Core Common Lisp Top 100 - uLisp Subset Cheat Sheet

by heow via cheatography.com/186454/cs/38972/

Symbols

nil	A symbol equivalent to the empty list ().
t	A symbol representing true.
nothing	A symbol with no value. It is useful if you want to suppress printing the result of evaluating a function. It is equivalent to (values) in Common Lisp.

List Functions

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.
car	**(car *list*)** or **(first *list*)** Returns the first item in a list.
cdr	**(cdr *list*)** or **(rest *list*)** Returns a list with the first item removed.
list	**(list *item item* ...)** Returns a list of the values of its arguments.
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.
nth	**(nth *number list*)** Returns the nth item in a list, counting from zero.
sort	**(sort *list test*)** Sorts the list according to the test function, and returns the sorted list. Note that for efficiency the sort is destructive, meaning that the value of list may be changed by the operation.

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**. Example: **((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) b (+ 4 5))**
let	**(let ((*var value*) …) *forms*)** Declares local variables then evaluates forms with those local variables. Example: **(let ((a 1) (b 2)) (* a a b))**
let*	Like let, but you can refer to local variables that have already been defined. Example: **(let* ((a 7) (b a)) (* a b))**

Strings

string<, string>	**(string< *string string*)** Returns t if the first string is alphabetically less than (or greater than) the second string, and nil otherwise.
string=	**(string= *string string*)** Tests whether two strings are the same.
stringp	**(stringp *item*)** Returns t if the argument is a string and nil otherwise.
concat-enate	**(concatenate 'string *string* ...)** Joins together the strings given in the second and subsequent arguments and returns a single string. Example: **(concatenate 'string "One" "Two" "Three")** ☐ **"OneTwoThree"**

Strings (cont)

princ-to-string	(princ-to-string item) Prints its argument to a string, and returns the string. Example: **(princ-to-string (* 2 3 4))** ☐ **"24"**

Math Functions

+ - *	**(+ *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, 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.

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. In the C Language, it is "!="
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 otherwise.
evenp, oddp	**(evenp number)** Returns t if the number is even (or odd) otherwise nil.

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 (= foo 42) "woohoo" "just a number"))**
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. Example: **(cond ((< a 64) "low") ((< a 192) "high") (t "error"))** Also see case
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 (digitalread 3) (digitalwrite 4 t))**
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.

Input / Output

read	(read [stream]) Reads an atom or list from the serial input and returns it. If stream is specified the item is read from the specified stream.
print	(print item [stream]) Prints its argument, and returns its value. Strings are printed with quotation marks and escape characters followed by a newline.
princ	(princ item [stream]) Prints its argument, and returns its value. Characters and strings are printed without quotation marks or escape characters
pprint	(pprint item [stream]) Prints its argument, using the pretty printer, to display it formatted in a structured way. It returns no value
readline, writeline	(read-line [stream]) Reads characters from the serial input up to a newline character, and returns them as a string, excluding the newline.
writestring	(write-string string [stream]) Writes a string. If stream is specified the string is written to the stream.
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) The answer is 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	**(listp *item*)** Returns t if its argument is a list.
consp	**(consp *item*)** Returns t if its argument is a non-null list.
numberp	**(numberp *item*)** Returns t if its argument is a number.
streamp	**(streamp *item*)** Returns t if the argument is a valid stream and nil otherwise.
eq	**(eq *item item*)** Tests whether the two arguments are the same **object** and returns t or nil. Objects are eq if they are the same symbol, same character, equal numbers, 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.
symbolp	**(symbolp *item*)** Returns t if the argument is a symbol and nil otherwise.

Characters

char	**(char *string n*)** Returns the nth character in a string, counting from zero.
char-code, code-char	**(char-code *character*)** Returns the ASCII code for a character, as an integer. **code-char** does the inverse.

Characters (cont)

characterp	**(characterp *item*)** Returns t if the argument is a character and nil otherwise.

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, or nil if it's not found. Example: **(search "cat" "the cat sat")) ☐ 4**

In-place Operations

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. (defvar xyz '(0 0 0)) xyz > (setf (nth 0 xyz) 90) 90 > (setf (nth 1 xyz) 45) 45 > xyz (90 45 0)
push, pop	(push item place) Modifies the value of place, which should be a list, to add item onto the front of the list, and returns the new list. The second argument place c
incf, decf	(incf place [number]) Adds one to the value of place, and returns the new value.

Iteration and Mapping

loop	**(loop *forms*...)** Loop executes its arguments repeatedly until one of them reaches a return form. **(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. **(dotimes (x 10) (princ x)) ☐ 0123456789**
mapc	**(mapc *function list1* [*list*]...)** Applies the function to each element in one or more lists, ignoring the results. It returns the first list argument.
mapcar	**(mapcar *function list* ...)** Applies the function to each element in one or more lists, and returns the resulting list. The function can be a built-in function; for example: **(mapcar * '(1 2 3) '(4 5 6) '(7 8 9)) ☐ (28 80 162)**
mapcan	(mapcan function list1 [list2]*) Applies the function to each element in one or more lists. The results should be lists, and these are appended together to give the value returned by mapcan.
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.

Iteration and Mapping (cont)	
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.
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.
eval	(eval form*) Evaluates its argument. ☐ Example:* **(eval (list '7 8)) 56**

uLisp Special Arduino Interface	
millis	(millis) Returns the time in milliseconds that uLisp has been running. On 8/16-bit versions of uLisp this is a signed 16-bit number, so after 32.767 seconds the number will wrap around from 32767 to -32768. In 32-bit versions of uLisp this is a signed 32-bit number which will wrap around after about 25 days.
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.