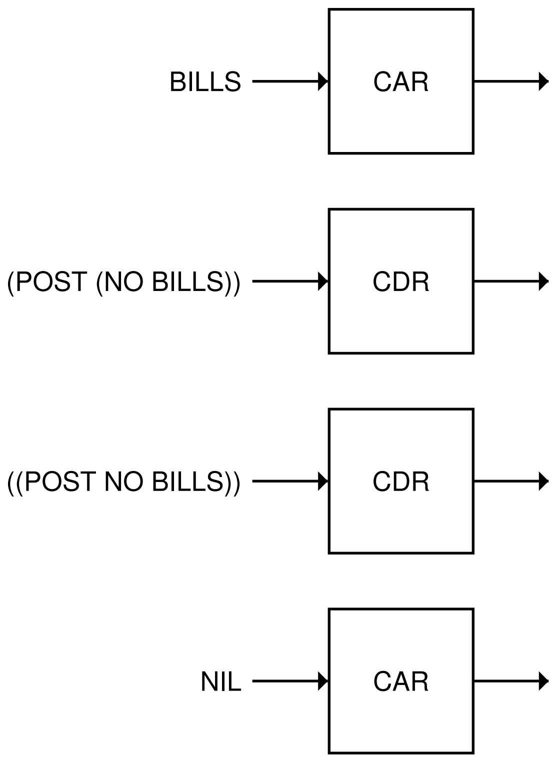
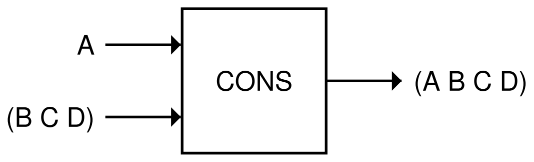


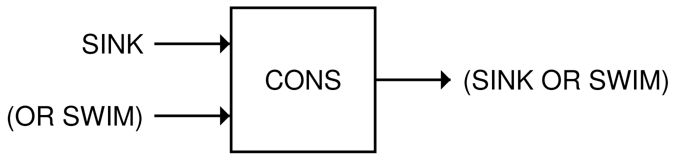
2.11 CONS

The CONS function creates cons cells. It takes two inputs and returns a pointer to a new cons cell whose CAR points to the first input and whose CDR points to the second. The term ‘‘CONS’’ is short for CONStruct.

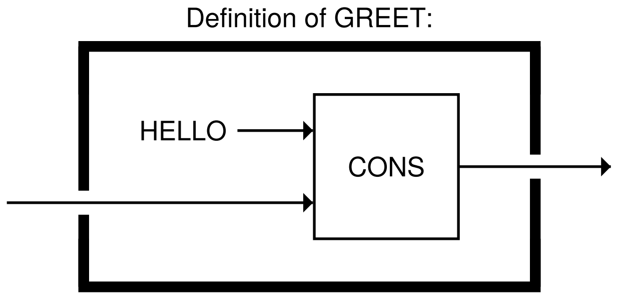
If we try to explain CONS using parenthesis notation, we might say that CONS adds an element to the front of a list. For example, we can add the symbol A to the front of the list (B C D):



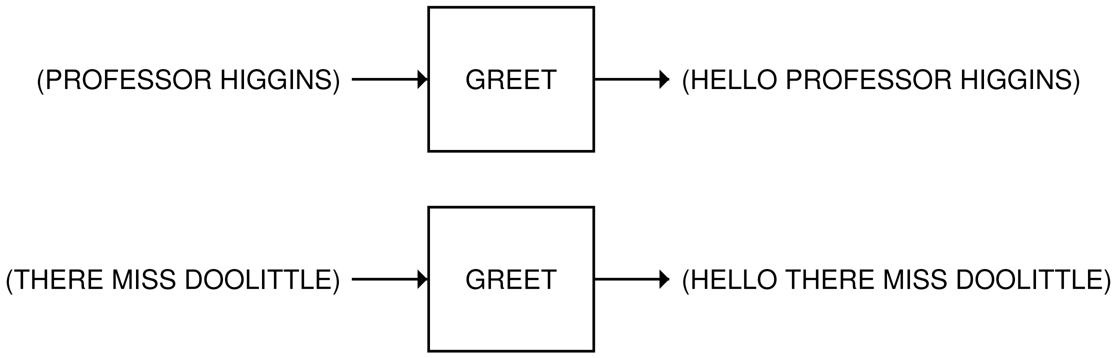
Another example: adding the symbol SINK onto the list (OR SWIM).



