

## LISP FUNCTIONS

Summarized below are several basic functions used in the Lisp programming language. In some cases, both a short form and a long form are given (separated by **or**).

### Basic List Operations

(car X) - return the head (first element) of the list X

(cdr X) - return the tail of the list X (X with the first element removed)

(cons X Y) - return the list created by inserting the list X onto the front of list Y

(append X Y) - return the list created by concatenating lists X and Y

(list X<sub>1</sub> X<sub>2</sub> ... X<sub>n</sub>) - return the list created by concatenating lists X<sub>1</sub>, X<sub>2</sub>, ... X<sub>n</sub>

(' X) **or** (quote X) - return the list X (treated as a literal)

### Arithmetic Operations

(+ E<sub>1</sub> E<sub>2</sub> ... E<sub>n</sub>) - return  $E_1 + E_2 + \dots + E_n$

(- E<sub>1</sub> E<sub>2</sub> ... E<sub>n</sub>) - return  $E_1 - E_2 - \dots - E_n$

(\* E<sub>1</sub> E<sub>2</sub> ... E<sub>n</sub>) - return  $E_1 * E_2 * \dots * E_n$

(/ E<sub>1</sub> E<sub>2</sub> ... E<sub>n</sub>) - return  $E_1 / E_2 / \dots / E_n$

(1+ X) - return  $X + 1$

(1- X) - return  $X - 1$

### Relational Operations

(zerop X) - if  $X = 0$  then return T else return NIL

(**relation** X Y) - if X **relation** Y then return T else return NIL (**relation** is =, /=, <, <=, >, >=)

(equal X Y) - if  $X = Y$  then return T else return NIL

(eq X Y) - if X and Y are the same object, then return T else return NIL

### Logical Operations

(and X<sub>1</sub> X<sub>2</sub> ... X<sub>n</sub>) - return  $X_1 \wedge X_2 \wedge \dots \wedge X_n$

(or X<sub>1</sub> X<sub>2</sub> ... X<sub>n</sub>) - return  $X_1 \vee X_2 \vee \dots \vee X_n$

(not X) - return  $\neg X$

### **Control Operations**

(setq X Y) - assign the value of Y to X and return the value of Y

(set X Y) - same as setq except that X can be a computed value

(cond (test <sub>1</sub> expression <sub>1</sub> )	if test <sub>1</sub> then return expression <sub>1</sub>
(test <sub>2</sub> expression <sub>2</sub> )	else if test <sub>2</sub> then return expression <sub>2</sub>
...	...
(test <sub>n</sub> expression <sub>n</sub> ))	else if test <sub>n</sub> then return expression <sub>n</sub>

(print X) - print X

### **Function Manipulation Operations**

(defun F (X<sub>1</sub> X<sub>2</sub> ··· X<sub>n</sub>) E) - define function F on arguments X<sub>1</sub>, ..., X<sub>n</sub> to be the value of expression E

(apply F L) - apply the function denoted by F to the argument list L

(eval E) - evaluate the expression denoted by E

(prog (X<sub>1</sub> X<sub>2</sub> ··· X<sub>n</sub>) E<sub>1</sub> E<sub>2</sub> ··· E<sub>m</sub>) - define local variables X<sub>1</sub>, ..., X<sub>n</sub> to be used in evaluating expressions E<sub>1</sub>, ..., E<sub>m</sub>, the value of the last being returned as the value of the PROG expression

(return E) - return the value of E (usually used in a prog expression)