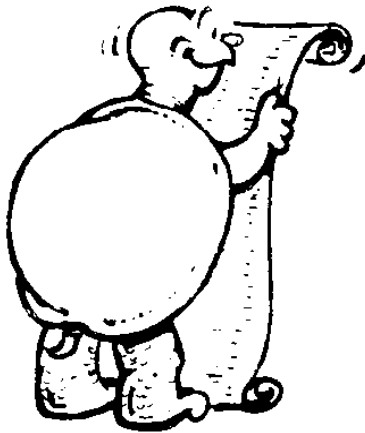


Chapter 12. Talk To Your Computer

Just when you thought you were getting good at this, here comes a whole new Logo world to explore, the world of list processing. It's one of the more interesting things about the Logo language. In fact, while most people think about Logo as a graphic language, it is actually based on Lisp, a list processing language. Turtle graphics were added later.



Logo is called a high level language. The first real computer language was machine language, which meant programming the computer using 0's and 1's. Then came assembly languages followed by other early computer languages that became known as “number crunchers.” Everything they do deals with numbers: money, inventories, counting, calculating, and things like that.

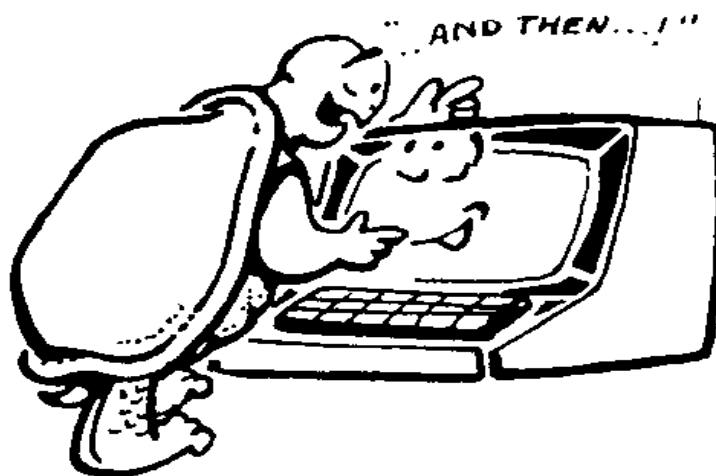
Logo is different. It uses “symbolic computation” that allows you to process ideas.

Just what does symbolic computation mean? Actually, it gets sort of complicated. For our purpose — which is to keep things simple — let's just say it means that in addition to numbers, you can process words and lists of words, numbers, and things.

You can add two lists together or add words to a list. And you can teach the computer to remember them. This is very important to the study of Artificial Intelligence, or AI. You'll get introduced to that later in the this chapter.

In the meantime...

Did you ever talk to your computer?



Here's a short and simple procedure to get you started. Type it, or load it (TALK.LGO), and then run it. What happens?

It's like a short conversation, isn't it? But, no, you're not really talking to the computer.

```
TO TALK
CT PRINT [HI! WHAT'S YOUR NAME?]
MAKE "NAME READWORD
MAKE "REPLY SE :NAME~
  [MY NAME IS ERNESTINE.]
PRINT SE [I DON'T THINK WE'VE MET,] :REPLY
PRINT [HAVE YOU EVER TALKED TO A ~
  COMPUTER?]
TEST READWORD = "NO
IFFALSE [PRINT ~
  [WOW! DO YOU TALK TO TURTLES, TOO?]]
IFTRUE [PRINT [OH BOY! A BEGINNER.]]
END
```

Logo Sentences

Take a look at the TALK procedure. There's really only one thing new there.

Do you see how SENTENCE works? SENTENCE, or SE for short, takes two inputs and prints them together. These can be two words, two lists of words, or a combination of words, lists, characters, or numbers. The output is a list.

```
PRINT SE [I DON'T THINK WE'VE MET,] :NAME
```

I DON'T THINK WE'VE MET, is a list that is the first input. The variable :NAME is the second. This is the word you typed when the procedure asked you for your name.



SPECIAL NOTE: When you want to add more than two inputs to SENTENCE, or when you add additional inputs to PRINT, TYPE, SHOW, WORD, or LIST, use parentheses. For example:

```
PR (SE :NAME ", [HOW ARE YOU?])
```

Let's say that :NAME is the variable with the value of Ernestine. This line is displayed as

```
ERNESTINE, HOW ARE YOU?
```

While it may seem strange, Logo treats the comma as a separate word. Anything that follows a single quotation mark is considered to be a word. That word ends with a space.

Logo Numbers, Characters, Words, and Lists

OK! You've explored a bit with Logo numbers, characters, words, and lists. You've played a bit with a few list processing commands too. But there is much, much more to list processing.

So let's review where we've been. It may seem strange, but when you're exploring so many different things, there are times when you have to look back to know where you're going. This is one of those times.

“First, what do you really mean by list processing?”

“One of the really neat things about Logo is that it allows you to process information, or data, in many different forms. It can be numbers, words, lists of other words and lists, property lists, or arrays.”

Numbers

Let's start with Logo numbers.

Numbers consist of one or more integers, decimals, or exponents such as engineering notation. Remember integers? Integers are “whole” numbers such as 3, 25, 423, or 1,324,598. Fractions and decimals are not whole numbers. They are parts of whole numbers, even if you have something like 1.5. 1.5 is part of 2, isn't it?

