts}$\end{verbatim} \quad $\text{Return array element pointed to by } \text{subscripts}. \text{ setfable.}$

\begin{verbatim}$\textsf{row\,-\text{-}major\,-\text{-}aref}\,\text{array}\,\text{i}$\end{verbatim} \quad $\text{Return } \text{i} \text{th element of } \text{array} \text{ in row\,-\text{-}major order. \text{ setfable.}$

\begin{verbatim}$\textsf{array\,-\text{-}dimensions}\,\text{array}$\end{verbatim} \quad $\text{Index in row\,-\text{-}major order of the element denoted by } \text{subscripts.}$

\begin{verbatim}$\textsf{array\,-\text{-}dimension}\,\text{array}\,\text{i}$\end{verbatim} \quad $\text{List containing the lengths of } \text{array}'s \text{ dimensions.}$

\begin{verbatim}$\textsf{array\,-\text{-}total\,-\text{-}size}\,\text{array}$\end{verbatim} \quad $\text{Number of elements in } \text{array.}$

\begin{verbatim}$\textsf{array\,-\text{-}rank}\,\text{array}$\end{verbatim} \quad $\text{Number of dimensions of } \text{array.}$

\begin{verbatim}$\textsf{array\,-\text{-}displacement}\,\text{array}\,\text{offset}$\end{verbatim} \quad $\text{Target array and offset.}$
8 Structures

(defstruct foo
  (conc-name (slot-name prefix))
  (constructor (make-foo (ord-lambda))))

(include struct
  (slot (init :type sl-type :read-only b))
  (copier (slot sq)))

(doc (slot init :type sl-type :read-only b))

▷ Define structure foo together with functions MAKE-foo, COPY-foo and foo-P; and setable accessor foo-slot. Instances are of class foo or, if defstruct option :type is given, of the specified type. They can be created by (MAKE-foo :slot-value*); or, if ord-lambda (see page 18) is given, by (maker arg* :key-value*). In the latter case, args and :keys correspond to the positional and keyword parameters defined in ord-lambda whose vars in turn correspond to slots.

:print-object;/print-function generate a :print-object method for an instance bar of foo calling (o-printer bar stream) or (f-printer bar stream print-level), respectively. If :type without :named is given, no foo-P is created.

(copy-structure structure)

▷ Return copy of structure with shared slot values.

9 Control Structure

9.1 Predicates

(eq foo bar)

▷ T if foo and bar are identical.

eql foo bar

▷ T if foo and bar are identical, or the same character, or numbers of the same type and value.

equal foo bar

▷ T if foo and bar are eql, or are equivalent pathnames, or are conses with equal cars and cdds, or are strings or bit-vectors with eql elements below their fill pointers.

equalp foo bar

▷ T if foo and bar are identical; or are the same character ignoring case; or are numbers of the same value ignoring type; or are equivalent pathnames; or are conses or arrays of the same shape with equal elements; or are structures of the same type with equalp elements; or are hash-tables of the same size with the same :test function, the same keys in terms of :test function, and equalp elements.

(not foo)

▷ T if foo is NIL; NIL otherwise.

boundp symbol

▷ T if symbol is a special variable.

6.2 Sequence Functions

(make-sequence sequence-type size [initial-element foo])

▷ Make sequence of sequence-type with size elements.

(concatenate type sequence*)

▷ Return concatenated sequence of type.

(merge type sequence-a sequence-b test [key function])

▷ Return interleaved sequence of type. Merged sequence will be sorted if both sequence-a and sequence-b are sorted.

(fill sequence foo [:start start] [:end end])

▷ Return sequence after setting elements between start and end to foo.

(length sequence)

▷ Return length of sequence (being value of fill pointer if applicable).

(count foo sequence)

▷ Return number of elements in sequence which match foo.

(count-if test sequence)

▷ Return number of elements in sequence which satisfy test.

(elt sequence index)

▷ Return element of sequence pointed to by zero-indexed index. setable.

(subseq sequence start [end])

▷ Return subsequence of sequence between start and end. setable.

(sort sequence test [key function])

▷ Return sequence sorted. Order of elements considered equal is not guaranteed/retained, respectively.

(reverse sequence)

▷ Return sequence in reverse order.

(find position) foo sequence

▷ Return first element in sequence which matches foo, or its position relative to the begin of sequence, respectively.
(|find-if| test sequence| (:from-end| :start| :end| :key function|)
|find-if-not| test sequence| (:from-end| :start| :end| :key function|)
|position-if| test sequence| (:from-end| :start| :end| :key function|)
|position-if-not| test sequence| (:from-end| :start| :end| :key function|)

▷ Return first element in sequence which satisfies test, or its position relative to the begin of sequence, respectively.

(|search| sequence-a sequence-b| (:from-end| :start 1| :end 2| :key function|)

▷ Search sequence-b for a subsequence matching sequence-a. Return position in sequence-b, or NIL.

(|remove| foo sequence| (:from-end| :test function|)
|delete| foo sequence| (:from-end| :test-not function|)

▷ Make copy of sequence without elements matching foo.

(|remove-if| test sequence| (:from-end| :start 1| :end 2| :key function|)
|remove-if-not| test sequence| (:from-end| :start 1| :end 2| :key function|)
|delete-if| test sequence| (:from-end| :start 1| :end 2| :key function|)
|delete-if-not| test sequence| (:from-end| :start 1| :end 2| :key function|)

▷ Make copy of sequence with all (or count) elements satisfying test removed.

(|remove-duplicates| sequence| (:from-end| :test function|)
|delete-duplicates| sequence| (:from-end| :test-not function|)

▷ Make copy of sequence without duplicates.

(|substitute| new old sequence| (:from-end| :start 1| :end 2| :key function|)
|nsubstitute| new old sequence| (:from-end| :start 1| :end 2| :key function|)

▷ Make copy of sequence with all (or count) oleds replaced by new.

(|substitute-if| new test sequence| (:from-end| :start 1| :end 2| :key function|)
|substitute-if-not| new test sequence| (:from-end| :start 1| :end 2| :key function|)

▷ Make copy of sequence with all (or count) elements satisfying test replaced by new.

(|replace| sequence-a sequence-b| (:start 1| :start 2| :end 1| :end 2|)

▷ Replace elements of sequence-a with elements of sequence-b.

(|map| type function sequence|)

▷ Apply function successively to corresponding elements of the sequences. Return values as a sequence of type. If type is NIL, return NIL.

(|map-into| result-sequence function sequence|)

▷ Store into result-sequence successively values of function applied to corresponding elements of the sequences.

(|reduce| function sequence| (:from-end| :start 1| :end 2| :key function|)

▷ Starting with the first two elements of sequence, apply function successively to its last return value together with the next element of sequence. Return last value of function.

(|copy-seq| sequence|)

▷ Copy of sequence with shared elements.

### 7 Hash Tables

The Loop Facility provides additional hash table-related functionality; see loop, page 22.

Key-value storage similar to hash tables can as well be achieved using association lists and property lists; see pages 10 and 17.

(|hash-table-p| foo|)

▷ Return T if foo is of type hash-table.

(|make-hash-table| size int| :rehash-size num| :rehash-threshold num|)

▷ Make a hash table.

(|gethash| key hash-table [default|)

▷ Return object with key if any or default otherwise; and NIL if found, NIL otherwise. settable.

(|hash-table-count| hash-table|)

▷ Number of entries in hash-table.

(|rehash| hash-table|)

▷ Remove from hash-table entry with key and return T if it existed. Return NIL otherwise.

