Core Common Lisp - uLisp subset

more at uLisp.com

SYMBOLS

nil	A symbol equivalent to the empty list ().
t	A symbol representing true.

LISTS

cons	**(cons *item item*)** If the second argument is a list, cons returns a new list with item added to the front of the list. `(cons 1 '(2 3))  (1 2 3)`
car, first	**(car *list*)** or **(first *list*)** Returns the first item in a list. `(first '(1 2 3))  1`
cdr, rest	**(cdr *list*)** or **(rest *list*)** Returns a list with the first item removed. `(rest '(0 1 2))  (1 2)`
list	**(list *item item* ...)** Returns a list of the values of its arguments. `(list 1 2 'foo)  (1 2 FOO)`
append	**(append *list list* ...)** Joins its arguments, which should be lists, into a single list.
length	**(length *item*)** Returns the number of items in a list, the length of a string, or the length of a one-dimensional array.
reverse	**(reverse *list*)** Returns a list with the elements of list in reverse order. `(reverse '(2 1))  (1 2)`
nth	**(nth *number list*)** Returns the nth item in a list, counting from zero. `(nth 1 '(a b c))  B`

DEFINING VARIABLES AND FUNCTIONS

lambda	**(lambda (p1 p2 ...) *form* ...)** Creates an unnamed function with parameters. Named functions are usually defined with `defun`. The usual use of lambda is to create a function for `mapcar`. `((lambda (x y) (+ x y)) 4 3)`
defun	**(defun name (p1 p2 ...) [*docstring*] *form* ...)** Allows you to define a function. Example: `(defun sq (x) (* x x))`
defvar	**(defvar *variable form*)** Defines a global variable.
setq	**(setq *sym1 val1* [*sym2 val2*]...)** Assigns the value to the symbol, optionally make several assignments. Example: `(setq a (+ 2 3)) a  5`
let	**(let ((*var value*) ...) *forms*)** Declares local variables then evaluates forms with those local variables. Also see let* Example: `(let ((a 2) (b 4)) (* a b))  8`

STRINGS

string=, string<, string>	**(string= *string string*)** Returns `t` if the first string is equal (or alphabetically less/greater than) the second string, and `nil` otherwise.
stringp	**(stringp *item*)** Returns `t` if the argument is a string and `nil` otherwise.
concatenate	**(concatenate 'string *string* ...)** Joins together the strings given in the second and subsequent arguments and returns a single string. Example: `(concatenate 'string "F" "OO")  "FOO"`

NUMERIC COMPARISONS

=, <, <=, >, >=	**(= *number number* ...)** Equal, less than (or equal) etc. Returns `t` if the comparison succeeds. Example: `(> 4 2 1)  t`
/=	**(/= *number number* ...)** Not equal. Returns `t` if none of the arguments are equal, and `nil` if two or more arguments are equal.
plusp, minusp	**(plusp number)** Returns `t` if the argument is greater than (with minusp, less than) zero, or nil otherwise. Note that `(plusp 0)` is `nil`.
zerop	**(zerop number)** Returns `t` if the argument is zero, or `nil` if otherwise.

MATH

+ - *	**(+ *number number* ...)** Adds, subtracts or multiplies its arguments together.
\	**(/ *number* ...)** Divides arguments. If there is one argument, inverts the argument. If there are two or more arguments, it divides the first argument by the second and subsequent arguments. Example: `(/ 60 2 3)  10`
mod	**(mod *number number*)** Returns its first argument modulo the second argument.
abs	**(abs *number*)** Returns the absolute, positive value of its argument.
random	**(random *number*)** Returns a random number between 0 and one less than its argument.
min, max	**(min *number* ...)** Returns the minimum or maximum of one or more arguments.
1+, 1-	**(1+ number)** Adds (or subtracts) one to its argument and returns it.

LOGIC AND CONDITIONALS

if	**(if *test then* [*else*])** Evaluates *test*. If it's non-nil the form then is evaluated and returned; otherwise the form *else* is evaluated and returned. The *else* form is optional. Example: `(if (= mynum 42) "woohoo" "meh"))`
cond	**(cond ((*test form*...) ...))** Cond provides a more flexible structure, each argument is a list consisting of a test optionally followed by one or more forms. If the test evaluates to non-nil the forms are evaluated and the last value is returned. If the test evaluates to nil, none of those forms are evaluated. Also see case. Example: `(cond ((< a 8) "L") ((< a 4) "H") (t "ERR"))`
when, unless	**(when *test form* ...)** Evaluates the test. If it's non-nil (nil for `unless`) the forms are evaluated and the last value is returned. Example: `(when (zerop (random 2)) "Zero!")`
and, or	**(and *item*...)** Evaluates its arguments until one returns nil (not-nil for `or`), and returns the last value.
not	**(not *item*)** Returns t if its argument is nil, or nil otherwise. Equivalent to null.