All the commands shown below use whole numbers.

| | |
|-----------|----------|
| FORWARD 5 | SETH 45 |
| REPEAT 4 | SETH 0 |
| RIGHT 20 | SETH -90 |

Characters

Logo can also display alphabetic and numeric characters using the CHAR or ASCII (American Standard Code for Information Interchange) primitives. You've may have heard about ASCII code before. It is a standard code that is used by computer manufacturers to display a set of 128 characters numbered from 0 to 127. These codes describe all the punctuation marks, upper and lower case letters. There's also a high-level set of codes that go from 128 to 255

Take a look.

SHOW CHAR 65 displays the letter A.

SHOW CHAR 67 displays the letter C.

SHOW CHAR 49 displays the number 1.

SHOW CHAR 32 displays a space.

Yes, there's even a code for a blank space.

CHAR followed by a code number displays the letter, number, or punctuation mark for the specific ASCII code. But, no, you don't have to memorize ASCII code.

If you ever want to find out the ASCII code for a something, type

SHOW ASCII *<character>*

SHOW ASCII "A displays 65.

SHOW ASCII "a displays 97.

Talk To Your Computer

Upper and lower case letters each have different codes. That's because they're different shapes.

Try a few ASCII and CHAR commands so that you get a good feel for what they do.

ASCII Art

You may have seen cute little characters added to e-mail and other messages — things like a happy face (:>) or the sad face (:>(Some people do it like this: :-) or :-(

There are all sorts of these little additions. It's a way to add some expression to what can sometimes be dull on-line text.

And, yes! You can add these touches to your procedures. For example, you can display your signature every time you create a picture.

```
TO SIGNATURE
CT TYPE [GRAPHICS BY JIM]
TYPE CHAR 32
TYPE CHAR 40
TYPE CHAR 58
TYPE CHAR 62
PR CHAR 41
END
```

Run this procedure and

```
GRAPHICS BY JIM (:>)
```

is printed in the Commander window.

You can also write it like this:

```
TO ARTIST
(TYPE [GRAPHICS BY JIM] CHAR 32 CHAR 40
 CHAR 58 CHAR 62) PR CHAR 41
END
```

Spacing

Before you leave this procedure, there's something else to look at.

```
(TYPE [GRAPHICS BY JIM] CHAR 32 CHAR 40
 CHAR 58 CHAR 62) PR CHAR 41
```

Why not include everything with the TYPE command? Why change to the PRINT command at the end? Or, why not just use PRINT in parentheses?

```
(PR [GRAPHICS BY JIM] CHAR 32 CHAR 40
 CHAR 58 CHAR 62 CHAR 41)
```

TYPE and PRINT do essentially the same thing, except that TYPE does not add a space between the characters or a carriage return at the end of the line.

- TYPE prints the string of characters all on one line. You must include a PRINT or SHOW command at the end of the command line to move to the next line.
- PRint and SHOW add a space between each character and a carriage return after printing the string of text, which sends the cursor to the next line.

Try these commands:

```
TYPE "TRY TYPE "THIS
```

What happened? Nothing — because there wasn't a PRINT or SHOW command at the end. So try this:

Talk To Your Computer

```
TYPE "TRY TYPE "THIS SHOW "  
TRYTHIS
```

There is no space between the two words. What about

```
SHOW "TRY SHOW "THIS  
TRY  
THIS
```

One more time, this time with a single space after the
backslash. What happens?

```
(TYPE "TRY "\ "THIS) SHOW "
```

What happens when you insert two spaces after the
backslash? What's that tell you about the backslash?

Play around with the spacing when using different
commands. This comes in handy at times, especially if you
want to start displaying some fancy ASCII artwork.

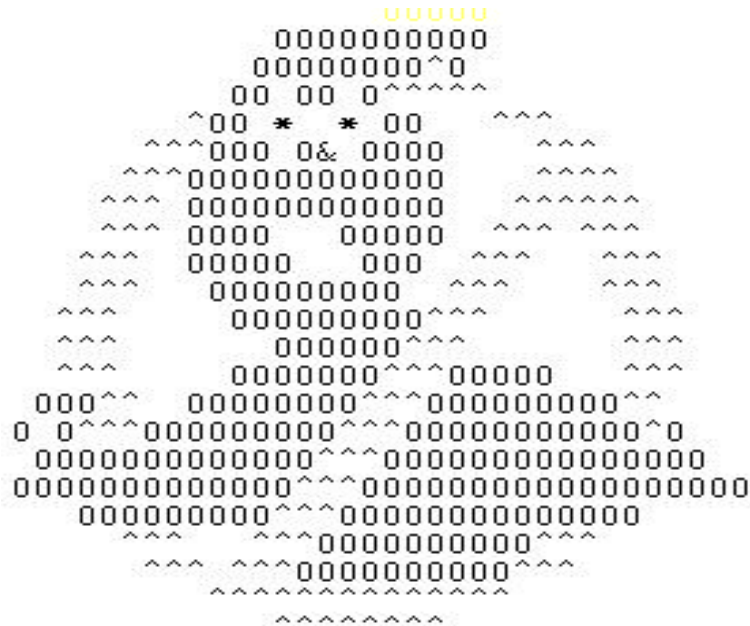
```
\ /  
|V|  
| ^ |  
/ \   
|o o|  
|..|  
V--V  
 \ /
```



SPECIAL NOTE: When you want to run text procedures such as ARTIST, ASCII, and SIGNATURE, open the Commander window. That's where they are displayed. Either drag the edge up, or left-click on the middle of the three boxes on the right of the Title bar.