(|maphash| function hash-table|)

▷ Iterate over hash-table calling function on key and value. Return NIL.

(|with-hash-table-iterator| foo hash-table| (declare (doc+) form+)

▷ Return values of forms. In forms, invocations of (foo) return: T if an entry is returned; its key; its value.

(|hash-table-test| hash-table|)

▷ Test function used in hash-table.

(|hash-table-size| hash-table|)

▷ Current size, rehash-size, or rehash-threshold, respectively, as used in make-hash-table.

(|sxhash| foo|)

▷ Hash code unique for any argument, equal foo.
(#\symbol-macrolet ((foo expansion-form*) (declare decl*) form*)
  \(\triangleright\) Evaluate forms with locally defined symbol macros foo.

(#\defsetf function
  updater [doc]
  ((self-\textlambda*) (s-var*) (declare decl*) [\textcode{doc}] form*)
  where \(\text{defsetf}\) lambda list \((\text{self-}\lambda*)\) has the form
  \(\text{var}^{+} \text{ [optional] [var \ [init \ [\text{supplied-p}]]]} \) \[&ALT]
  \([\text{var}^{+} \text{ [optional] [var \ [\text{key} \ [\text{var} \ [\text{key} \ \text{supplied-p}]]]}]} \) \[&ALT]
  \([\text{rest} \ \text{var}] \text{ [optional] [\text{var} \ [\text{rest} \ \text{var} \ [\text{key} \ [\text{key} \ \text{supplied-p}]]]}]} \) \[&ALT]
  \([\text{allow-other-keys}] \text{ [optional] [\text{environment} \ \text{var}]} \) \[&ALT]
  \(\text{\triangleright\triangleright\triangleright\reduce\triangleright\triangleright}\) Specify how to \(\text{setf}\) a place accessed by \(\text{function}\).
  Short form: \((\text{setf} \ \text{function} \ \text{arg}^{*}) \) value-form is replaced by \((\text{ updater \ arg}^{*} \ \text{value-form})\); the latter must return value-form.
  Long form: on invocation of \((\text{setf} \ \text{function} \ \text{arg}^{*}) \) value-form, forms must expand into code that sets the place accessed \((\text{self-}\lambda\text{ and s-var}^{*}\text{ describe the arguments of function and the value(s) to be stored, respectively; and that returns the value(s) of s-var}^{*}\text{. forms are enclosed in an implicit \textquote{block} named function.}}\)

(#\define-setf-expander function (macro-\textlambda*) (declare decl*) [doc] form*)
  \(\text{\triangleright\triangleright\triangleright\reduce\triangleright\triangleright}\) Specify how to \(\text{setf}\) a place accessed by \(\text{function}\). On invocation of \((\text{setf} \ \text{function} \ \text{arg}^{*}) \) value-form, forms must expand into code returning arg-vars, args, newval-vars, set-form, and get-form as described with \((\text{get-setf-expansion})\) where the elements of macro lambda list macro-\(\lambda^{*}\) are bound to corresponding \(\text{args}\). forms are enclosed in an implicit \textquote{block} named function.

(#\get-setf-expansion place [environment \textquote{doc}]
  \(\text{\triangleright\triangleright\triangleright\reduce\triangleright\triangleright}\) Return lists of temporary variables arg-vars and of corresponding args as given with \(\text{place}\). \text{newval-vars} with temporary variables corresponding to the new values, and \(\text{set-form}\) and \(\text{get-form}\) specifying in terms of \(\text{arg-vars}\) and \text{newval-vars} how to \(\text{setf}\) and how to \text{read place.}

(#\define-modify-macro foo ([\text{optional} \text{var} \ [\text{init} \ [\text{supplied-p}]]]) \[&ALT]
  \([\text{var} \ [\text{key} \ [\text{key} \ \text{supplied-p}]]) \) \[&ALT]
  \([\text{rest} \ \text{var}] \text{ [optional] [\text{var} \ [\text{rest} \ \text{var} \ [\text{key} \ [\text{key} \ \text{supplied-p}]]]}]) \) \[&ALT]
  \([\text{allow-other-keys}] \text{ [optional] [\text{environment} \ \text{var}]} \) \[&ALT]
  \(\text{\triangleright\triangleright\triangleright\reduce\triangleright\triangleright}\) Define macro \(\text{foo}\) able to modify a place. On invocation of \((\text{foo \ place \ arg}^{*})\), the value of \(\text{function} \) applied to \(\text{place} \) and \text{args} will be stored into \text{place} and returned.

\(\text{\textquote{lambda-list-keywords}}\)
  \(\text{\triangleright}\) List of macro lambda list keywords. These are at least:

  \&\textit{whole} var
  \(\text{\triangleright}\) Bind \textit{var} to the entire macro call form.

  \&\textit{optional} var*
  \(\text{\triangleright}\) Bind \textit{var}s to corresponding arguments if any.

  \{\text{rest}\{\text{keybody}\} \text{var}
  \(\text{\triangleright}\) Bind \textit{var}s to a list of remaining arguments.

  \&\textit{key} var*
  \(\text{\triangleright}\) Bind \textit{var}s to corresponding keyword arguments.

  \&\textit{allow-other-keys}
  \(\text{\triangleright}\) Suppress keyword argument checking. Callers can do so using \&\textit{allow-other-keys} \(\text{T}\).

  \&\textit{environment} var
  \(\text{\triangleright}\) Bind \textit{var} to the lexical compilation environment.

  \&\textit{aux} var*
  \(\text{\triangleright}\) Bind \textit{var}s as in \textit{\textquote{let}}.

\(\text{\textquote{9.2\ Variables}}\)
\(\text{\triangleright\triangleright\triangleright\reduce\triangleright\triangleright}\)

(#\deconstant \text{foo} form \textquote{doc}]
  \(\text{\triangleright}\) Assign value of \textit{form} to global constant/dynamic variable \textit{foo}.

(#\deparameter \text{foo} form [doc])
  \(\text{\triangleright}\) Unless bound already, assign value of \textit{form} to dynamic variable \textit{foo}.

(#\setf \text{foo} \textquote{form} \textquote{doc}]
  \(\text{\triangleright}\) Set \textit{places} to primary values of \textit{forms}. Return values of \textit{last form}/NIL; work sequentially/in parallel, respectively.

(#\setq \text{symbol} \textquote{form} \textquote{doc}]
  \(\text{\triangleright}\) Set symbols to primary values of \textit{forms}. Return value of \textit{last form}/NIL; work sequentially/in parallel, respectively.

(#\set symbol \text{foo} \text{\triangleright}\) Set symbol’s value cell to \textit{foo}. Deprecated.

(#\multiple-value-setq \text{vars} \text{form})
  \(\text{\triangleright}\) Set elements of \textit{vars} to the values of \textit{forms}. Return \textit{forms’} primary value.

(#\shift \text{place} \text{foo} \text{\triangleright}\) Store value of \textit{foo} in rightmost \textit{place} shifting values of \textit{places} left, returning first \textit{place}.

(#\rotate \text{place} \text{\triangleright}\) Rotate values of \textit{places} left, old first becoming new last \textit{place}’s value. Return NIL.

(#\makunbound \text{foo} \text{\triangleright}\) Delete special variable \textit{foo} if any.

(#\get symbol \text{key} \textquote{default} \textquote{doc}]
  \(\text{\triangleright}\) First entry \textit{key} from property list stored in \textit{symbol}/in \textit{place}, respectively, or \textit{default} if there is no \textit{key}. \textit{Setable}.