Here is a function GREET that adds the symbol HELLO onto whatever list it is given as input:

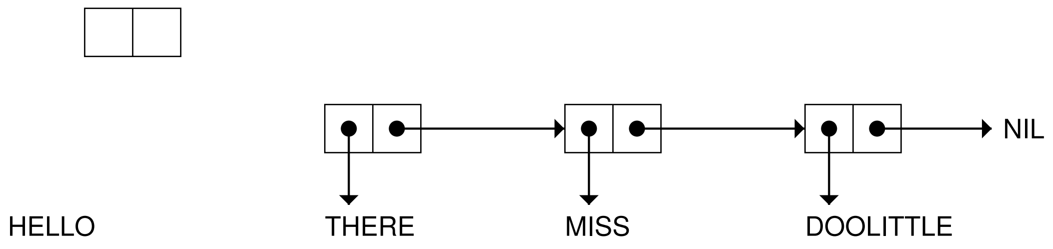


Examples of GREET:

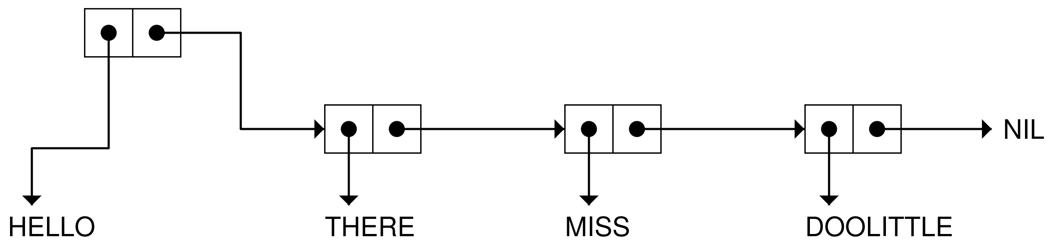


To really understand what CONS does, it is better to think about it using cons cell notation. CONS is a very simple function: It doesn't know anything about the ''front of a list.'' (Remember, inside the computer there are no parentheses.) All CONS does is create one new cons cell. But if the second input to CONS is a cons cell chain of length *n*, the new cell will form the head of a cons cell chain of length *n*+1. See Figure 2-1. So even though CONS just returns a pointer to the cell it created, in effect it returns a cons cell chain one longer than its second input.

CONS creates a new cons cell:



It fills in the CAR and CDR pointers:



And it returns a pointer to the new cell, which is now the head of a cons cell chain one longer than CONS's second input:

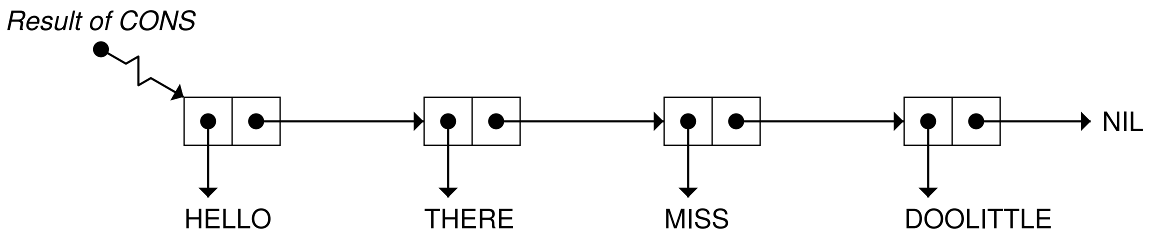
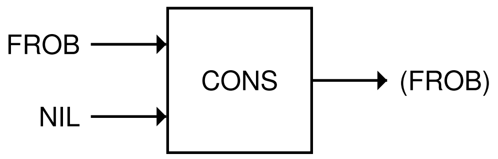


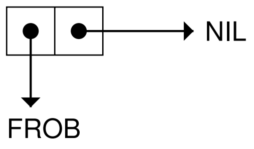
Figure 2-1 Creating a new cons cell with CONS.

2.11.1 CONS and the Empty List

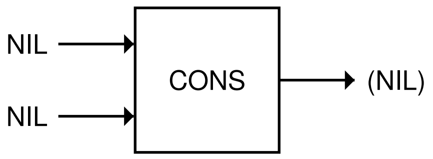
Since NIL is the empty list, if we use CONS to add something onto NIL we get a list of one element.



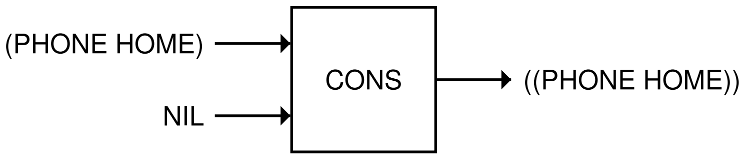
You should be able to confirm this result by looking at the cons cell notation for the list (FROB). The CAR of (FROB) is the symbol FROB and the CDR of (FROB) is NIL, so CONS must have built the list from the inputs FROB and NIL.