Take a look at ASCII.LGO file \procs\chpt12 subdirectory for an interesting twist on ASCII art.

This one's a famous movie title of some years back.



Add Some Pizzazz

After TurtleBusters, why not add some Pizzazz to your signature? Try this:

```

TO SIGNATURE
CT REPEAT 10 [TC 32]
TC 71 TC 114 TC 97 TC 112 TC 104 TC 105 TC 99
TC 115 TC 32 TC 66 TC 121 TC 32 TC 74 TC 105
TC 109
FLASHER 1 1
END

TO TC :C
TYPE CHAR :C
END
    
```

Talk To Your Computer

```
TO FLASHER :X :Y
CT REPEAT :X [PR "]
REPEAT :Y [TC 32]
TC 71 TC 114 TC 97 TC 112 TC 104 TC 105 TC 99
TC 115 TC 32 TC 66 TC 121 TC 32 TC 74 TC 105
TC 109 PR "
MAKE "X :X + 1
MAKE "Y :Y + 5
IF :X > 24 [MAKE "X 1 MAKE "Y 1]
WAIT 3 FLASHER :X :Y
END
```

Put your own name in this procedure (FLASHER.LGO) and then run it. If you can't find a list of the ASCII codes, why not write a procedure that prints them out for you?

That's not as hard as you might think. So why not give it a try. You can do it.

This new SIGNATURE procedure is a step in the right direction, but it really isn't that flashy. What else can you add to this?

Add Some Real Pizzazz

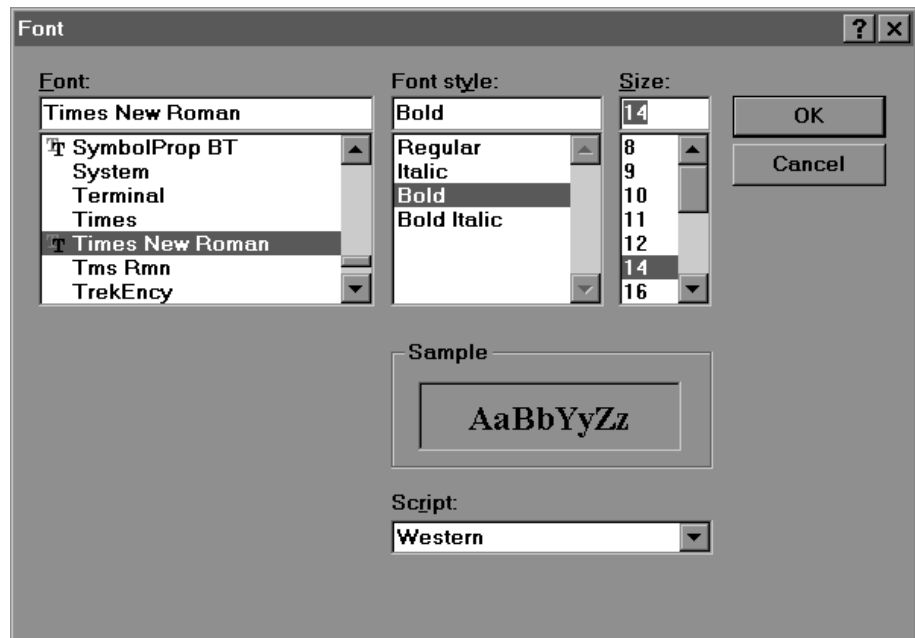
But what about the graphics screen? Why not see what you can do to add a flashy signature to your drawings on the graphics screen?

LABEL is one of those commands that means different things in different versions of Logo. In MSW Logo, LABEL's input, which may be a word or a list, is printed on the graphics screen.

To define the font in which the word or list is to be displayed, you have two choices:

1. Open the **Select** menu and left-click on **Font**.

The Font selection window is displayed.



Here you can select the Font. This defines what the text is going to look like. Left-click on a font to see what it looks like in the Sample window.

You can change the style of the font to display:

- regular type.
- type in italics
- bold type
- bold italics.

The third window is where you select the size you want to display. Type is measured in points, 72 points to the inch. This book is printed in 14 point Times New Roman to give you some idea of size. The Sample window also helps you select the size you want.

The other method for selecting a font is to use the SETTEXTFONT command.

SETTEXTFONT [[font] <attribute list>]

The input to the MSW Logo command, SETTEXTFONT, describes a font and the attributes it is to display: the size of the type and whether it is to be in bold, italics, underlined, etc.

MSW Logo gives you a wide range of attributes that define exactly how the text is to appear. These include:

SETTEXTFONT

```
[  
[Font]  
Height  
Width  
Orientation  
Weight  
Italic  
Underline  
StrikeOut  
CharSet  
OutPrecision  
ClipPrecision  
Quality  
PitchAnd-Family  
]
```

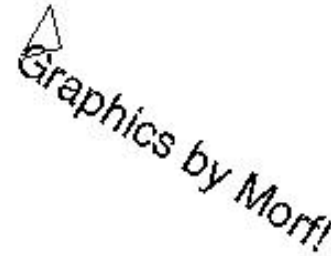
The on-line help file describes each of these attributes. However, you don't have to use any of them if you don't want to. You can accept the default values. Just type

SETTEXTFONT []

You may want to specify the size of the type. So type the name of the font followed by a height and width.

```
SETTEXTFONT [[HELV] 24 16]  
LABEL [Graphics by Morf!]
```

MSW Logo prints in upper and lower case. It also prints in the direction the turtle is facing.