(#\get-properties \text{property-list} \text{keys} \text{\triangleright}\) Return key and value of first entry from \textit{property-list} matching a \textit{key} from \textit{keys}, and tail of \textit{property-list} starting with that \textit{key}. Return NIL, NIL, and NIL if there was no matching key in \textit{property-list}.

(#\remprop \text{symbol} \text{key} \text{\triangleright}\) Remove first entry \textit{key} from property list stored in \textit{symbol}/in \textit{place}, respectively. Return \text{T} if \textit{key} was there, or NIL otherwise.

(#\progv \text{symbols} \text{values} \text{\textquote{form}} \textquote{doc}]
  \(\text{\triangleright}\) Evaluate \textit{forms} with \textit{names} lexically bound (in parallel or sequentially, respectively) to \textit{values}. Return values of \textit{forms}.
9.3 Functions

Below, ordinary lambda list (\(\text{ord-}\lambda^*\)) has the form

\[
\text{function form} \Rightarrow \text{body-form}^n
\]

\(\leadsto\) Evaluate body-forms with vars lexically bound to the return values of values-form. Return values of body-forms.

\(\text{destructuring-bind destruct-\lambda\ \text{bar function form}^b}\)

\(\leadsto\) Evaluate forms with variables from tree destruct-\lambda bound to corresponding elements of tree bar, and return their values. destruct-\lambda resembles macro-\lambda (section 9.4), but without any \&environment clause.

\(\var\) supplied-p is \(\text{T}\) if there is a corresponding argument. init forms can refer to any init and supplied-p to their left.

\[
\text{mdeff} \Rightarrow \text{mlambda} \Rightarrow \text{mdefun} \Rightarrow \text{msetf}
\]

\(\Rightarrow\) Define a function named foo or (setf foo), or an anonymous function, respectively, which applies forms to ord-\(\lambda\). For mdeff, forms are enclosed in an implicit \(\text{block}\) named foo.

\[
\text{mlet} \Rightarrow \text{mlabels}
\]

\(\Rightarrow\) Evaluate forms with locally defined functions foo. Globally defined functions of the same name are shadowed. Each foo is also the name of an implicit \(\text{block}\) around its corresponding local-forms. Only for \text{mlabels}, functions foo are visible inside local-forms. Return values of forms.

\(\text{function}\)

\(\Rightarrow\) Return lexically innermost function named foo or a lexical closure of the \text{lambda expression}.

\(\text{apply}\)

\(\Rightarrow\) Values of function called with args and the list elements of args. \text{setf} able if function is one of \(\text{aref}, \text{rbit}, \text{and sbit} \).

\(\text{funcall}\)

\(\Rightarrow\) Values of function called with args.

\(\text{multiple-value-call}\)

\(\Rightarrow\) Call function with all the values of each form as its arguments. Return values returned by function.

\(\text{values-list}\)

\(\Rightarrow\) Return elements of list.

\(\text{values foo}\)

\(\Rightarrow\) Return as multiple values the primary values of the foo. \text{setf} able.

\(\text{multiple-value-list}\)

\(\Rightarrow\) List of the values of form.

\(\text{nth-value}\)

\(\Rightarrow\) Zero-indexed nth return value of form.

\(\text{complement}\)

\(\Rightarrow\) Return new function with same arguments and same side effects as function, but with complementary truth value.

\(\text{constantly foo}\)

\(\Rightarrow\) Function of any number of arguments returning foo.

\(\text{identity foo}\)

\(\Rightarrow\) Return foo.

\(\text{function-lambda-expression}\)

\(\Rightarrow\) If available, return lambda expression of function, \text{NIL} if function was defined in an environment without bindings, and name of function.

\(\text{definition}\)

\(\Rightarrow\) Definition of global function foo. \text{setf} able.

\(\text{makunbound foo}\)

\(\Rightarrow\) Remove global function or macro definition foo.

\(\text{call-arguments-limit}\)

\(\Rightarrow\) Upper bound of the number of function arguments or lambda list parameters, respectively, \(\geq 50\).

\(\text{multiple-values-limit}\)

\(\Rightarrow\) Upper bound of the number of values a multiple value can have, \(\geq 20\).

9.4 Macros

Below, macro lambda list (macro-\(\lambda^*\)) has the form of either

\[
\begin{align*}
\text{(&whole}\ \text{var}\ [\text{E}] &\Rightarrow [\text{var\ (macro-}\lambda^*)]\ [\text{E}] \\
\text{&optional}\ &\Rightarrow \text{var}\ [\text{E}\ (\text{macro-}\lambda^*)]\ [\text{E}] \\
\text{&key}\ &\Rightarrow \text{var\ (macro-}\lambda^*)\ [\text{E}] \\
\text{&body}\ &\Rightarrow \text{var\ (macro-}\lambda^*)\ [\text{E}] \\
\text{&allow-other-keys}\ &\Rightarrow \text{var\ (macro-}\lambda^*)\ [\text{E}] \\
\end{align*}
\]

\(\Rightarrow\) One toplevel \(\text{E}\) may be replaced by \&environment var. supplied-p is \(\text{T}\) if there is a corresponding argument. init forms can refer to any init and supplied-p to their left.

\[
\begin{align*}
\text{defmacro}\ &\Rightarrow \text{define-complier-macro}\Rightarrow \text{foo}\ [\text{E}]\ (\text{macro-}\lambda^*)\ (\text{declare}\ \text{decl}^*) \Rightarrow \text{doc}\ [\text{E}] \\
\text{setf}\ &\Rightarrow \text{Define macro foo which on evaluation as (foo tree) applies expanded forms to arguments from tree, which corresponds to tree-shaped macro-\text{lambda}. forms are enclosed in an implicit \text{block} named foo.} \\
\end{align*}
\]

\(\text{define-symbol-macro}\)

\(\Rightarrow\) Define symbol macro foo which on evaluation evaluates expanded form.

\(\text{macrolet}\)

\(\Rightarrow\) Evaluate forms with locally defined mutually invisible macros foo which are enclosed in implicit \text{blocks} of the same name.
9.5 Control Flow

\(\text{if test then } \text{then-bar}\)  
\(\text{return values of then if test returns } T; \text{return values of else otherwise.}\)

\(\text{when (test then-bar)}\)  
\(\text{return the values of the first then\textquotesingle s whose test returns } T; \text{return NIL if all tests return NIL.}\)

\(\text{when (test foo)}\)  
\(\text{evaluate foo\textquotesingle s and return their values if test returns } T \text{or NIL, respectively. Return NIL otherwise.}\)

\(\text{case (test (key))}\)  
\(\text{return values of the first foo\textquotesingle s one of whose keys is eql test. Return values of bar if there is no matching key.}\)

\(\text{and form\textquotesingle s}\)  
\(\text{evaluate forms from left to right. Immediately return NIL if one form\textquotesingle s value is NIL. Return values of last form otherwise.}\)

\(\text{or form\textquotesingle s}\)  
\(\text{evaluate forms from left to right. Immediately return primary value of first non-NIL-evaluating form, or all values if last form is reached. Return NIL if no form returns } T.\)

\(\text{progn form\textquotesingle s}\)  
\(\text{evaluate forms sequentially. Return values of last form.}\)

\(\text{multiple-value-prog1 form-r form\textquotesingle s}\)  
\(\text{multiple-value-prog2 form-r form\textquotesingle s}\)  
\(\text{multiple-value-prog2 form-r form\textquotesingle s}\)  
\(\text{evaluate forms in order. Return values/primary value, respectively, of form-r.}\)

\(\text{unwind-protect protected cleanup\textquotesingle s}\)  
\(\text{evaluate protected and then, no matter how control leaves protected, cleanups. Return values of protected.}\)

\(\text{block name form\textquotesingle s}\)  
\(\text{evaluate forms in a lexical environment, and return their values unless interrupted by return-from.}\)

\(\text{return-from foo result}\)  
\(\text{have nearest enclosing block named foo/named NIL, respectively, return with values of result.}\)

\(\text{tagbody (tag form\textquotesingle s)}\)  
\(\text{evaluate forms in a lexical environment. tags (symbols or integers) have lexical scope and dynamic extent, and are targets for } \text{go. Return NIL.}\)

\(\text{go tag}\)  
\(\text{within the innermost possible enclosing tagbody, jump to a tag } \text{eql tag.}\)
(catch tag form)
▷ Evaluate forms and return their values unless interrupted by throw.