Here's another example that's very similar, except that NIL has been substituted for FROB:

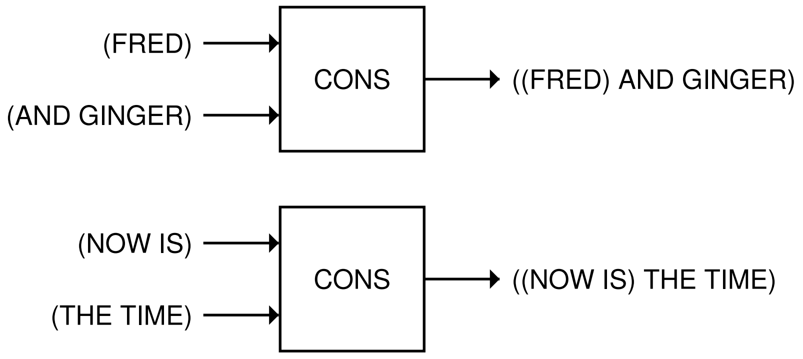


In printed notation, consing something onto NIL is equivalent to throwing an extra pair of parentheses around it.



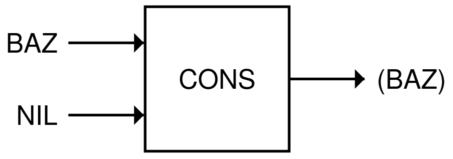
2.11.2 Building Nested Lists With CONS

Any time the first input to CONS is a nonempty list, the result will be a nested list, that is, a list with more than one level of cons cells. Examples:

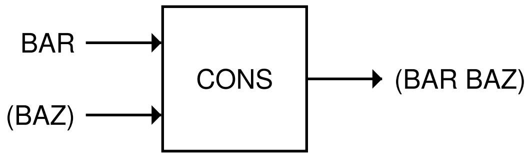


2.11.3 CONS Can Build Lists From Scratch

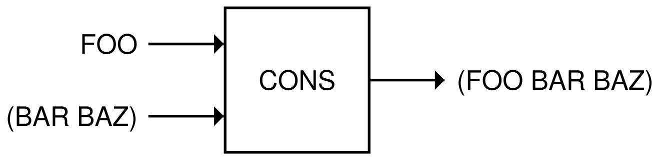
Suppose we wish to construct the list (FOO BAR BAZ) from scratch. We could start by adding the symbol BAZ onto the empty list. This gives us the list (BAZ).



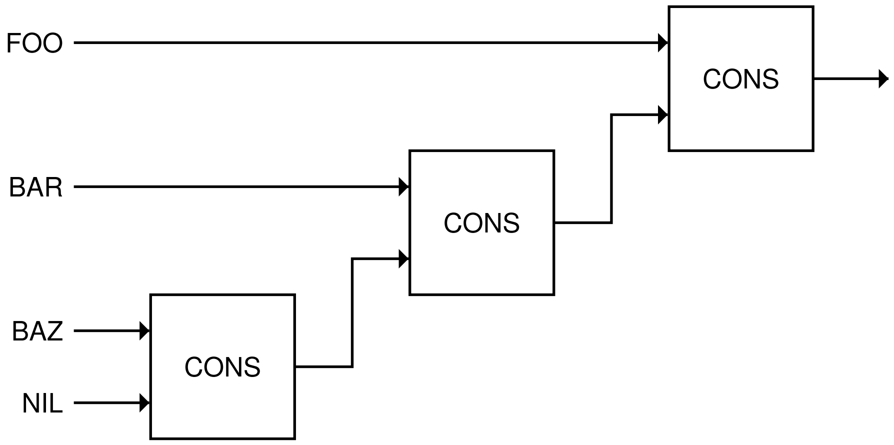
Then we can add BAR onto that:



Finally we add the FOO:



We have cascaded three CONSs together to build the list (FOO BAR BAZ) from scratch. Here is a diagram of the cascade:



If you turn this diagram sideways you will see that it is almost identical to the cons cell diagram for the list (FOO BAR BAZ). This should give you a clue as to why cons cells and the CONS function share the same name.

EXERCISE

2.18. Write a function that takes any two inputs and makes a list of them using CONS.

2.12 SYMMETRY OF CONS AND CAR/CDR

There is an interesting symmetry between CONS and CAR/CDR. Given some list *x*, if we know the CAR of *x* and the CDR of *x* we can CONS them together to figure out what *x* is. For example, if the CAR of *x* is the symbol A and the CDR of *x* is the list (E I O U), we know that *x* must be the list (A E I O U).