Graphics by Morf!

Why not experiment with the font attributes? See what type of crazy pizzazz you can come up with. Then you can get back to list processing.

Logo Words

Logo words are easy. They're anything between a quotation mark to the left and a space on the right. However, there can not be any spaces in a word.

Just remember that when you identify a Logo word, you must use a quotation mark in front of it. You don't have to do that with numbers. Logo treats 3 as a number just as it treats 325491587 as a number.

As you have seen, Logo treats a single letter or a punctuation mark as a word. It treats

```
PRINT "T the same as
```

```
PRINT "Tyranosaurus
```

Of course, it isn't really strange at all once you think about it. "A" is a word, isn't it? It's one of the three articles in the English language. The other two are "an" and "the."

Talk To Your Computer

What about the word "I?"

A Logo word is any combination of letters, numbers, and punctuation marks with no spaces.

```
SHOW "ABC_456  
  displays ABC_456.
```

```
SHOW "A93HK8  
  displays A93HK8.
```

SENTENCE and WORD

As you saw in the TALK procedure, SENTENCE produces a list. However, WORD, produces another word.

To see the difference, try something like this:

```
PRINT WORD "A "B  
PRINT SENTENCE "A "B
```

or this:

```
PRINT WORD 1 2  
PRINT SE 1 2
```

Hmmmm, what would happen if we did this:

```
MAKE "A WORD 1 2  
MAKE "B WORD 3 4  
PRINT :A + :B
```

Try this:

```
MAKE "C WORD :A :B  
PRINT :C
```

What happened? Bet you got 1234, right? So let's do some testing to see what we've got here.

```
PRINT NUMBERP :C
```

or

```
IF NUMBERP :A [PRINT "TRUE]
```

Remember NUMBERP? If you want to review, go back to the *Great Math Adventure* chapter.

The statements above tell Logo that if :A is a number, print TRUE. Another way to say this is:

```
IF NOT NUMBERP :C [PRINT "FALSE]
```

Even though you created a word from :A and :B, it's still a number, right?

Before we leave this confusion, let's mess it up some more. What would this line print?

```
PRINT (WORD 1 CHAR 32 2 CHAR 32 3)
```

The result would look something like a sentence but would actually be a word.

```
1 2 3
```

This actually reads as 1 space 2 space 3, because the spaces are deliberately inserted as characters. In Logo, that's a word.

Just to check the difference, try it this way:

```
PRINT (WORD 1 2 3)
```

Where'd the spaces go?

Just to be absolutely sure, try this one:

```
MAKE "CHECK (WORD 1 CHAR 32 2 CHAR 32 3)
SHOW WORDP :CHECK
```

The result **True** is displayed. So I guess that the output is a word, not a list.

Lists

Lists are elements within square brackets — []. A List can include words, numbers, or other lists. Of course, if a List contains a list, both have to be enclosed in brackets.

```
PRINT [This is a list.]
```

displays

This is a list.

```
PRINT [1 2 3 this is also a list_A B C?]
```

displays

1 2 3 this is also a list_A B C?

What about this?

```
PRINT [1 2 3 [this is also a list_A B C]]
```

Here's a list within a list that displays as:

```
1 2 3 [this is also a list_A B C]
```

Do you remember the difference between how PRINT displays lists and how SHOW displays them? Check it out.

Creating Lists

You can, of course, create lists. You started this in Chapter 8 when you used `SETPOS LIST :X :Y`. Now lets try something different:

```
MAKE "SAMPLE (LIST 1 2 3 45 98 "PR "JIM)
SHOW :SAMPLE
```

[1 2 3 45 98 PR JIM] is displayed.

In the previous section on words, you asked if `:CHECK` was a word?

```
MAKE "CHECK (WORD 1 CHAR 32 2 CHAR 32 3)
SHOW WORDP :CHECK
```

Now change that second line to

```
SHOW LISTP :CHECK
```

What's the response going to be, TRUE or FALSE?

LIST or []

When do you use the `LIST` command? When do you use brackets? Why does `SETPOS LIST :X :Y` work when `SETPOS [:X :Y]` doesn't?

There is a very simple but very important rule that explains this. Brackets do not execute what's inside them. They merely list the contents. `LIST` makes things happen.

Remember this when writing procedures.

Talk To Your Computer

Logo Postcards

Has your family ever been on a vacation and sent lots of postcards to friends back home? Those cards seem to be all the same, don't they.

Dear _____,

Here we are in wonderful _____. We're having a great time _____ and _____. The weather is very _____. We'll be home _____. See you then.

Love,

Wouldn't it be nice to have a procedure to print your cards for you.

TO SETUP

PRINT [WHO IS THIS CARD FOR?]

MAKE "FRIEND READWORD

PRINT [WHERE ARE YOU WRITING FROM?]

MAKE "VAC READWORD

MAKE "VAC WORD :VAC ".

PRINT [WHAT ARE YOU DOING?]

MAKE "ACT1 READWORD

PRINT [WHAT ELSE?]

MAKE "ACT2 READWORD

MAKE "ACT2 (WORD :ACT2 ".)

PRINT [HOW'S THE WEATHER?]

MAKE "WEA READWORD

MAKE "WEA WORD :WEA ".

PRINT [WHEN WILL YOU BE HOME?]

MAKE "ARR READWORD

MAKE "ARR WORD :ARR ".