(throw tag form)
▷ Have the nearest dynamically enclosing catch with a tag equal to tag return with the values of form.

(sleep n) ▷ Wait n seconds; return NIL.

9.6 Iteration

{do m do{ [var start step] } (stop result) (declare deccl)*
▷ Evaluate tagbody-like body with vars successively bound according to the values of the corresponding start and step forms. vars are bound in parallel/sequentially, respectively. Stop iteration when stop is T. Return values of result*. Implicitly, the whole form is a block named NIL.

(dotimes (var i [result]) (declare deccl)* {tag form})*
▷ Evaluate tagbody-like body with var successively bound to integers from 0 to i – 1. Upon evaluation of result, var is i. Implicitly, the whole form is a block named NIL.

(dolist (var list [result]) (declare deccl)* {tag form})*
▷ Evaluate tagbody-like body with var successively bound to the elements of list. Upon evaluation of result, var is NIL. Implicitly, the whole form is a block named NIL.

9.7 Loop Facility

(loop form)*
▷ Simple Loop. If forms do not contain any atomic Loop Facility keywords, evaluate them forever in an implicit block named NIL.

(loop clause)*
▷ Loop Facility. For Loop Facility keywords see below and Figure 1.

named n ▷ Give loop’s implicit block a name.

(with {var-s} {var-s=} [d-type] [foo]*)
▷ Begin of iteration control clauses. Initialize and step (possibly trees of) local variables var-s sequentially and var-p in parallel.

(for as) {var-s} [d-type] [foo] )
▷ Specify form as the end value for stepping.

(in on) list
▷ Bind var to successive elements/tails, respectively, of list.

by {step function}
▷ Specify the (positive) decrement or increment or the function of one argument returning the next part of the list.

Figure 1: Loop Facility, Overview.
10.3 Method Combination Types

standard
▷ Evaluate most specific :around method supplying the values of the generic function. From within this method, \texttt{call-next-method} can call less specific :around methods if there are any. If not, or if there are no :around methods at all, call all \texttt{before} methods, most specific first, and the most specific primary method which supplies the values of the calling \texttt{call-next-method} if any, or of the generic function; and which can call less specific primary methods via \texttt{call-next-method}. After its return, call all \texttt{after} methods, least specific first.

\texttt{append list conc prog max min +}
▷ Simple built-in \texttt{method-combination} types; have the same usage as the \texttt{c-type} defined by the short form of \texttt{define-method-combination}.

\texttt{(define-method-combination \texttt{c-type}}
\begin{verbatim}
[(documentation string)
 [identity-with-one-argument bool]]
\end{verbatim}
▷ Short Form. Define new method-combination \texttt{c-type}. In a generic function using \texttt{c-type}, evaluate most specific :around method supplying the values of the generic function. From within this method, \texttt{call-next-method} can call less specific :around methods if there are any. If not, or if there are no :around methods at all, return from the calling \texttt{call-next-method} or from the generic function, respectively, the values of \texttt{operator \{primary-method \texttt{gen-arg} \texttt{*}\)}, \texttt{gen-arg} being the arguments of the generic function. The primary-methods are ordered \{\texttt{most-specific-first} \texttt{most-specific-last}\} (specified as \texttt{c-arg} in \texttt{defgeneric}). Using \texttt{c-type} as the qualifier in \texttt{make-method} makes the method primary.

\texttt{(define-method-combination \texttt{c-type \{ord-\texttt{a}\}})}
\begin{verbatim}
[\{qualifier* \texttt{[]}\} \texttt{predicate}]
\end{verbatim}
▷ Long Form. Define new method-combination \texttt{c-type}. A call to a generic function using \texttt{c-type} will be equivalent to a call to the forms returned by \texttt{body*} with \texttt{ord-\texttt{a}} bound to \texttt{c-arg*} (cf. \texttt{defgeneric}), with symbol bound to the generic function, \texttt{method-combination-\texttt{a}} bound to the arguments of the generic function, and with \texttt{group} bound to lists of methods. An applicable method becomes a member of the leftmost \texttt{group} whose \texttt{predicate} or \texttt{qualifiers} match. Methods can be called via \texttt{call-method}. Lambda lists \texttt{(ord-\texttt{a})} and \texttt{(method-combination-\texttt{a})} according to \texttt{ord-\texttt{a}} on page 18, the latter enhanced by an optional \texttt{&whole} argument.

\texttt{(call-method}
\begin{verbatim}
[\texttt{method \{make-method-form\}}]
\end{verbatim}
▷ From within an effective method form, call \texttt{method} with the arguments of the generic function and with information about its \texttt{next-methods}; return its \texttt{values}.

11 Conditions and Errors

For standardized condition types cf. Figure 2 on page 32.
(change-class instance new-class \{initarg value\} \{other-keyarg\})
▷ Change class of instance to new-class. Retain the status of any slots that are common between instance's original class and new-class. Initialize any newly added slots with the values of the corresponding initargs if any, or with the values of their :initform forms if not.

(make-instances-obsolete class)
▷ Update all existing instances of class using update-instance-for-redefined-class.

(update-instance-for-redefined-class new-instance added-slots discarded-slots discarded-slots-property-list \{initarg value\} \{other-keyarg\})
▷ Set slots on behalf of make-instance of change-class by means of shared-initialize.

(update-instance-for-redefined-class new-instance added-slots discarded-slots discarded-slots-property-list \{initarg value\} \{other-keyarg\})
▷ On behalf of make-instances-obsolete and by means of shared-initialize, set any initarg slots to their corresponding values; set any remaining added-slots to the values of their :initform forms. Not to be called by user.

allocate-instance class \{initarg value\} \{other-keyarg\})
▷ Return uninitialized instance of class. Called by make-instance.

(shared-initialize instance \{initform-slots\} \{initarg-slot value\} \{other-keyarg\})
▷ Fill the initarg-slots of instance with the corresponding values, and fill those initform-slots that are not initarg-slots with the values of their :initform forms.

(slot-missing class instance slot \{setf\} \{slot-boundp\} \{slot-makunbound\} \{slot-value\})
▷ Called on attempted access to non-existing or unbound slot. Default methods signal error/unbound-slot, respectively. Not to be called by user.

10.2 Generic Functions

(next-method-p)
▷ T if enclosing method has a next method.

(defgeneric \{foo\} \{setf foo\} \{required-var\} \{optional\} \{var\} \{\texttt{&rest} var\} \{\texttt{&key} var\} \{\texttt{&allow-other-keys}\})
\{(argument-precedence-order required-var\textsuperscript{+})
\{declare \{optimize \textit{method-selection-optimization}\\textsuperscript{+}\}
\{documentation string\}
\{generic-function-class \textit{gf-class}\}
\{method-class \textit{method-class}\}
\{method-combination \textit{c-type} \textit{c-arg}\}
\{lambda-list lambda-list\}
\{environment environment\}
▷ Define or modify generic function \textit{foo}. \textit{gf-class} and lambda-list must be compatible with a pre-existing generic function or with existing methods, respectively. Changes to method-class do not propagate to existing methods. For c-type see section 10.3.