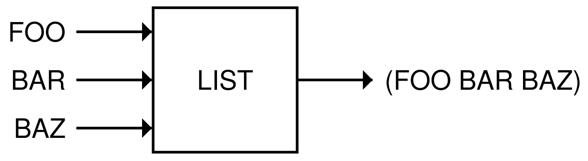
The symmetry between CONS and CAR/CDR can be expressed formally as:

x = CONS of (CAR of *x*) and (CDR of *x*)

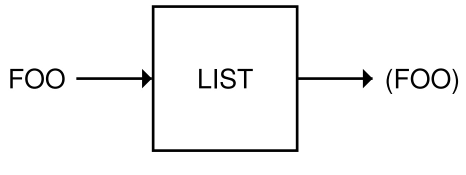
However, this relationship only holds for nonempty lists. When *x* is NIL, the CAR and CDR of *x* are also NIL. If we try to reconstruct *x* by consing together its CAR and CDR portions—that is, CONS of NIL and NIL—we get the list (NIL), not the empty list NIL. This should not be taken to mean that NIL and (NIL) are identical, for we know that they are not. Instead it serves to remind us that although NIL is a list, it's a very unusual one. Certain facts about lists apply only to nonempty ones, in other words, those containing at least one cons cell.

2.13 LIST

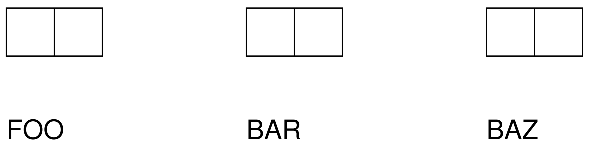
Creating a list from a bunch of elements is such a common operation that Lisp has a built-in function to do just that. The LIST function takes any number of inputs and makes a list of them. That is, it makes a new cons cell chain, ending in NIL, with as many cells as there are inputs. Figure 2-2 demonstrates this process.



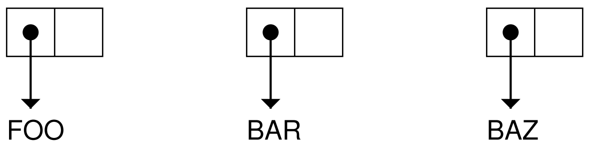
Recall that CONS always makes a single new cons cell; it appears to add its first input onto the list that is its second input. The LIST function, on the other hand, makes an entirely new cons cell chain. In parenthesis notation, it appears to throw a pair of parentheses around its inputs, however many there are. The result of LIST always has one more level of parenthesization than any input had.



LIST allocates three new cons cells:



It fills in the CAR pointers:



Then it fills in the CDR pointers to form a chain, and returns a pointer to the first cell:

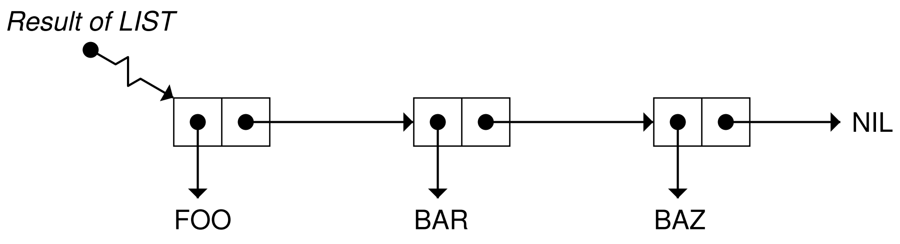
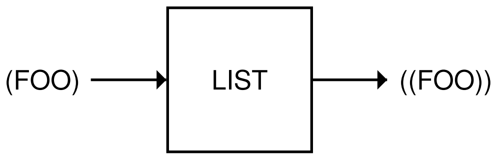
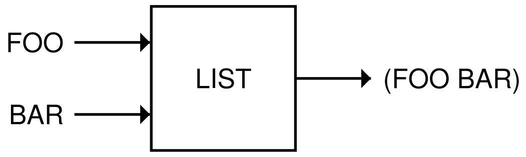
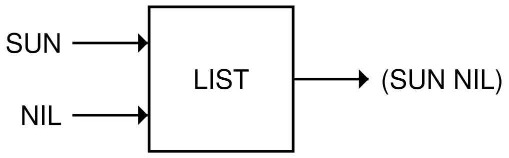


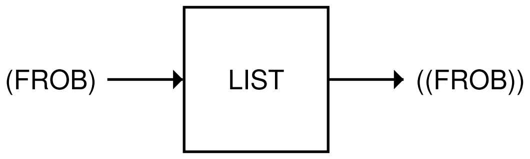
Figure 2-2 How LIST builds a new list.

