3

EVAL Notation

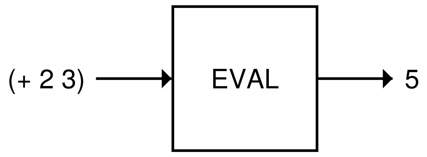
3.1 INTRODUCTION

Before progressing further in our study of Lisp, we must switch to a more flexible notation, called **EVAL notation**. Instead of using boxes to represent functions, we will use lists. Box notation is easy to read, but EVAL notation has several advantages:

- Programming concepts that are too sophisticated to express in box notation can be expressed in EVAL notation.
- EVAL notation is easy to type on a computer keyboard; box notation is not.
- From a mathematical standpoint, representing functions as ordinary lists is an elegant thing to do, because then we can use exactly the same notation for functions as for data.
- In Lisp, functions *are* data, and EVAL notation allows us to write functions that accept other functions as inputs. We'll explore this possibility further in chapter 7.
- When you have mastered EVAL notation, you will know most of what you need to begin conversing in Lisp with a computer.

3.2 THE EVAL FUNCTION

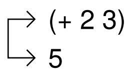
The EVAL function is the heart of Lisp. EVAL's job is to evaluate Lisp **expressions** to compute their result. Most expressions consist of a function followed by a set of inputs. If we give EVAL the expression (+ 2 3), for example, it will invoke the built-in function + on the inputs 2 and 3, and + will return 5. We therefore say the expression (+ 2 3) **evaluates to** 5.



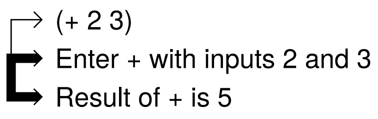
From now on, instead of drawing an EVAL box we'll just use an arrow. The preceding example will be written like this:

```
(+ 2 3)  ⇒  5
```

When we want to be slightly more verbose, we'll use a two-headed arrow:



And when we want to show as much detail as possible, we will use a three-headed arrow, like this:

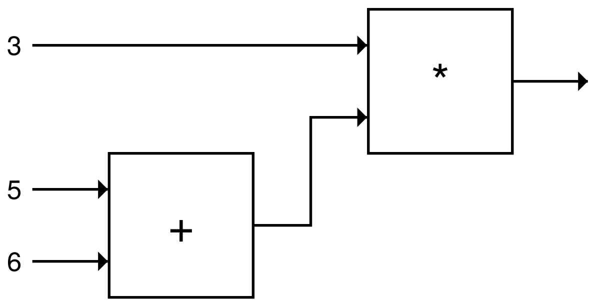


The meanings of the thin and thick lines will be explained later. Here are some more examples of expressions in EVAL notation:

```
(+ 1 6)  ⇒  7

(oddp (+ 1 6))  ⇒  t

(* 3 (+ 1 6))  ⇒  21

(/ (* 2 11) (+ 1 6))  ⇒  22/7
```

3.3 EVAL NOTATION CAN DO ANYTHING BOX NOTATION CAN DO

It should be obvious that any expression we write in box notation can also be written in EVAL notation. The expression



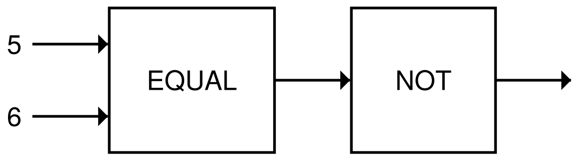
can be represented in EVAL notation as

```
(* 3 (+ 5 6))
```

Similarly, the EVAL notation expression

```
(not (equal 5 6))
```

is represented in box notation as



You may notice that EVAL notation appears to read opposite to box notation, in other words, if you read the box notation expression above as ‘‘five six, EQUAL, NOT,’’ the corresponding EVAL notation expression reads ‘‘NOT EQUAL five six.’’ In the box notation version the computation starts on the left and flows rightward. In EVAL notation the inputs to a function are processed left to right, but since expressions are nested, evaluation actually starts at the innermost expression and flows outward, making the order of function calls in this example right to left.