(defunmethod \{foo\} \{setf foo\} \{before\} \{after\} \{around\} \{qualifier\})

(add-method \{remove-method\} generic-function method)
▷ Add (if necessary) or remove (if any) method to/from generic-function.

(find-method generic-function qualifiers specializers \{error\})
▷ Return suitable method, or signal error.

(compute-applicable-methods generic-function args)
▷ List of methods suitable for args, most specific first.

(call-next-method args \{call-next-method\})
▷ From within a method, call next method with args; return its values.

(no-applicable-method generic-function arg*)
▷ Called on invocation of generic-function on args if there is no applicable method. Default method signals error. Not to be called by user.

(invalid-method-error method \{method-combination-error\})
\{control arg\}
▷ Signal error on applicable method with invalid qualifiers, or on method combination. For control and args see format, page 38.

(no-next-method generic-function method args)
▷ Called on invocation of call-next-method when there is no next method. Default method signals error. Not to be called by user.

(function-keywords method)
▷ Return list of keyword parameters of method and T if other keys are allowed.

(method-qualifiers method)
▷ List of qualifiers of method.
Every type is also a supertype of NIL, the empty type.
The object not readily printable under condition.

Package, path, or stream, respectively, which caused the condition of indicated type.

Object which caused condition of type type-error, or its expected type, respectively.

Return format control or list of format arguments, respectively, of condition.

Condition type debugger is to be invoked on.

Function of condition and function itself. Called before debugger.

12 Types and Classes

For any class, there is always a corresponding type of the same name.

Return values of forms if restart is called during their evaluation. In this case, describe restart using format control and args (see page 38) and return NIL and T.

Return values of form or, if during evaluation of form one of the dynamically established restarts is called, the values of its restart-forms. A restart is visible under condition if (funcall #\"test-function condition\") returns T. If presented in the debugger, restarts are described by string or by #\"report-function\" (of a stream). A restart can be called by (invoke-restart restart arg*), where args match ord-\* or by (invoke-restart-interactively) where a list of the respective args is supplied by arg-function. See page 18 for ord-\*.

Return values of forms evaluated with dynamically established restarts whose restart-functions should perform a non-local transfer of control. A restart is visible under condition if (test-function condition) returns T. If presented in the debugger, restarts are described by restart-function (of a stream). A restart can be called by (invoke-restart restart arg*), where args must be suitable for the corresponding restart-function, or by (invoke-restart-interactively) where a list of the respective args is supplied by arg-function.

Call function associated with restart with arguments given or prompted for, respectively. If restart function returns, return its values.

Return innermost restart name, or a list of all restarts, respectively, out of those either associated with condition or un-associated at all; or, without condition, out of all restarts. Return NIL if search is unsuccessful.

Name of restart.

Transfer control to innermost applicable restart with same name (i.e. abort, ..., continue ...) out of those either associated with condition or un-associated at all; or, without condition, out of all restarts. If no restart is found, signal control-error for abort and muffle-warning, or return NIL for the rest.

Evaluate forms with restarts dynamically associated with condition. Return values of forms.

List of function or of its operands respectively, used in the operation which caused condition.

Name of cell which caused condition.

Instance with unbound slot which caused condition.

The object not readily printable under condition.

Package, path, or stream, respectively, which caused the condition of indicated type.

Object which caused condition of type type-error, or its expected type, respectively.

Return format control or list of format arguments, respectively, of condition.

Condition type debugger is to be invoked on.

Function of condition and function itself. Called before debugger.
#+feature when-feature
-#- feature unless-feature
▷ Means when-feature if feature is T; means unless-feature if feature is NIL. feature is a symbol from , #+features, or {(and |or) feature *}, or (not feature).

#+features
▷ List of symbols denoting implementation-dependent features.

|c*|: \c
▷ Treat arbitrary character(s) c as alphabetic preserving case.

### 13.4 Printer

(print | /prin1 | /print | /pprint | /princ)

| foo [stream | \standard-output]
▷ Print foo to stream, readably, readably between a newline and a space, readably after a newline, or human-readably without any extra characters, respectively. /prin1, /print and /princ return foo.

(print-to-string foo)

(prin-to-string foo)
▷ Print foo to string, readably or human-readably, respectively.

(print object object stream)
▷ Print object to stream. Called by the Lisp printer.

(print-unreadable-object (foo ~ stream ([:type boolean identity boolean]) form *))
▷ Enclosed in #< and >, print foo by means of forms to stream. Return NIL.

(terpri [stream | \standard-output])
▷ Output a newline to stream. Return NIL.

(fresh-line | \standard-output)
▷ Output a newline to stream and return T unless stream is already at the start of a line.

(write-char char [stream | \standard-output])
▷ Output char to stream.

(write-string string [stream | \standard-output] [:start start | end end])
▷ Write string to stream without/with a trailing newline.

(write-byte byte stream)
▷ Write byte to binary stream.

(write-sequence sequence stream [:start start | end end])
▷ Write elements of sequence to binary or character stream.

### 13 Input/Output

#### 13.1 Predicates

(streamp foo)

(pathnamep foo) ▷ T if foo is of indicated type.

(readablep foo)

(input-stream-p stream)

(output-stream-p stream)

(interactive-stream-p stream)

(open-stream-p stream)
▷ Return T if stream is for input, for output, interactive, or open, respectively.

(pathname-match-p path wildcard)
▷ T if path matches wildcard.

(wild-pathname-p path [:host :device :directory :name :type :version | NIL])
▷ Return T if indicated component in path is wildcard. (NIL indicates any component.)

#### 13.2 Reader

(y-or-n-p \control arg *)
▷ Ask user a question and return T or NIL depending on their answer. See page 38, /format, for control and args.

(with-standard-io-syntax form *)
▷ Evaluate forms with standard behaviour of reader and printer. Return values of forms.

(read | read-preserving-whitespace | \standard-input | \eof-error | \eof-value | \recursive))
▷ Read printed representation of object.

(\read-from-string string \eof-error \eof-value | [:start start | end end | \preserve-whitespace \boolean])
▷ Return object read from string and zero-indexed position of next character.
13.3 Character Syntax

| multi-line-comment* | # |
| one-line-comment* |
| foo | (quote foo); foo unevaluated.
| (foo [baz | [quux | [bing]) |
| Backquote. ;quote foo and bing; evaluate bar and splice the lists baz and quux into their elements. When nested, outermost commas inside the innermost backquote expression belong to this backquote.

| \c | (character "c"), the character c.
| \B; | #O(n); #Xn; #Rn |
| n/d | Integer of radix 2, 8, 16, 10, or r; 2 ≤ r ≤ 36.

| \n | The ratio \frac{m}{n}.
| \{m\} | m \cdot 10^n as short-float, single-float, double-float, long-float, or the type from *read-default-float-format*.

| \(a b) | (\text{complex } a + bi).
| \#foo | (function foo); the function named foo.
| \#sequence | n-dimensional array.
| \[n\] | Vector of some (or n) foos filled with last foo if necessary.
| \[n\] | Bit vector of some (or n) bs filled with last b if necessary.