PRINT [HOW WILL YOU SIGN THE CARD?]

```
MAKE "SIG READWORD  
END
```

Now we'll print this information on the postcards.

```
TO POSTCARD  
SETUP CT TS  
PRINT SE [DEAR] :FRIEND  
PRINT "  
TYPE SE [HERE WE ARE IN WONDERFUL] :VAC  
  PRINT [WE'RE HAVING]  
TYPE SE [A GREAT TIME] :ACT1 TYPE SE "AND  
  :ACT2) PRINT "THE  
TYPE SE [WEATHER IS] :WEA PRINT SE [WE'LL BE  
  HOME] :ARR  
PRINT [SEE YOU THEN.]  
PRINT "  
PRINT [LOVE,]  
PRINT "  
PRINT :SIG  
END
```

Do you remember the difference between PRINT and TYPE?

TYPE prints its input without adding a carriage return at the end the way that PRINT does. TYPE lets you print text more like you do with word processing software. PRINT only allows you to print one string of text on a line. Of course, you can also write the line as

```
PR (SE [A GREAT TIME] :ACT1 "AND :ACT2 "  
  "THE)
```

See! There's usually an easier way if you'll look for it. But there's still a problem with the postcard procedure.

Making Headlines

Look at any book on Logo and usually you'll find a procedure that picks parts of speech and combines these into sentences — something like this:

```
TO HEADLINES
MAKELISTS
PRINTHEADLINES
END
```

```
TO MAKELISTS
MAKE "ADJ [HAPPY GLAD ANXIOUS GOOD
SINCERE]
MAKE "NOUN [SANTA PEOPLE FRIENDS
PARENTS]
MAKE "VERB [GAVE RECEIVED SPREAD
SHAREDWISHED]
MAKE "OBJ [GIFTS LOVE CHEER GLADNESS JOY
HAPPINESS]
END
```

```
TO PRINTHEADLINES
PR (SE PICK :ADJ PICK :NOUN PICK :VERB PICK
:OBJ)
WAIT 10
PRINTHEADLINES
END
```

When you type HEADLINES, the first thing that Logo does is run the MAKELISTS procedure to make up lists of words: ADJ, NOUN, VERB, and OBJ. Then the next procedure is called.

PRINTHEADLINES tells the computer to PRInt a SEntence — but what sentence? PICK is a command that randomly selects an item from a list. So it randomly picks elements from the four lists and prints them as a sentence.

Some versions of Logo don't have a PICK command. You can write one like this:

```
TO PICK :WORDS
  OUTPUT ITEM (1 + RANDOM (COUNT :WORDS))
  :WORDS
END
```

That's fine. But what's with all the parentheses?

To understand any Logo statement, you start at the left and move to the right, one command at a time. OUTPUT takes one input. That input is ITEM.

ITEM takes two inputs: a number and something else, like the 4th list, the 2nd word. The next thing you see after ITEM is in parenthesis. That means that

```
(1 + RANDOM (COUNT :WORDS))
```

is all one element. RANDOM typically selects a number between 0 and, in this case COUNT :WORDS. But again, what about all those parentheses?

The easiest way to read those is to start with the parentheses on the inside:

```
(COUNT :WORDS)
```

Talk To Your Computer

The first task for PICK in the PRINTHEADLINES procedure is to select an adjective — PICK :ADJ. So, in the first case, (COUNT :WORDS) temporarily becomes

(COUNT :ADJ)

How many adjectives (:ADJ) are inside the brackets in the MAKELISTS procedure? There are five adjectives, right? So, if you COUNT the ADJectives, you get 5. Substitute 5 in the original procedure and here's what you get:

(1 + RANDOM (5)) which is the same as 1 + RANDOM 5 after you take the unneeded parentheses away.

1 + RANDOM 5 gives you the five numbers 1, 2, 3, 4 and 5. So now we have:

OUTPUT ITEM RANDOM 5 :WORDS

or

OUTPUT a randomly selected item from the five words in the ADJ list.

Now the following line begins to make a bit more sense:

PR (SE PICK :ADJ PICK :NOUN PICK :VERB PICK :OBJ)

Logo is going to print a random ADJ, a random NOUN, a random VERB, and a random :OBJ. You'll get sentences like

HAPPY SANTA GAVE CHEER
GLAD PEOPLE RECEIVED GIFTS
ANXIOUS PARENTS WISHED JOY

HAPPY PARENTS SPREAD LOVE
HAPPY PEOPLE GAVE GIFTS
SINCERE FRIENDS SHARED LOVE
ANXIOUS PEOPLE RECEIVED CHEER

Explore these headlines some more, why don't you?
Make a longer headline using five or more words. Combine words and lists. You can even throw in a sentence or two.

The main thing is to explore, to learn, and to have some fun doing both.

Word Games

Logo postcards and headlines may seem a bit silly. But what about a word game — like the game of Nim. This also gives you the chance to see more Windows commands in action.

Nim is a game the computer always wins. Can you figure out why? Take a look at the NIM.LGO procedure in the `\procs\chpt12` subdirectory to find the answer.

Before you start digging into the windows programming tools used in Nim, here's a couple of challenges for you.

1. Change the game so that the computer does not always win.
2. Word games aren't very pretty. Why not change this into a graphic game where you pick real stones, or trees, or shapes?
3. Set up different rows of objects so the player has to select which row to select from?

What other things can you dream up?

Windows Programming

"Logo, I'm having enough trouble learning Logo for Windows. Now you want me to learn Windows programming?"