3.4 EVALUATION RULES DEFINE THE BEHAVIOR OF EVAL

EVAL works by following a set of evaluation rules. One rule is that numbers and certain other objects are ‘‘self-evaluating,’’ meaning they evaluate to themselves. The special symbols T and NIL also evaluate to themselves.

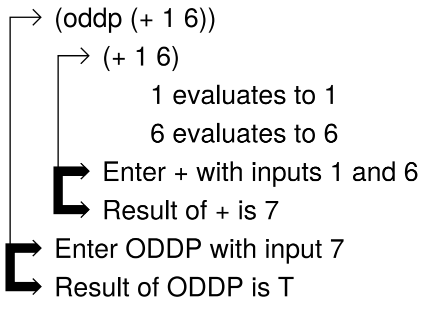
```
23  ⇒  23

t  ⇒  t

nil  ⇒  nil
```

Evaluation Rule for Numbers, T, and NIL: *Numbers, and the symbols T and NIL, evaluate to themselves.*

There is also a rule for evaluating lists. The first element of a list specifies a function to call; the remaining elements are the unevaluated **arguments** to the function. These arguments must be evaluated, in left to right order, to determine the inputs to the function. For example, to evaluate the expression (ODDP (+ 1 6)) the first thing we must do is evaluate ODDP’s argument: the list (+ 1 6). To do that, we start by evaluating the arguments to +. 1 evaluates to 1, and 6 evaluates to 6. Now we can call the + function with those inputs and get back the result 7. The 7 then serves as the input to ODDP, which returns T.

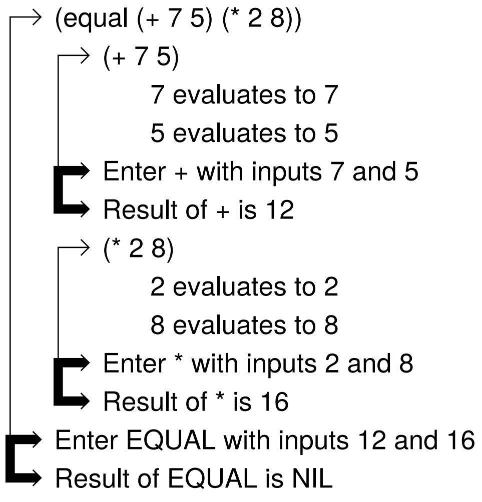
Evaluation Rule for Lists: *The first element of the list specifies a function to be called. The remaining elements specify arguments to the function. The function is called on the evaluated arguments.*

The following diagram, called an **evaltrace diagram**, shows how the evaluation of (ODDP (+ 1 6)) takes place. Notice that evaluation proceeds from the inner nested expression, (+ 1 6), to the outer expression, ODDP. This inner-to-outer quality is reflected in the shape of the evaltrace diagram.



```
→ (oddp (+ 1 6))
    → (+ 1 6)
        1 evaluates to 1
        6 evaluates to 6
    Enter + with inputs 1 and 6
    Result of + is 7
Enter ODDP with input 7
Result of ODDP is T
```

Here's another example of the arguments to a function getting evaluated before the function is called: an evaltrace for the expression (EQUAL (+ 7 5) (* 2 8)):



```
→ (equal (+ 7 5) (* 2 8))
    → (+ 7 5)
        7 evaluates to 7
        5 evaluates to 5
    Enter + with inputs 7 and 5
    Result of + is 12
    → (* 2 8)
        2 evaluates to 2
        8 evaluates to 8
    Enter * with inputs 2 and 8
    Result of * is 16
Enter EQUAL with inputs 12 and 16
Result of EQUAL is NIL
```

EXERCISES

3.1. What does (NOT (EQUAL 3 (ABS -3))) evaluate to?

3.2. Write an expression in EVAL notation to add 8 to 12 and divide the result by 2.

3.3. You can square a number by multiplying it by itself. Write an expression in EVAL notation to add the square of 3 and the square of 4.