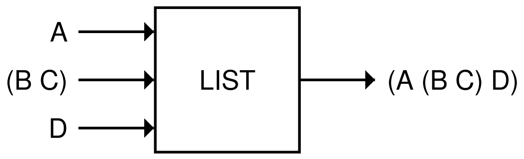


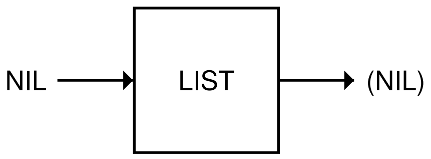
LIST actually works by building a new chain of cons cells. The CAR halves of the cells point to the inputs LIST received. The result of LIST is a pointer to the first cell in the chain. Examples of LIST:





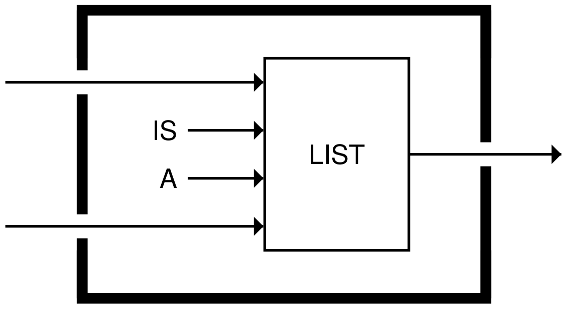




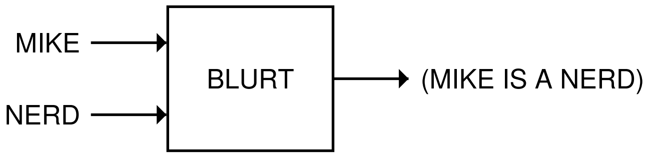


Here is a function called BLURT that takes two inputs and uses them to fill in the blanks in a sentence constructed with LIST.

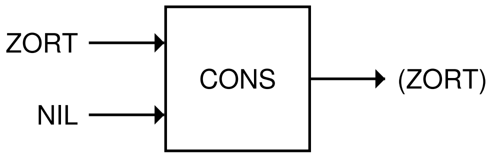
Definition of BLURT:

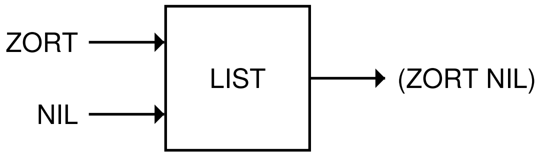


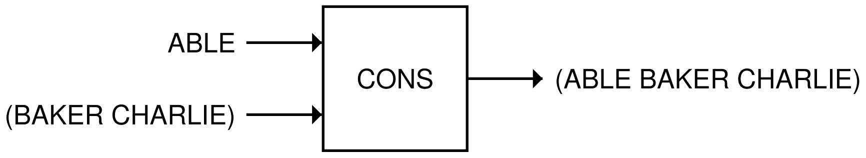
Example of BLURT:

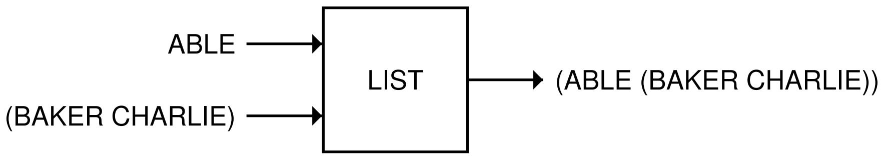


Let's look again at the difference between CONS and LIST. CONS makes a single cons cell. LIST makes a new cons cell chain list out of however many inputs it receives.