"Well, not exactly. However, to use some of the features of MSW Logo, you have to learn a few of the things about what makes windows programming different from other Logo programming. What follows are a few examples to explore. For a better description, read the Windows Functions section of On-line Help."

Up until now, you've seen how Logo reads one line at a time and then interprets that line from left to right. That's the Modal mode where the application is in control. Windows programming supports two modes or methods of programming, Modal and Modeless.

Modal Mode

Those of you who have done some DOS programming before are familiar with the Modal mode. This is where the program or procedure controls the action. It is the type of programming used in the NIM procedure above.

DIALOGCREATE is used to stop the action of a procedure while it asks you for some information. In the case of NIM, it asks you to pick a number.

DIALOGCREATE creates a dialog box in which you can add radiobuttons, scrollbars, listboxes, checkboxes, and the other types of input devices you see in windows applications. DIALOGCREATE is similar to the WINDOWCREATE command described below, except it will not return to the calling procedure until the dialog is closed.

Dialog boxes aren't the only devices that stop the action. NIM shows you examples of some others: the YESNOBOX and the MESSAGEBOX. There's also the QUESTIONBOX. Change the OVER procedure to something like this and see what happens.

```

TO OVER
PR [There is one stone left. I win again.]
MAKE "ANS QUESTIONBOX [Nim] ~
  [Want to play again?]
IFELSE (FIRST FIRST :ANS) = "Y [NIM][CT PR ~
  [Bye for now!]]
END

```

Modeless Mode

The Modeless mode is just the opposite of Modal mode. In the Modeless mode, the Window (user) is in control. This takes some getting used to but is a very important idea.

In the Modeless Mode, you use the WINDOWCREATE command as in the MODELESS procedure listed below.

```

TO START
CS CT
PR [Let's draw a shape.]
PR [How many sides should it have?]
MODELESS
END

TO MODELESS
WINDOWCREATE "ROOT "SELECTOR ~
  "SELECTBOX 250 0 100 180 []
GROUPBOXCREATE "SELECTOR "NIM 10 10 80 140
RADIOBUTTONCREATE "SELECTOR "NIM "ONE ~
  [ONE] 20 15 60 15
RADIOBUTTONCREATE "SELECTOR "NIM "TWO~
  [TWO] 20 30 60 15

```

```
RADIOBUTTONCREATE "SELECTOR "NIM
  "THREE [THREE] 20 45 60 15
RADIOBUTTONCREATE "SELECTOR "NIM ~
  "FOUR [FOUR] 20 60 60 15
RADIOBUTTONCREATE "SELECTOR "NIM ~
  "FIVE [FIVE] 20 75 60 15
RADIOBUTTONCREATE "SELECTOR "NIM ~
  "SIX [SIX] 20 90 60 15
RADIOBUTTONCREATE "SELECTOR "NIM
  "SEVEN [SEVEN] 20 105 60 15
RADIOBUTTONCREATE "SELECTOR "NIM ~
  "EIGHT [EIGHT] 20 120 60 15
RADIOBUTTONCREATE "SELECTOR "NIM ~
  "NINE [NINE] 20 135 60 15
RADIOBUTTONSET "ONE "TRUE
RADIOBUTTONSET "TWO "FALSE
RADIOBUTTONSET "THREE "FALSE
RADIOBUTTONSET "FOUR "FALSE
RADIOBUTTONSET "FIVE "FALSE
RADIOBUTTONSET "SIX "FALSE
RADIOBUTTONSET "SEVEN "FALSE
RADIOBUTTONSET "EIGHT "FALSE
RADIOBUTTONSET "NINE "FALSE
BUTTONCREATE "SELECTOR "GAME "OK ~
  35 150 25 20 [EXECUTE]
END
```

```
TO EXECUTE
IF RADIOBUTTONGET "ONE [HT GREET STOP]
IF RADIOBUTTONGET "TWO [ST GREET STOP]
IF RADIOBUTTONGET "THREE [CS REPEAT 3 ~
  [FD 50 LT 120] GREET STOP]
IF RADIOBUTTONGET "FOUR [CS REPEAT 4 ~
  [FD 50 LT 90] GREET STOP]
IF RADIOBUTTONGET "FIVE [CS REPEAT 5 ~
  [FD 50 LT 72] GREET STOP]
IF RADIOBUTTONGET "SIX [CS REPEAT 6 ~
  [FD 50 LT 60] GREET STOP]
```

```
IF RADIOBUTTONGET "SEVEN [CS REPEAT 7 ~  
  [FD 50 LT 360/7] GREET STOP]  
IF RADIOBUTTONGET "EIGHT [CS REPEAT 8 ~  
  [FD 50 LT 45] GREET STOP]  
IF RADIOBUTTONGET "NINE [CS REPEAT 9 ~  
  [FD 50 LT 40] GREET STOP]  
END
```

```
TO GREET  
PR [How about that!]  
END
```

```
TO DEL  
WINDOWDELETE "SELECTOR  
END
```

In MODELESS, the dialog box created by WINDOWCREATE doesn't need to be deleted. You can continue to press buttons to draw all the shapes you want. Type DEL to delete the dialog box.