3.4. Draw an evaltrace diagram for each of the following expressions.

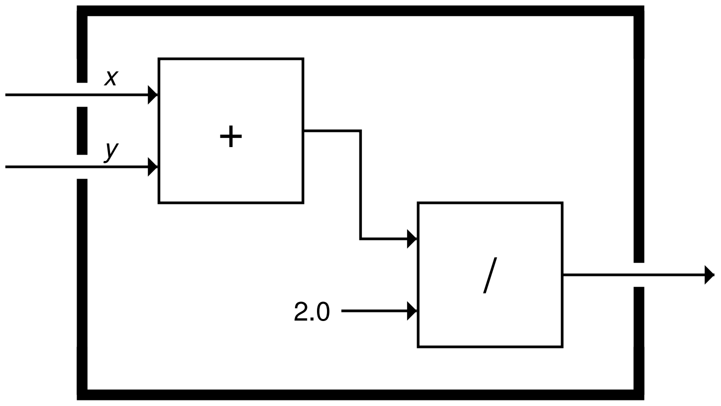
```
(- 8 2)
```

```
(not (oddp 4))

(> (* 2 5) 9)

(not (equal 5 (+ 1 4)))
```

3.5 DEFINING FUNCTIONS IN EVAL NOTATION

In box notation we defined a function by showing what went on inside the box. The inputs to the function were depicted as arrows, In EVAL notation we use lists to define functions, and we refer to the function's arguments by giving them names. We can name the inputs to box notation functions too, by writing the name next to the arrow like this:

Definition of AVERAGE:



The AVERAGE function is defined in EVAL notation this way:

```
(defun average (x y)
  (/ (+ x y) 2.0))
```

DEFUN is a special kind of function, called a **macro function**, that does not evaluate its arguments. Therefore they do not have to be quoted. DEFUN is used to define other functions. The first input to DEFUN is the name of the function being defined. The second input is the **argument list**: It specifies the names the function will use to refer to its arguments. The remaining inputs to DEFUN define the **body** of the function: what goes on ''inside the box.'' By the way, DEFUN stands for *de*fine *fun*ction.

Once you've typed the function definition for AVERAGE into the computer, you can call AVERAGE using EVAL notation. When you type (AVERAGE 6 8), for example, AVERAGE uses 6 as the value for X and 8 as the value for Y. The result, naturally, is 7.0.

Here is another example of function definition with DEFUN:

```
(defun square (n) (* n n))
```

The function's name is SQUARE. Its argument list is (N), meaning it accepts one argument which it refers to as N. The body of the function is the expression (* N N). The right way to read this definition aloud (or in your head) is: ‘‘DEFUN SQUARE of N, times N N.’’

Almost any symbol except T or NIL can serve as the name of an argument. X, Y, and N are commonly used, but BOZO or ARTICHOKE would also work. Functions are more readable when their argument names mean something. A function that computed the total cost of a merchandise order might name its arguments QUANTITY, PRICE, and HANDLING-CHARGE.

```
(defun total-cost (quantity price handling-charge)
  (+ (* quantity price) handling-charge))
```

EXERCISES

3.5. Write definitions for HALF, CUBE, and ONEMOREP using DEFUN. (The CUBE function should take a number *n* as input and return n^3.)

3.6. Define a function PYTHAG that takes two inputs, *x* and *y*, and returns the square root of x^2+y^2. You may recognize this as Pythagoras's formula for computing the length of the hypotenuse of a right triangle given the lengths of the other two sides. (PYTHAG 3 4) should return 5.0.

3.7. Define a function MILES-PER-GALLON that takes three inputs, called INITIAL-ODOMETER-READING, FINAL-ODOMETER-READING, and GALLONS-CONSUMED, and computes the number of miles traveled per gallon of gas.

3.8. How would you define SQUARE in box notation?