Note:

To make sense of this, it is important to understand what is evaluated by Lisp vs what is evaluated by our interpreter.

The general format should be:

(starteval 'e)

where e is the expression to be evaluated by our interpreter.

The quote is needed here, otherwise Lisp will already evaluate the argument e before handing it to starteval.

Another important restriction is that our interpreter has no notion of a built-in list data type. All lists must be constructed using cons. If you give it a list such as (1 2), it will treat it as a function application of function 1 to argument 2, and fail.

To understand more about how the interpreter works, try:

(trace xeval)

Examples:

(starteval 5)

(starteval (+ 5 2)) # it works, but lisp already reduced the (+5 2) to 7 before handing it to our interpreter

(starteval '(+ 5 2))

7 # the right way to call it. Our interpreter handles the + function application. One way to see that is to redefine what our + function computes.

(starteval t)

(starteval nil)

(starteval '(xquote (1 2)))

(starteval '(atom x))

(starteval '(atom (1 2))) # ERROR

(starteval '(atom (cons 1 (cons 2 nil)))) # CORRECT

(starteval '(car (cons 1 nil)))

(starteval '(car (cons 1 (cons 2 nil))))

(starteval '(cdr (cons 1 (cons 2 nil)))) # note it returns a Lisp list, since calling the Lisp cdr in our interpreter

(starteval '((lambda (x) (+ x 1)) 5))

(starteval '(((lambda (x y) (lambda (x) (+ x y))) 2 3) 4))

(starteval '((lambda (x) (x 2)) (lambda (x) (+ x 1))))

(starteval '((lambda (x y) (+ x y)) ((lambda (y z) (+ y z)) 2 3) 4))