OUTPUT

princ	**(princ item [stream])** Prints its argument, and returns its value.
pprint	**(pprint item [stream])** Prints its argument, using the pretty printer to display it formatted in a structured way. It returns no value.
format	**(format output controlstring arguments ...)** Outputs its arguments formatted according to the format directives in the control string. `(format t "The answer is ~a" 42)`

TESTS

null	**(null *item*)** Returns t if its argument is nil, or nil otherwise. Equivalent to not.
atom	**(atom *item*)** Returns t if its argument is a single number, symbol, or nil.
listp, consp, symbolp, numberp	**(listp *item*)** Returns t if its argument is a list, cons, symbol, or number respectively.
eq	**(eq *item item*)** Tests whether the two arguments are the same OBJECT and returns t or `nil`. They are eq if they are the same symbol, character, number, or point to the same cons. Use `equal` to compare strings.
equal	**(equal *item item*)** Tests whether the two arguments look the same when printed, and returns t or nil as appropriate. Note that they might not necessarily be the same object. Objects are equal if they are `eq`, have the same string representation, or have the same list structure.

STRINGS AND LISTS

subseq	**(subseq *string start* [*end*])** Returns a subsequence of a list or string. `(subseq '(0 1 2 3 4) 2 4)  (2 3)`
search	**(search pattern target)** Returns the index of the first occurrence of pattern in target, which can be lists or strings. `nil` if it's not found. Example: `(search "cat" "a cat sat")  2`

INPLACE OPERATORS

setf	**(setf place value [place value]...)** Modifies an existing list by setting the position in the list specified by *place* to the result of evaluating value. *Place* can be *symbol*, `car`, `cdr` or `nth` or more. `Example:` `(setq x '(1 3)) (setf (nth 1 x) 2)  (1 2)`
push, pop	**(push item place)** Modifies the value of *place*, which should be a list, adds (or remove for `pop`) *item* onto the front of the list. Example: `(setq x '(2 3)) (push 1 x)  (1 2 3)`

ITERATION AND MAPPING

loop	**(loop *forms*...)** Loop executes its arguments repeatedly until one of them reaches a return form. Example: `(loop (princ "$") (if (= 0 (random 10)) (return)))  $$$`
return	**(return [*value*])** Exits from a `loop`, `dotimes`, or `dolist` block. Returns value, or `nil` if no value is specified.
dolist	**(dolist (*var list* [*result*]) *form*...)** Sets the local variable var to each element of list in turn, and executes the forms. It then returns result, or `nil` if result is omitted.
dotimes	**(dotimes (*var number* [*result*]) *form..)** Executes the forms number times, with the local variable var set to each integer from 0 to number-1 in turn. It then returns result, or nil if result is omitted. Example: `(dotimes (x 10) (princ x))  0123456789`
mapcar	**(mapcar *function list* ...)** Applies the function to each element in one or more lists, and returns the resulting list. Example: `(mapcar '1+ '(0 1 2))  (1 2 3)`
progn	**(progn form*)** Evaluates several forms grouped together into a block, and returns the result of evaluating the last form.
assoc	**(assoc key list)** Looks up a key in an association list of (key . value) pairs, and returns the matching pair, or nil if no pair is found.
member	**(member item list)** Searches for an item in a list, using eq, and returns the list starting from the first occurrence of the item, or nil if it is not found.
funcall	**(funcall function argument ...)** Calls the function with the specified arguments.
apply	**(apply function list)** Returns the result of evaluating the function specified by the first argument with the list of arguments specified by the second parameter. Example: `(apply '+ '(1 2 3))  6`
eval	**(eval form)** Evaluates its argument. Example: `(eval (list '* 2 24))  42`

ULISP SPECIAL

millis	**(millis)** Returns the time in milliseconds that uLisp has been running.
for-millis	**(for-millis ([number]) form ...)** Executes the forms and then waits until a total of number milliseconds have elapsed. It returns the total number of milliseconds taken.
delay	**(delay number)** Delays for a specified number of milliseconds.