| \{type \{slot value\}\} | Structure of type.
| \string | A pathname.
| #:foo | Uninterned symbol foo.
| #form | Read-time value of form.
| \*read-eval | If NIL, a reader-error is signalled at #\*.
| #integer = foo | Give foo the label integer.
| #integer | Object labelled integer.
| #$ | Have the reader signal reader-error.
13.6 Streams

- \{- n \} \{ \text{- \} \}
  \> Indent. Set indentation to \text{n} relative to leftmost/to current position.
- \{- n \} \{ \text{- \} \}
  \> Tabulate. Move cursor forward to column number \text{c + ki}, \text{k \geq 0} being as small as possible. With \text{;}, calculate column numbers relative to the immediately enclosing section. With \text{O}, move to column number \text{c0 + c + ki} where \text{c0} is the current position.
- \{- n \} \{ \text{- \} \}
  \> Go-To. Jump \text{m} arguments forward, or backward, or to argument \text{n}.
- \{- limit \} \{ \text{- \} \}
  \> Iteration. Use \text{text} repeatedly, up to \text{limit}, as control string for the elements of the list argument or (with \text{0}) for the remaining arguments. With \text{; or \text{0}}, list elements or remaining arguments should be lists of which a new one is used at each iteration step.
- \{ x \} \{ \text{- \} \}
  \> Escape Upward. Leave immediately \text{< - >}, \text{< - : >}, \text{- } \{ \text{- \} \}, \text{- \}, or the entire \text{format} operation. With one to three prefixes, act only if \text{x} = \text{0}, \text{x} = \text{y}, or \text{x} \leq \text{y} \leq \text{z}, respectively.
- \{ i \} \{ \text{- \} \}
  \> Conditional Expression. Use the zero-indexed argument (i+th if given) \text{text} as a \text{format} control subclause. With \text{;}, use the first text if the argument value is \text{NIL}, or the second text if it is \text{T}. With \text{O}, do nothing for an argument value of \text{NIL}. Use the only \text{text} and leave the argument to be read again if it is \text{T}.
- \text{- loop}
  \> Recursive Processing. Process two arguments as control string and argument list, or take one argument as control string and use then the rest of the original arguments.
- \{ prefix \} \{ \text{- \} \}
  \> Call Function. Call all-upper case \text{package: function} with the arguments stream, format-argument, colon-p, at-sign-p and \text{prefixes} for printing format-argument.
- \{ \text{- \} \}
  \> Write. Print argument of any type obeying every printer control variable. With \text{;}, pretty-print. With \text{O}, print without limits on length or depth.

\{ \text{- \} \}
\> Print \text{foo} to \text{stream} and return \text{foo}, or print \text{foo} into string, respectively, after dynamically setting printer variables corresponding to keyword parameters (\text{print-bar \text{\text{*}}} becoming \text{:bar}.

\{ \text{- \} \}
\> \text{Print \text{foo} to \text{stream}. If \text{foo} is a list, print as many elements per line as possible; do the same in a table with a column width of \text{n} ems; or print either all elements on one line or each on its own line, respectively. Return \text{NIL}. Use \text{format} directive \text{-/-}.

\{ \text{- \} \}
\> \text{Evaluate \text{forms}, which should print list, with \text{stream} locally bound to a pretty printing stream which outputs to the original \text{stream}. If \text{list} is in fact not a list, it is printed by \text{write}. Return \text{NIL}.

\{ \text{- \} \}
\> \text{Take next element of \text{list}. If there is no remaining tail of \text{list}, or \text{print-length} or \text{print-circles} indicate printing should end, send element together with an appropriate indicator to stream.

\{ \text{- \} \}
\> \text{Move cursor forward to column number \text{c + ki}, \text{k \geq 0} being as small as possible.

\{ \text{- \} \}
\> \text{If \text{list} is empty, terminate logical block. Return \text{NIL} otherwise.

\{ \text{- \} \}
\> \text{Print a conditional newline if \text{stream} is a pretty printing stream. Return \text{NIL}.

\{ \text{- \} \}
\> \text{If \text{T}, print arrays \text{readably}.

\{ \text{- \} \}
\> \text{Radix for printing rationals, from 2 to 36.}
13.5 Format

(set-pprint-dispatch type function [priority]
 [table (print-pprint-dispatch)])
 Install entry comprising function of arguments stream and object to print; and priority as type into table. If function is NIL, remove type from table. Return NIL.

(print-pprint-dispatch foo [table (print-pprint-dispatch)])
 Return highest priority function associated with type of foo and T if there was a matching type specifier in table.

(copy-pprint-dispatch [table (print-pprint-dispatch)])
 Return copy of table or, if table is NIL, initial value of +*print-pprint-dispatches*.

Current pretty print dispatch table.

{-R-:R,-OR,-OR:-R:}
➤ Roman. Take argument as number and print it as English cardinal number, as English ordinal number, as Roman numeral, or as old Roman numeral, respectively.

[width] [[; pad-char & ][; comma-char \n \noslash{\text{comma-interval}}]] \oslash 0 (\oslash[0])
➤ Decimal/Binary/Octal/Hexadecimal. Print integer argument as number. With \oslash, group digits comma-interval each; with 0, always prepend a sign.

[width] [[(dec-digits) \n \noslash{\text{shift}} \n \noslash{\text{overflow-char}} \n \noslash{\text{pad-char}} \n ]] \oslash F
➤ Fixed-Format Floating-Point. With 0, always prepend a sign.

[width] [[(dec-digits) \n \noslash{\text{exp-digits}} \n \noslash{\text{scale-factor}} \n \noslash{\text{overflow-char}} \n \noslash{\text{pad-char}} \n ]] \oslash [E C]
➤ Exponential/General Floating-Point. Print argument as floating-point number with dec-digits after decimal point and exp-digits in the signed exponent. With -G, choose either -E or -F. With 0, always prepend a sign.

[(dec-digits) \n \noslash{\text{int-digits}} \n \noslash{\text{width}} \n \noslash{\text{pad-char}} \n ]] \oslash S
➤ Monetary Floating-Point. Print argument as fixed-format floating-point number. With \oslash, put sign before any padding; with 0, always prepend a sign.

{C-:-C:-0C:-0-C:}
➤ Character. Print, spell out, print in \#\ syntax, or tell how to type, respectively, argument as (possibly non-printing) character.

\{- (text -) \nobreak{-} (text -) \nobreak{-} (text -) \nobreak{-} (text -) \}
➤ Case-Conversion. Convert text to lowercase, convert first letter of each word to uppercase, capitalize first word and convert the rest to lowercase, or convert to uppercase, respectively.

{P-:-P:-0P:-0-P:}
➤ Plural. If argument eql 1 print nothing, otherwise print a; do the same for the previous argument; if argument eql 1 print y, otherwise print x; do the same for the previous argument, respectively.

[r: %]
➤ Newline. Print n newlines.

[r: &]
➤ Fresh-Line. Print n newlines at the beginning of a line, or n newlines otherwise.

{\-1: \-0: \-1:}
➤ Conditional Newline. Print a newline like pprint-newline with argument \dline, \fill, \miser, or \mandatory, respectively.

\{-1: \-0: \-\}
➤ Ignored Newline. Ignore newline, or whitespace following newline, or both, respectively.

[r: ]
➤ Page. Print n page separators.

