Down and dirty Lisp functions

List Func	Syntax	Example	Description
car	(**car** *list*)	(car '(a b c)) → a	Returns the first element of a list
cdr	(**cdr** *list*)	(cdr '(a b c)) → (b c)	Returns the sublist of *list* after the first element
c…r	(**c..r** *list*)	(cdadr *list*) → same as	(**cdr** (**car** (**cdr** *list*)))
nth	(**nth** number *list*)	(nth 2 '(a b c)) → c	Returns nth element of list (0 based)
cons	(**cons** *object list*)	(cons 'a '(b c)) → (a b c)	Places *object* into *list* object can be a list or atom
append	(**append** *lists..*)	(append '(a b) '(c d) '(e)) → (a b c d e)	
list	(**list** *objects..*)	(list 'a '(b c) 'd) → (a (b c) d)	Turns objects into a list
nconc	(**nconc** *lists*)	Also known as a destructive append	See handout of link lists *******
assc	(**assc** *object asslist*)	(assc 'y '((x 3) (y 4) (z 6))) → (y 4)	Look up association list
rplacd	(**rplacd** list object)	See handout on link lists for example**	Replaces the cdr of *list* with *object*
rplaca	(**rplaca** *list object*)	See handout on link lists for example**	Replaces the car of *list* with *object*
length	(**length** *list*)	(length '(a (b c) d)) → 3	Finds the number of items in a list
remove	(**remove** *object list*)	(remove 'a '(x a c d)) → (x c d)	Remove object from list
reverse	(**reverse** *list*)	(reverse '(a b c)) → (c b a)	Reverse the items in a list
member	(**member** *obj list*)	(member 'x '(a x c)) → (x c)	If in list **returns rest of list** for t, if not then **nil**
atom	(**atom** *object*)	(atom (car '(a b))) → t	Returns **t** if an atom, else **nil**
null	(**null** *object*)	(null '( )) → t	Returns **t** if nil or empty, else **nil**
numberp	(**numberp** *object*)	(numberp 3) → t (numberp 'a)→ nil	Return **t** if number, else **nil**
listp	(**listp** *object*)	(listp '(a b))→ t	Returns **t** if list, else **nil**
equal	(**equal** *obj1 obj2*)	(equal '(a) '(a))→ t	General form of equality, usually **t** if print the same way , else **f**
>, >=, =, <=, <	(< n1 n2)	(< 3 6) → t	Relations operation
zerop minusp plusp evenp oddp	(**minusp** number)	(plusp 7) → t	Number testing functions
+ - * / mod rem 1+ 1-	(+ 3 2)	(1+ 6)→ 7 (+ 1 2 3) → 6	Arithmetic functions
and	(and (< 1 2) (evenp 3)) → t	Evaluates each until nil, else evaluates last expression	
or	(or (null x) (numberp)) → t if x is nil or a number	Evaluates until t, else evaluates to nil	
not	(not (evenp 3)) → t	Inverts the t/f value	
Other Functions (see example page)			
cond def load setq setf set eval quote			

Simple examples of some Lisp functions

cond is much like a switch function in a language like C++

```
(cond
        (  (null list) 0 )        ; if (null list) is true then evaluate to  nil
        (  (atom car(list)) 1)   ; if the car of list is an atom evaluate to 1
        (   t  2)                    ; t is always true so works as default
)
```

defun allows the user to define a new function

```
(defun   add (a b)   ;   add  is the name,  (a b) is the formal parameter list,  always pass by value
   (  + a b)
)
```

* (add 2 4) → 6

load allows the user to load a text file

Assume you have placed the add function above into a file called myadd. You can then load it into the Lisp environment using load.

(load "myload")

setq allows you to bind a value to a symbol (does not evaluate the first argument)

(setq x 'a) x → a (setq L '(a b c)) L → (a b c) (setq M (car (a b c))) M → a
(setq num 4) num → 4

set is like setq but it evaluates both arguments.

(setq A 'X) A → X
(set A 'Z) X → Z
A → X

setf is the most general form of set which allows a lot of flexibility but could be used just like setq

eval forces an evaluation of a list

(setq x '(cons 'a '(b c))) x → (cons 'a '(b c))
(eval x) → (a b c)

quote allow you to quote something much like using the ' mark. It prevents the object from being evaluated.

(quote (+ 2 4)) → (+ 2 4)

Some example user functions

Reducing function add_list adds all the numeric elements in a list

```
(defun add_list (L)
  (cond ((null L) 0)
        ( t (+ (car L) (add_list (cdr L))))
  )
)
```

(add_list '(1 2 3)) → 6

Mapping function add1_list adds 1 to each numeric item in the list.

```
(defun add1_list (L)
  (cond ((null L) nil ) ; ground returns empty list
        ( t (cons (+ 1 (car L)) (add1_list (cdr L))))
  )
)
```

(add1_list '(1 2 3)) → (2 3 4)

Filtering function minus_pick returns a list with all the negative elements

```
(defun minus_pick (L)
  (cond ((null L) nil)
        ((minusp (car L)) (cons (car L) (minus_pick (cdr L))))
        ( t (minus_pick (cdr L)))
  )
)
```

(minus_pick '(2 -3 4 -6 -45 3)) → (-3 -6 -45)

The function **atomify** removes all list nestings and returns a list of atoms only.

```
(defun atomify (L)
  (cond ((null L) nil )
        ((atom (car L)) (cons (car L) (atomify (cdr L))))
        (  t (append (atomify (car L)) (atomify (cdr L))))
  )
)
```

(atomify '(a (b c) (e (f (g h) i)) j) → (a b c e f g h i j)