This is barely a taste of what you can do with windows and dialog boxes. To learn more, explore the on-line help files for each of these new commands. Most importantly, explore the examples given for each command. For example, here's a slight variation of the DODRAW procedure listed for the LISTBOXADDSTRING command.

```
TO DEL  
WINDOWDELETE "MYWINDOW  
END
```

```
TO DODRAW  
;SELECT FIGURE FROM LISTBOX AND THEN ~  
  CLICK ON DRAW BUTTON  
CS
```

Talk To Your Computer

```
IF EQUALP [TRIANGLE] LISTBOXGETSELECT ~  
  "MYLIST [REPEAT 3 [FD 100 RT 120]]  
IF EQUALP [SQUARE] LISTBOXGETSELECT ~  
  "MYLIST [REPEAT 4 [FD 100 RT 90]]  
END
```

```
TO START  
WINDOWCREATE "MAIN "MYWINDOW ~  
  "MYTITLE 0 0 100 100 []  
LISTBOXCREATE "MYWINDOW "MYLIST ~  
  25 0 50 50  
LISTBOXADDSTRING "MYLIST [TRIANGLE]  
LISTBOXADDSTRING "MYLIST [SQUARE]  
BUTTONCREATE "MYWINDOW "MYDRAW ~  
  "DRAW 25 50 50 25 [DODRAW]  
END
```

For a better example, take a look at the MODELESS procedure in the Examples\Windows directory that was setup when you installed MSW Logo.

More Fun With Text

Among the first things you did in this book was to make different shapes. Well, how about making word shapes? That means making shapes out of the text you print on the screen. In addition to being a fun project, it gives you some practice with some new list processing commands.

Here's an easy one to start with. It's on your CD as WORDTRI.LGO. What's the new command?

```
TO WORDTRIANGLE :WORDS  
IF :WORDS = " [STOP]  
PR :WORDS  
WORDTRIANGLE BUTFIRST :WORDS  
END
```

This procedure simply prints :WORDS and then calls itself BUTFIRST :WORDS. BUTFIRST, BF for short, means it prints everything BUT the FIRST element of the variable :WORDS. You'll see how this works as we go along.

WORDTRIANGLE "LOGOADVENTURES

The result:

LOGOADVENTURES
OGOADVENTURES
GOADVENTURES
OADVENTURES
ADVENTURES
DVENTURES
VENTURES
ENTURES
NTURES
TURES
URES
RES
ES
S

And this raises some questions:

- What would happen if :WORDS was a list? What would you have to change to drop the first letter of each word?
- What would you have to do to have the procedure drop the last letter?

Maybe if you turn that triangle around, you'll get some ideas.

Turning It Around

Here's one very simple way to turn the triangle around. All you have to do is more the PR :WORDS command.

```
TO WORDTRIANGLE :WORDS
IF :WORDS = " [STOP]
WORDTRIANGLE BUTFIRST :WORDS
PR :WORDS
END
```

These two word triangle procedures show the difference between tail-end and embedded recursion. And while this second procedure does turn the triangle around, it

Have the procedure print the first letter, then the first and second, then the first three, and so on. Along the way, you'll see how some other list processing commands work. (This is WORDTRI2.LGO on the Sourcebook diskette.)

You've already seen the command FIRST. It selects the first element of a word or list. In the START procedure, after clearing all the text, Logo prints the FIRST element of the variable :WORDS. If you use the same variable again, an L is displayed.

```
TO START :WORDS
CT PR FIRST :WORDS
MAKE "LINE WORD (FIRST :WORDS) ~
(FIRST BUTFIRST:WORDS)
PR :LINE
REST BF BF :WORDS
END
```

After the L is displayed, Logo makes a new word, :LINE, by combining FIRST :WORDS with FIRST BUTFIRST :WORDS. What would that be?

In the WORDTRIANGLE procedure, you found out that BUTFIRST :WORDS printed everything but the first element of :WORDS. Using the traditional Logo parsing technique, start at the right with FIRST. That takes one input, which is BUTFIRST :WORDS. BUTFIRST :WORDS removes the first element but leaves the rest of the variable there. So what does FIRST BUTFIRST do?

You got it. It prints the first element in BUTFIRST :WORDS, or 0. So here's what you have when the START procedure runs:

```
L  
LO
```

This sets up the new triangle. Since you don't want to repeat these steps, you move on to the REST procedure for the rest of the triangle.

```
TO REST :WORDS  
IF :WORDS = " [STOP]  
MAKE "LINE WORD :LINE (FIRST :WORDS)  
PR :LINE  
REST BF :WORDS  
END
```

If you'll follow the printout below, you'll see how it works. REST picks up the variable :LINE from START. After START has run, :LINE has a value of . :WORDS has a value of

```
GOADVENTURES
```

Talk To Your Computer

```
MAKE "LINE WORD :LINE (FIRST :WORDS)
```

Make "LINE a word by combining LO and FIRST :WORDS or G. This keeps repeating: LOG, then LOGO, etc. Get the idea?

Here's what it looks like:

```
L  
LO  
LOG  
LOGO  
LOGOA  
LOGOAD  
LOGOADV  
LOGOADVE  
LOGOADVEN  
LOGOADVENT  
LOGOADVENTU  
LOGOADVENTUR  
LOGOADVENTURE  
LOGOADVENTURES
```

We've got one more triangle to check out. How would you make this one?

