A, B, C, D represent any expression, and will be evaluated first
A B C ... indicates any number of arguments, even just one.

Functions that work only on numeric arguments

(zerop A)	T if A=0, NIL otherwise
(plusp A)	T if A>0, NIL otherwise
(minusp A)	T if A<0, NIL otherwise
(evenp A)	T if A is even, NIL otherwise
(oddp A)	T if A is odd, NIL otherwise
(= A B)	T if A=B, NIL otherwise
(= A B C D ...)	T if all of A B C D ... are equal, NIL otherwise
(/= A B)	T if A≠B, NIL otherwise
(/= A B C D ...)	T if all of A B C D ... are different, NIL otherwise
(< A B)	T if A<B, NIL otherwise
(< A B C D ...)	T if A<B and B<C and C<D ..., NIL otherwise
<=, >, >=	behave the same way as <
(max A B C ...)	the biggest of them all
(min A B C ...)	the smallest of them all
(+ A B C D ...)	A+B+C+D...
(- A B C D ...)	A-B-C-D...
(* A B C D ...)	A*B*C*D...
(/ A B C D ...)	A/B/C/D...
(rem A B)	(rem 9 4)=1, (rem -9 4)=-1, (rem 9 -4)=1, (rem -9 -4)=-1
(mod A B)	(mod 9 4)=1, (mod -9 4)=3, (mod 9 -4)=-3, (mod -9 -4)=-1
(gcd A B C ...)	greatest common divisor of them all
(lcm A B C ...)	least common multiple of them all
(random A)	random number, same type as A, ≥0 and <A
(expt A B)	A to the power of B
(sqrt A)	square root of A

other obvious operations, used just like sqrt:

abs exp log sin cos tan asin acos atan
floor ceiling truncate round numerator denominator

Functions that work only on character arguments

(alpha-char-p A)	T if A in 'A'..'Z' or 'a'..'z', NIL otherwise
(upper-case-p A)	T if A in 'A'..'Z', NIL otherwise
(lower-case-p A)	T if A in 'a'..'z', NIL otherwise
(digit-char-p A)	T if A in '0'..'9', NIL otherwise
(graphic-char-p A)	T if A is a normal non-invisible character, NIL otherwise
(alphanumericp A)	T if A in 'A'..'Z' or 'a'..'z' or '0'..'9', NIL otherwise
(char= A B ...)	used the same ways as (= A B) is for numbers also char/= char< char<= char> char>=
(char-upcase A)	if A is in 'a'..'z', convert it to capital. Otherwise = A
(char-downcase A)	if A is in 'A'..'Z', convert it to lower case. Otherwise = A
(char-code A)	converts from character to ASCII code (char-code #\a) = 97
(ocde-char A)	the opposite, (code-char 97) = #\a

Special things

`(setq SYM A)`	`SYM` is a symbol, it is not evaluated `A` is any expression, it is evaluated first the symbol's value is set to `A`
`(quote X)`	`X`, unevaluated. It can be any expression
`'X`	equivalent to `(quote X)`
`(set A B)`	`A` and `B` may be any expressions, they are both evaluated first The value of `A` must be something that can behave like a variable. `(setq A B)` is equivalent to `(set (quote A) B)`
`T`	constant used to represent the boolean value TRUE
`nil`	like NULL and null in C++ and Java, also used for FALSE

Cons-cell functions

`(cons A B)`	cons cell whose CAR is A and whose CDR is B
`(car A)`	`(car (cons A B)) = A`
`(cdr A)`	`(cdr (cons A B)) = A`
`(caar A)`	equivalent to `(car (car A))`
`(cadr A)`	equivalent to `(car (cdr A))`
`(cdaadr A)`	equivalent to `(cdr (car (car (cdr A))))` all combinations of up to four As and Ds are provided
`(nth A B)`	A must an int, B must be a list. Ath item of B, count from 0. so `(nth 0 '(a b c)) = a` and `(nth 2 '(a b c)) = c`
`(first A)`	equivalent to `(nth 0 A)`
`(second A)`	equivalent to `(nth 1 A)`, very tricky `second` is not `nth 2`.
`(tenth A)`	`(nth 9 A)`, all in between are provided too.

String functions

`(char A B)`	`A` must evaluate to a string, and `B` to an int returns `B`th character of `A`, counting from 0
`(string= A B ...)`	used the same ways as `(= A B)` is for numbers also `string/= string< string<= string> string>=`
`(string-upcase A)`	String same as A, but with every character in upper case
`(string-downcase A)`	String same as A, but with every character in lower case
`(coerce A 'cons)`	if `A` is a string, this produces a list of characters
`(coerce A 'string)`	if `A` is a list of characters, this produces a string

Input and Output

`(print A)`	What you'd expect, but with a newline printed first
`(prin1 A)`	What you'd expect `print` to do - no automatic newline
`(princ A)`	Same as `prin1` except characters and strings printed plain
`(princ-to-string A)`	returns a string, nothing is actually printed also `prin1-to-string` and `print-to-string`
`(terpri)`	print a newline
`(read)`	one whole Lisp value typed by user, not evaluated
`(read-line)`	string containing one whole line typed by user

Types

The important types are:

`number` `symbol` `cons` `string` `null` `character` `integer` `float` `rational`

`(coerce A B)`	Convert A to type B if reasonably possible e.g. `(coerce 7/5 'float) = 1.4`
`(typep A B)`	is A's type equal to B? T or nil e.g. `(typep 7/5 'rational) = T`
`(null A)`	is A = nil? T or nil
`(consp A)`	is A a cons cell? T or nil
`(symbolp A)`	is A a symbol? T or nil
`(functionp A)`	is A a function? T or nil
`(atom A)`	is A not a cons cell? T or nil - note that `'()` is an atom.
`(listp A)`	is A either nil or a cons cell? T or nil
`(numberp A)`	is A numeric? T or nil
`(stringp A)`	is A a string? T or nil
`(characterp A)`	is A a character? T or nil
`(integerp A)`	is A an integer ? T or nil
`(rationalp A)`	is A a rational? T or nil
`(floatp A)`	is A a float? T or nil
`(complexp A)`	is A a complex number? T or nil
`(type-of A)`	usually returns the type of A

Logic

`(eq A B)`	`T` if `A` and `B` are the same thing, `nil` otherwise Does not compare the contents of cons cells, but returns `T` if `A` and `B` are the same cons cell. for big numbers this will seem not to work
`(eql A B)`	`T` if `(eq A B)` would be `T`, also extended to work for characters and same-typed numbers
`(equal A B)`	Deep equality test, even for lists and trees.
`(not A)`	`T` if `A` is `nil`, `nil` otherwise
`(and A B C ...)`	evaluate A, B, C in turn until one is nil as soon as a nil is found, stop, the result is nil if no nil is found, the result is the value of the last argument
`(or A B C ...)`	evaluate A, B, C in turn until one is not nil as soon as a non-nil value is found, stop, that is the result if all values are nil, the result is nil