

About those quotes...

Question: Why do we say the value of `(REST ' (C A B))` is `(A B)` ?
Why don't we say it is `' (A B)` ?

Answer: `(A B)` is the value and `' (A B)` is an expression that has the value, just as `5` is a value and `(+ 2 3)` is an expression that has the value.

Another ex.: When we evaluate `(FIRST (REST ' (A B)))`, here's what happens:

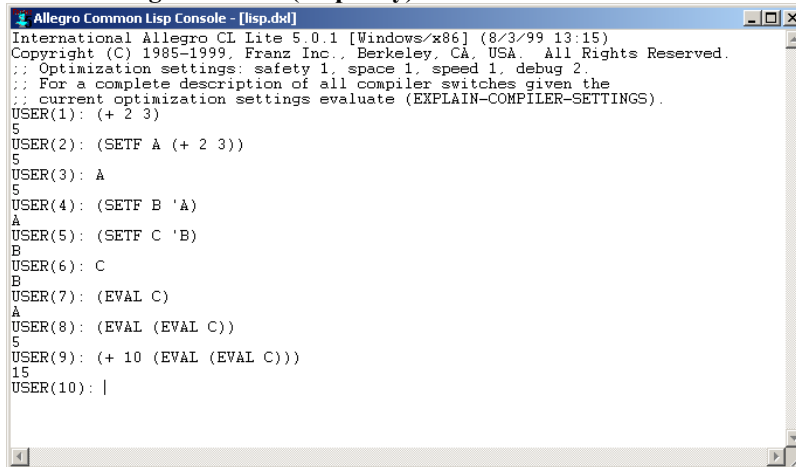
- (1) `FIRST` is a function, so it evaluates its argument `(REST ' (A B))`.
- (2) `REST` is a function, so it evaluates its argument `' (A B)`.
- (3) The value of `' (A B)` is `(A B)`.
- (4) `REST` operates on `(A B)` to give `(B)`.
- (5) `FIRST` operates on `(B)` to give `B`.

The quote is used to block evaluation at a specific place (so that we won't try to call a function named `A`).
It is not part of the value itself.

A sample session with Allegro Common Lisp

(free software for Windows, from the AI Center lend-out disc; also available on the PCs in the AI Lab, Room 111, Grad. Studies; *you need an account* in order to use these PCs).

This is "Allegro CL Lite (Lisp only)."



```

Allegro Common Lisp Console - [lisp.dcl]
International Allegro CL Lite 5.0.1 [Windows/x86] (8/3/99 13:15)
Copyright (C) 1985-1999, Franz Inc., Berkeley, CA, USA. All Rights Reserved.
;; Optimization settings: safety 1, space 1, speed 1, debug 2.
;; For a complete description of all compiler switches given the
;; current optimization settings evaluate (EXPLAIN-COMPILER-SETTINGS).
USER(1): (+ 2 3)
5
USER(2): (SETF A (+ 2 3))
5
USER(3): A
5
USER(4): (SETF B 'A)
A
USER(5): (SETF C 'B)
B
USER(6): C
B
USER(7): (EVAL C)
A
USER(8): (EVAL (EVAL C))
5
USER(9): (+ 10 (EVAL (EVAL C)))
15
USER(10): |
  
```

When you get an error, the prompt will change to

[1] USER(2) :

or the like, where `[1]` means you are in the debugger. To get back to the original prompt, type:

:reset

Defining your own functions

```
(defun name (sym1 sym2...) expression expression expression expression...)
```

Evaluating a `(defun ...)` expression causes the symbol *name* to become the name of a function which is computed as follows:

- (1) Symbols *sym1 sym2* etc. are used as local variables for the arguments of the function.
- (2) All of the expressions are evaluated, in order. (There is often only one.)
- (3) The value of the function is the value of the last expression evaluated.

The `(defun ...)` expression itself has a value, which is the name of the function.

Examples:

```
> (defun double (x) (* 2 x))
DOUBLE
> (double 3)
6
```

```
> (defun second-element (x) (first (rest x)))
SECOND-ELEMENT
> (second-element '(alpha beta gamma))
BETA
```

Storing function definitions in files

It is helpful to store function definitions in a text file. The function

```
(load "filename")
```

 (with backslashes written TWICE)

reads the file and evaluates all the expressions in it, but does not print their values.

Example:

```
(load "c:\\temp\\test.lisp")
```

reads file `c:\temp\test.lisp`.