[r: -]
➤ Tilde. Print n tildes.

\[\]"[\text{col-in}] \n \[\text{min-pcln}] \n \[\text{pad-char} \n \]
➤ Aesthetic/Standard. Print argument of any type for consumption by humans/\by the reader, respectively. With \[, print NIL as () rather than NIL; with \], add pad-chars on the left rather than right.

\[\]"[\text{width}] \n \[\text{pad-char} \n \[\text{comma-char} \n \noslash{\text{comma-interval}}\noslash{\text{width}}\noslash{\text{pad-char}}\]
➤ Radix. (With one or more prefix arguments.) Print argument as number; with \[, group digits comma-interval each; with 0, always prepend a sign.

\[\]"[\text{width}] \n \[\text{pad-char} \n \[\text{comma-char} \n \noslash{\text{comma-interval}}\noslash{\text{width}}\noslash{\text{pad-char}}\]
➤ Current pretty print dispatch table.
14.2 Packages

:bar

/package symbol

> Exported symbol of package.

/package::symbol

> Possibly unexported symbol of package.

(make-package foo

> Create package foo.

(rename-package package new-name (new-nicknames))

> Rename package. Return renamed package.

(make-package foo

> Make package foo current.

(use-package)

> Make exported symbols of other-packages available in package, or remove them from package, respectively. Return T.

(package-use-list package)

> List of other packages used by/using package.

(delete-package package)

> Delete package. Return T if successful.

(list-all-packages)

> List of registered packages.

/package-name package

> Name of package.

/package-nicknames package

> Nicknames of package.

(find-package name)

> Package with name (case-sensitive).

/find-all-symbols symbol

> List of symbols symbol from all registered packages.

/intern symbol

> Intern or find, respectively, symbol symbol in package. Second return value is one of `internal`, `external`, or `inherited` (or NIL if `intern` has created a fresh symbol).

/shadow symbol

> Remove symbol from package, return T on success.

/import symbols

> Make symbols internal to package. Return T. In case of a name conflict signal correctable `package-error` or shadow the old symbol, respectively.

/shadow symbols

> Make symbols of package shadow any otherwise accessible, equally named symbols from other packages. Return T.

/with-open-file stream path open-args* (declare decl)* form)

> Use `with-open-file` to temporarily create stream to path; return values of forms.

/with-input-from-string

> Evaluate forms with foo locally bound to input string-stream from string. Return values of forms; store next reading position into index.

/with-string-input-stream string [start] [end]

> Return a string-stream supplying the characters from string.

/with-string-output-stream [element-type type] synonyms)

> Return a string-stream accepting characters (available via `get-output-stream-string`).

/concatenated-streams

> Return list of streams concatenated-stream still has to read from/broadcast-stream is broadcasting to.

/two-way-stream-input-stream two-way-stream

> Return source stream or sink stream of two-way-stream/echo-stream, respectively.

/synonym-stream-symbol synonym-stream

> Return symbol of synonym-stream.

/get-output-stream-string string

> Clear and return as a string characters on string-stream.

/file-position stream (start end position)

> Return position within stream, or set it to position and return T on success.

/file-string-length stream

> Length foo would have in stream.

/listen stream

> T if there is a character in input stream.

/clear-input stream

> Clear input from stream, return NIL.

/finish-output stream

> End output to stream and return NIL immediately, after initiating flushing of buffers, or after flushing of buffers, respectively.

/close stream

> Close stream. Return T if stream had been open. If `:abort` is T, delete associated file.

/with-input-from-string

> Evaluate forms with foo locally bound to input string-stream from string. Return values of forms; store next reading position into index.
13.7 Pathnames and Files

(pathname-host)
(pathname-device)
(pathname-directory)
(pathname-name)
(pathname-type)
(pathname-version)

(parse-namestring 

(merge-pathnames 

(default-pathname-defaults)

(user-homedir-pathname)

Packages and Symbols

The Loop Facility provides additional means of symbol handling; see loop, page 22.

14.1 Predicates

(symbolp)

(packagep)

(keywordp)
### 45.4 Declarations

**proclaim decl**

> Globally make declaration(s) `decl`. `decl` can be: *declaration*, *type*, *ftype*, inline, notinline, `optimize`, or *special*. See below.

**declare decl**

> Inside certain forms, locally make declarations `decl`. `decl` can be: *dynamic-extent*, *type*, *ftype*, *ignore*, *inline*, *notinline*, `optimize`, or *special*. See below.

**declaration foo**

> Make foos names of declarations.

**dynamic-extent variable* (function function)**

> Declare lifetime of `variables` and/or `functions` to end when control leaves enclosing block.

**type type variable**

**ftype type function**

> Declare variables or functions to be of `type`.

**ignore**

**var**

**function function**

> Suppress warnings about used/unused bindings.

**inline**

**notinline**

> Tell compiler to integrate/not to integrate, respectively, called `functions` into the calling routine.

### 14.3 Symbols

A symbol has the attributes name, home package, property list, and optionally value (of global constant or variable name) and function (macro, or special operator name).

**make-symbol name**

> Make fresh, uninterned symbol name.

**gensym**

> Return fresh, uninterned symbol #:sn with n from `*gensym-counter*`. Increment `*gensym-counter*`.

**gentemp [prefix] [package package]**

> Intern fresh symbol in package. Deprecated.

**copy-symbol symbol [props]**

> Give copy the same value, function and property list.

**symbol-name symbol**

**symbol-package symbol**

**symbol-plist symbol**

**symbol-value symbol**

**symbol-function symbol**

> Name, package, property list, value, or function, respectively, of symbol. `setf`-able.

**documentation documentation new-doc**

> Get/set documentation string of `foo` of given type.

**package-shadowing-symbols package**

> List of symbols of package that shadow any otherwise accessible, equally named symbols from other packages.

**export symbols [package packages]**

> Make symbols external to package. Return T.

**unexport symbols [package packages]**

> Revert symbols to internal status. Return T.

**do-symbols do-external-symbols do-all-symbols (var result)**

> Evaluate `tagbody`-like body with `var` successively bound to every symbol from `package`, to every external symbol from `package`, or to every symbol from all registered packages, respectively. Return values of `result`. Implicitly, the whole form is a `block` named NIL.

**with-package-iterator (foo packages [internal external inherited])**

> Return values of `forms`. In `forms`, successive invocations of `foo` return: T if a symbol is returned; a symbol from `packages`; accessibility (`internal`, `external`, or `inherited`); and the package the symbol belongs to.

**require module [paths]**

> If not in `*modules*`, try paths to load module from. Signal error if unsuccessful. Deprecated.

**provide module**

> If not already there, add module to `*modules*`. Deprecated.

**modules**

> List of names of loaded modules.
\[\texttt{eval}\langle\texttt{form}\rangle\]  
▷ Returns values of \texttt{form} if \texttt{eval-when} is in the top-level of a file being compiled.

\[\texttt{eval-when}\langle\texttt{form}^a, \texttt{form}^b\rangle\]  
▷ Returns values of \texttt{forms} if \texttt{eval-when} is in the top-level of a file being compiled.

\[\texttt{load-time-value}\langle\texttt{form}\rangle\]  
▷ Evaluate \texttt{form} at compile time.

\[\texttt{load-time-value}\langle\texttt{read-only}\rangle\]  
▷ Evaluate \texttt{form} at compile time.

\[\texttt{quote}\langle\texttt{foo}\rangle\]  
▷ Return unevaluated \texttt{foo}.

\[\texttt{quote}\langle\texttt{expression}\rangle\]  
▷ Return unevaluated \texttt{expression}.

\[\texttt{quote}\langle\texttt{symbol}\rangle\]  
▷ Return \texttt{symbol} as literal.

\[\texttt{quote}\langle\texttt{string}\rangle\]  
▷ Return \texttt{string} as literal.