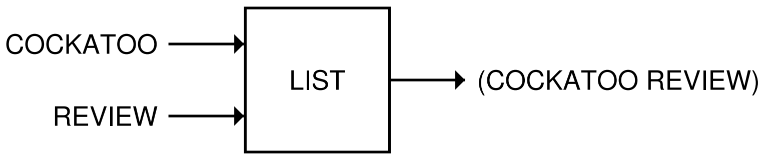




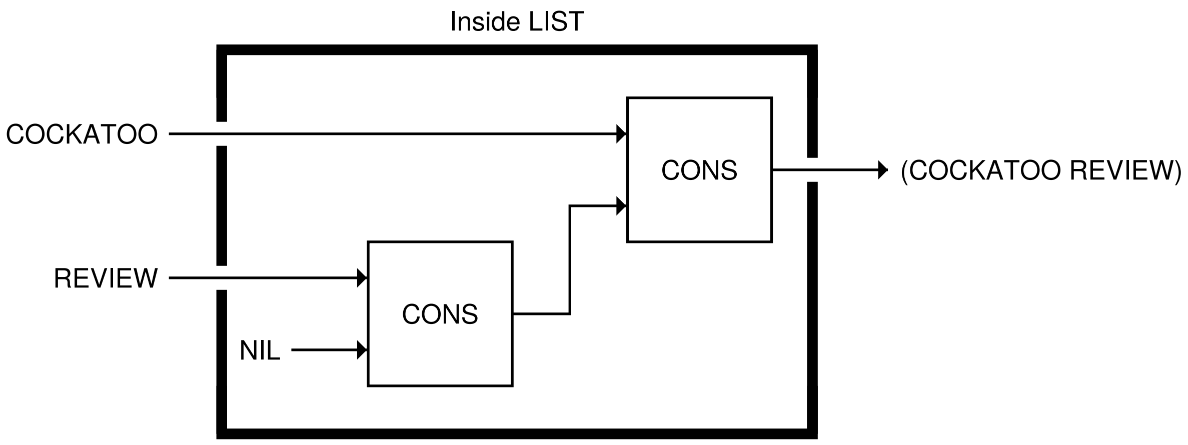




Another way to understand LIST is to think of it as expanding into a cascade of CONS boxes, much the way a call to an arithmetic function like ‘‘+ of 2, 3, 7, and 12’’ expands into a cascade of calls to the two-input version of +. So, what really goes on inside the LIST primitive, given an expression like

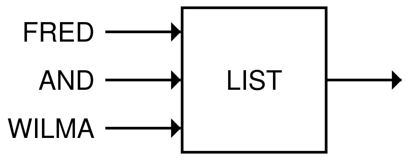


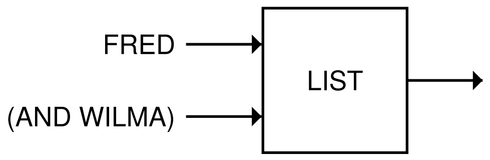
is that several cascaded calls to CONS are made:

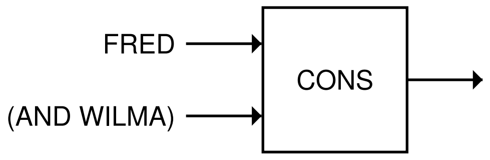


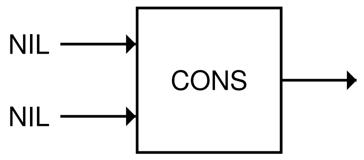
EXERCISE

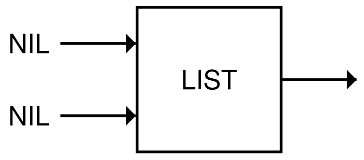
2.19. Fill in the results of the following computations.







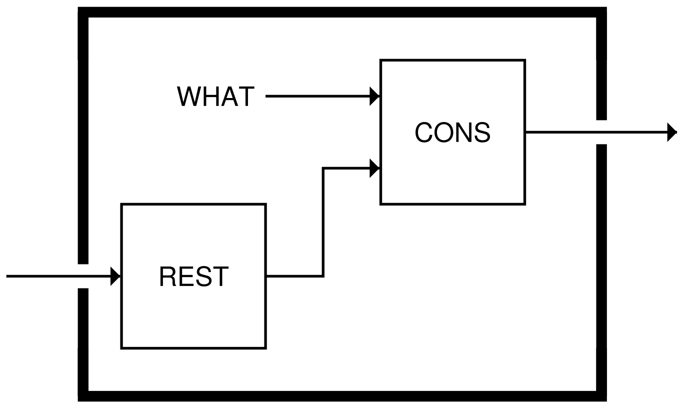




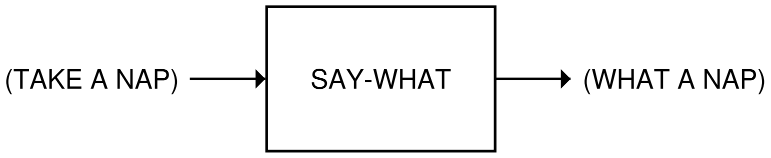
2.14 REPLACING THE FIRST ELEMENT OF A LIST

Suppose we want to replace the first element of a list with the symbol WHAT. The REST function can be used to obtain the sublist beyond the first element; then we can use CONS to add the symbol WHAT to the front of that sublist. We'll call our function SAY-WHAT.

Definition of SAY-WHAT:

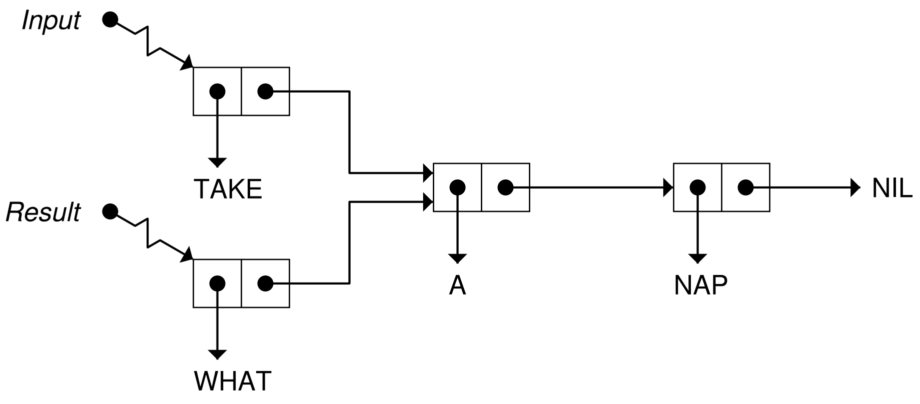


Here's an example of SAY-WHAT:



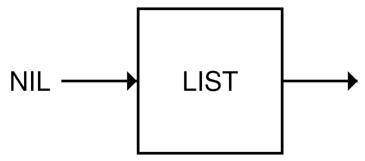
The REST of (TAKE A NAP) is (A NAP). Consing the symbol WHAT onto that yields (WHAT A NAP).

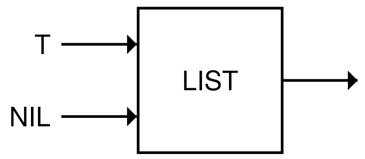
As you can see now, the SAY-WHAT function doesn't really replace any part of the list. What it does is generate a new list by making a new cons cell whose CDR half points to a portion of the old list. The input to SAY-WHAT and the result it returns are both shown below.

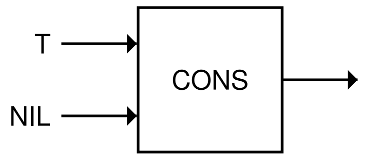


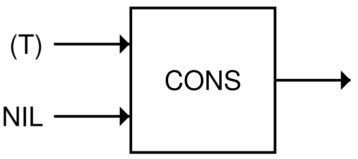
EXERCISES

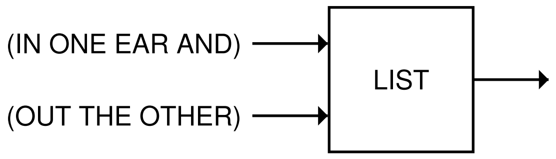
2.20. What results are returned by the following?

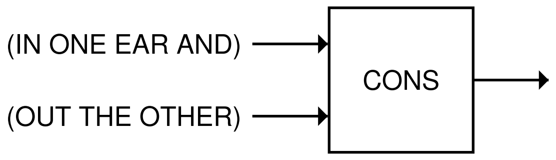












2.21. Write a function that takes four inputs and returns a two-element nested list. The first element should be a list of the first two inputs, and the second element a list of the last two inputs.

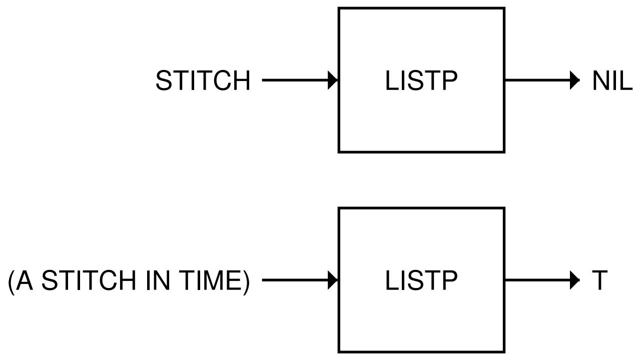
2.22. Suppose we wanted to make a function called DUO-CONS that added two elements to the front of a list. Remember that the regular CONS function adds only one element to a list. DUO-CONS would be a function of three inputs. For example, if the inputs were the symbol PATRICK, the symbol SEYMOUR, and the list (MARVIN), DUO-CONS would return the list (PATRICK SEYMOUR MARVIN). Show how to write the DUO-CONS function.

2.23. TWO-DEEPER is a function that surrounds its input with two levels of parentheses. TWO-DEEPER of MOO is ((MOO)). TWO-DEEPER of (BOW WOW) is (((BOW WOW))). Show how to write TWO-DEEPER using LIST. Write another version using CONS.

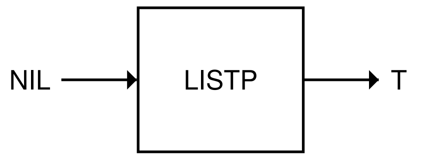
2.24. What built-in Lisp function would extract the symbol NIGHT from the list (((GOOD)) ((NIGHT)))?

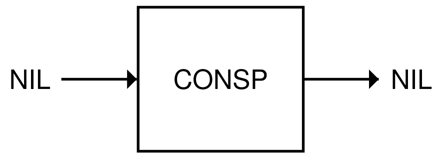
2.15 LIST PREDICATES

The LISTP predicate returns T if its input is a list. LISTP returns NIL for non-lists.

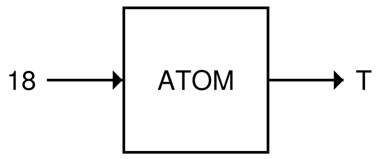


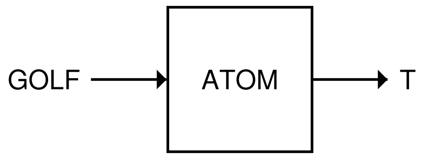
The CONSP predicate returns T if its input is a cons cell. CONSP is almost the same as LISTP; the difference is in their treatment of NIL. NIL is a list, but it is not a cons cell.

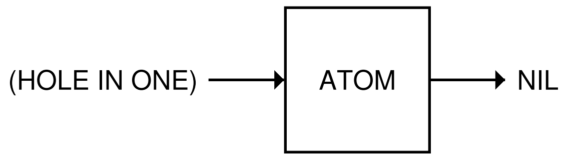




The ATOM predicate returns T if its input is anything other than a cons cell. ATOM and CONSP are opposites; when one returns T, the other always returns NIL.







The word ‘‘atom’’ comes from the Greek *atomos*, meaning indivisible. Numbers and symbols are atomic because they cannot be taken apart. Nonempty lists aren’t atomic: FIRST and REST take them apart.

The NULL predicate returns T if its input is NIL. Its behavior is the same as the NOT predicate. By convention, Lisp programmers reserve NOT for logical operations: changing *true* to *false* and *false* to *true*. They use NULL when they want to test whether a list is empty.

SUMMARY

This chapter introduced the most versatile data type in Lisp: lists. Lists have both a printed and an internal representation. They may contain numbers, symbols, or other lists as elements.

We can take lists apart using CAR and CDR (‘‘first’’ and ‘‘rest’’) and put them together with CONS or LIST. The LENGTH function counts the number of elements in a list, which is the same as its number of top-level cons cells.

The important points about CAR and CDR are:

- CAR and CDR accept only lists as input.
- FIRST and REST are the same as CAR and CDR.
- SECOND and THIRD are the same as CADR and CADDR.
- Common Lisp provides built-in C...R functions for all combinations of CAR and CDR up to and including four As and Ds.

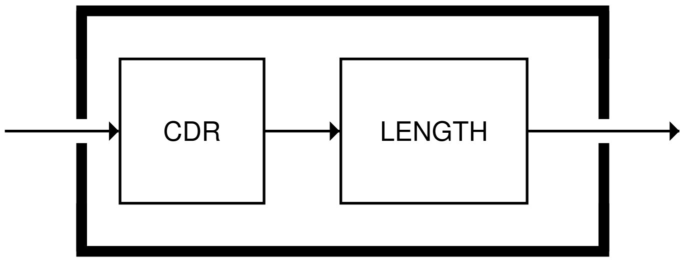
The symbol NIL has several interesting properties:

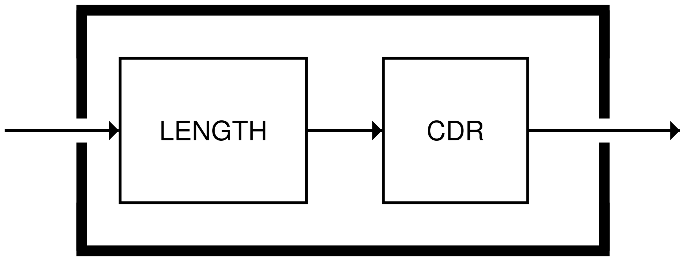
- NIL is a symbol. It is the only way to say ‘‘no’’ or ‘‘false’’ in Lisp.
- NIL is a list. It is the empty list; its LENGTH is zero.
- NIL is the only Lisp object that is both a symbol and a list.
- NIL marks the end of a cons cell chain. When lists are printed in parenthesis notation, the NILs at the end of chains are omitted by convention.
- NIL and () are interchangeable notations for the same object.
- The CAR and CDR of NIL are defined to be NIL.

REVIEW EXERCISES

2.25. Why do cons cells and the CONS function share the same name?

2.26. What do these two functions do when given the input (A B C)?





2.27. When does the internal representation of a list involve more cons cells than the list has elements?

2.28. Using just CAR and CDR, is it possible to write a function that returns the *last* element of a list, no matter how long the list is? Explain.

FUNCTIONS COVERED IN THIS CHAPTER

List functions: FIRST, SECOND, THIRD, FOURTH, REST, CAR, CDR, CONS, LIST, LENGTH.

Compositions of CAR and CDR: CADR, CADDR, and so on.

Predicates: LISTP, CONSP, ATOM, NULL.