\[\texttt{quote}\langle\texttt{boolean}\rangle\]  
▷ Return \texttt{false}.

\[\texttt{quote}\langle\texttt{number}\rangle\]  
▷ Return \texttt{number} as literal.

\[\texttt{quote}\langle\texttt{symbol}\text{'}\rangle\]  
▷ Return \texttt{symbol} as literal.

\[\texttt{quote}\langle\texttt{string}\text{'}\rangle\]  
▷ Return \texttt{string} as literal.

\[\texttt{quote}\langle\texttt{boolean}\text{'}\rangle\]  
▷ Return \texttt{false}.

\[\texttt{quote}\langle\texttt{number}\text{'}\rangle\]  
▷ Return \texttt{number} as literal.

\[\texttt{quote}\langle\texttt{symbol}\rangle\]  
▷ Return \texttt{symbol} as literal.

\[\texttt{quote}\langle\texttt{string}\rangle\]  
▷ Return \texttt{string} as literal.

\[\texttt{quote}\langle\texttt{boolean}\rangle\]  
▷ Return \texttt{false}.

\[\texttt{quote}\langle\texttt{number}\rangle\]  
▷ Return \texttt{number} as literal.

\[\texttt{quote}\langle\texttt{symbol}\text{'}\rangle\]  
▷ Return \texttt{symbol} as literal.

\[\texttt{quote}\langle\texttt{string}\text{'}\rangle\]  
▷ Return \texttt{string} as literal.

\[\texttt{quote}\langle\texttt{boolean}\text{'}\rangle\]  
▷ Return \texttt{false}.

\[\texttt{quote}\langle\texttt{number}\rangle\]  
▷ Return \texttt{number} as literal.

\[\texttt{quote}\langle\texttt{symbol}\rangle\]  
▷ Return \texttt{symbol} as literal.

\[\texttt{quote}\langle\texttt{string}\rangle\]  
▷ Return \texttt{string} as literal.

\[\texttt{quote}\langle\texttt{boolean}\rangle\]  
▷ Return \texttt{false}.

\[\texttt{quote}\langle\texttt{number}\rangle\]  
▷ Return \texttt{number} as literal.

\[\texttt{quote}\langle\texttt{symbol}\text{'}\rangle\]  
▷ Return \texttt{symbol} as literal.

\[\texttt{quote}\langle\texttt{string}\text{'}\rangle\]  
▷ Return \texttt{string} as literal.

\[\texttt{quote}\langle\texttt{boolean}\text{'}\rangle\]  
▷ Return \texttt{false}.

\[\texttt{quote}\langle\texttt{number}\rangle\]  
▷ Return \texttt{number} as literal.

\[\texttt{quote}\langle\texttt{symbol}\rangle\]  
▷ Return \texttt{symbol} as literal.

\[\texttt{quote}\langle\texttt{string}\rangle\]  
▷ Return \texttt{string} as literal.

\[\texttt{quote}\langle\texttt{boolean}\rangle\]  
▷ Return \texttt{false}.

\[\texttt{quote}\langle\texttt{number}\rangle\]  
▷ Return \texttt{number} as literal.

\[\texttt{quote}\langle\texttt{symbol}\text{'}\rangle\]  
▷ Return \texttt{symbol} as literal.

\[\texttt{quote}\langle\texttt{string}\text{'}\rangle\]  
▷ Return \texttt{string} as literal.

\[\texttt{quote}\langle\texttt{boolean}\text{'}\rangle\]  
▷ Return \texttt{false}.

\[\texttt{quote}\langle\texttt{number}\rangle\]  
▷ Return \texttt{number} as literal.

\[\texttt{quote}\langle\texttt{symbol}\rangle\]  
▷ Return \texttt{symbol} as literal.

\[\texttt{quote}\langle\texttt{string}\rangle\]  
▷ Return \texttt{string} as literal.

\[\texttt{quote}\langle\texttt{boolean}\rangle\]  
▷ Return \texttt{false}.

\[\texttt{quote}\langle\texttt{number}\rangle\]  
▷ Return \texttt{number} as literal.

\[\texttt{quote}\langle\texttt{symbol}\text{'}\rangle\]  
▷ Return \texttt{symbol} as literal.

\[\texttt{quote}\langle\texttt{string}\text{'}\rangle\]  
▷ Return \texttt{string} as literal.

\[\texttt{quote}\langle\texttt{boolean}\text{'}\rangle\]  
▷ Return \texttt{false}.

\[\texttt{quote}\langle\texttt{number}\rangle\]  
▷ Return \texttt{number} as literal.

\[\texttt{quote}\langle\texttt{symbol}\rangle\]  
▷ Return \texttt{symbol} as literal.

\[\texttt{quote}\langle\texttt{string}\rangle\]  
▷ Return \texttt{string} as literal.

\[\texttt{quote}\langle\texttt{boolean}\rangle\]  
▷ Return \texttt{false}.

\[\texttt{quote}\langle\texttt{number}\rangle\]  
▷ Return \texttt{number} as literal.

\[\texttt{quote}\langle\texttt{symbol}\text{'}\rangle\]  
▷ Return \texttt{symbol} as literal.

\[\texttt{quote}\langle\texttt{string}\text{'}\rangle\]  
▷ Return \texttt{string} as literal.

\[\texttt{quote}\langle\texttt{boolean}\text{'}\rangle\]  
▷ Return \texttt{false}.

\[\texttt{quote}\langle\texttt{number}\rangle\]  
▷ Return \texttt{number} as literal.

\[\texttt{quote}\langle\texttt{symbol}\rangle\]  
▷ Return \texttt{symbol} as literal.

\[\texttt{quote}\langle\texttt{string}\rangle\]  
▷ Return \texttt{string} as literal.

\[\texttt{quote}\langle\texttt{boolean}\rangle\]  
▷ Return \texttt{false}.

\[\texttt{quote}\langle\texttt{number}\rangle\]  
▷ Return \texttt{number} as literal.

\[\texttt{quote}\langle\texttt{symbol}\text{'}\rangle\]  
▷ Return \texttt{symbol} as literal.

\[\texttt{quote}\langle\texttt{string}\text{'}\rangle\]  
▷ Return \texttt{string} as literal.

\[\texttt{quote}\langle\texttt{boolean}\text{'}\rangle\]  
▷ Return \texttt{false}.

\[\texttt{quote}\langle\texttt{number}\rangle\]  
▷ Return \texttt{number} as literal.

\[\texttt{quote}\langle\texttt{symbol}\rangle\]  
▷ Return \texttt{symbol} as literal.

\[\texttt{quote}\langle\texttt{string}\rangle\]  
▷ Return \texttt{string} as literal.

\[\texttt{quote}\langle\texttt{boolean}\rangle\]  
▷ Return \texttt{false}.

\[\texttt{quote}\langle\texttt{number}\rangle\]  
▷ Return \texttt{number} as literal.
16 External Environment

### (get-internal-real-time)
- Current time, or computing time, respectively, in clock ticks.
  - **internal-time-units-per-second**
    - Number of clock ticks per second.

### (encode-universal-time)
- **seconds** from 1900-01-01, 00:00, ignoring leap seconds.

### (decode-universal-time)
- **time-zone**
  - String representing physical location of computer.

### (isp-implementation)
- **machine**
  - Name or version of implementation, operating system, or hardware, respectively.

### (machine-instance)
- **Computer**
  - Name of computer.

### (print-time)
- **time**
  - String representing physical location of computer.

### (print-internal-runtime)
- **time**
  - String representing physical location of computer.