A Logo Tree

Pick your way through this one. You've been through the other two so this one should be easy.

```
TO START :WORDS  
CT TC ((COUNT :WORDS) / 2) PR FIRST :WORDS  
MAKE "LINE (WORD (FIRST :WORDS) ~  
  (FIRST BF :WORDS) (FIRST BF BF :WORDS))  
TC ((COUNT :WORDS) / 2) - 1 PR :LINE
```



```

REST BF BF BF :WORDS
END
TO TC :N
IF :N < 1 [STOP]
REPEAT ROUND :N [TYPE CHAR 32]
END

```

```

TO REST :WORDS
IF :WORDS = " [STOP]
IF COUNT :WORDS = "1 ~
  [MAKE "WORDS WORD :WORDS CHAR 32]
MAKE "LINE (WORD :LINE (FIRST :WORDS)~
  (FIRST BF :WORDS))
TC ((COUNT :WORDS) / 2) - 1 PR :LINE
REST BF BF :WORDS
END

```

You've already explored COUNT. This tells you that to make an even tree shape, you have to type blank spaces (CHAR 32) to take you to the middle of :WORDS — COUNT :WORDS / 2. The rest shouldn't be too difficult to follow.

```

L
LOG
LOGOA
LOGOADV
LOGOADVEN
LOGOADVENTU
LOGOADVENTURE
LOGOADVENTURES!

```

Once you get that one figured out, you're on your own. What other word shape procedures can you make?

How about changing all the FIRST commands to LAST, and all the BUTFIRST commands to BUTLAST. What

happens? Can you rewrite these three procedures using LAST and BUTLAST?

How about making a square in the middle of the screen? Have the procedure repeat printing a phrase until it completes a square. Can you make a circle?

The Amazing Oracle

Here's a fun game to play on a group of friends or on a class. The Oracle thinks up a story. The object of the game is to figure out the story by asking the Oracle direct questions that can be answered by Yes or No.

Well, the Oracle is sneaky. Take a look at the procedures and see if you can figure out how it works. Then take another look. Because the Oracle gives you some ideas on how to use LAST and BUTLAST to handle lists.

```
TO ORACLE
  CLEARTEXT TEXTSCREEN
  PR [I'M THINKING OF A STORY.~
    ASK ME DIRECT QUESTIONS,]
  PR [THOSE I CAN ANSWER WITH YES OR NO.]
  PR [THEN I'LL TELL YOU WHAT IT IS.]
  QUIZ
  END

TO QUIZ
  PR "
  PR [WHAT'S YOUR QUESTION?]
  MAKE "QUESTION READLIST
  IF NOT (LAST LAST :QUESTION) = "? [PRINT ~
    [QUESTIONS MUST END WITH A QUESTION ~
    MARK.] QUIZ]
  IF MEMBERP ~
```

```
(LAST BUTLAST LAST :QUESTION)
[A E I O U] [PR "YES QUIZ STOP]
IF (LAST BUTLAST LAST :QUESTION) = "Y ~
  [PR "MAYBE QUIZ STOP]
PR "NO
QUIZ
END
```

Oracle begins by asking you for a question.

```
MAKE "QUESTION READLIST
```

When you type a question, you are actually typing a list. This list becomes the variable :QUESTION. Logo then checks to see if you added a question mark.

```
IF NOT (LAST LAST :QUESTION) = "?" [PRINT
  [Questions must end with a question mark.] QUIZ]
```

What this says is that if the LAST character of the LAST word in the list is not a question mark, print the statement and return to the top of the QUIZ procedure.

Now take a look at this one.

```
IF MEMBERP (LAST BUTLAST LAST :QUESTION)
[A E I O U] [PR "YES QUIZ]
```

That really does makes sense because what this statement says is if the next to the last — last but last — character of the last word of the sentence is a member of the list [a e i o u], print Yes and then call the QUIZ procedure again.

If the next to last character is not a member of the list, Logo reads at the next line.

Talk To Your Computer

```
IF (LAST BUTLAST LAST :QUESTION) = "Y [PR  
"MAYBE QUIZ]
```

This time, Logo asks if the next to last character is 'y. If so, Logo prints Maybe and calls the QUIZ procedure again.

By the way, what's LAST :QUESTION? This question gives you a clue. If the next to last character is neither a vowel or 'y, Logo prints 'No and calls QUIZ again.

Now do you know how the Oracle works? It simply answers your questions based on the last letter in the last word. You're actually the one who makes up the story. And some of them can get pretty crazy.

Now that you've had a healthy taste of list processing, take a look at the *Data Structure Commands* in the MSW Logo On-line Help file. This will give you an overview of the other list processing commands. You'll be using more of them in the next exercise.



Time-out for a few experiments first. They're really a review, but you know how Morf hates tests.

Try this one:

```
SHOW LAST "TEST  
(You can use SHOW or PRINT)
```

What do you think is going to be shown (or printed)?

```
SHOW FIRST "TEST
```

What's this going to show?

How about these?

SHOW BUTFIRST "TEST

SHOW BF [THIS IS A TEST.]

SHOW LAST BF [THIS IS A TEST.]

SHOW LAST FIRST BF [THIS IS A TEST.]

One HUGE Gold Star if you said S for that last one.

Try a few of your own. Using the commands of FIRST, LAST, BUTFIRST, and BUTLAST, you can pick any letter of any word in a list.

Word Sums

“What’s two plus two?”

“How easy can you get? It’s four, of course!”

“How’d you get the answer?”

“I added two plus two and got four. What do you think?”

“Did you add $2 + 2$? Or two plus two? If you’re talking about Logo, there is a difference you know.”

“Well, I never really thought about that.”

“OK, let’s give it a whirl.”

Adding words together is a great exercise in list processing. Here’s a brief description of how it works.

WORDSUM "THREE "SIX

Talk To Your Computer

WORDSUM counts through a list to see where the words THREE and SIX are located. They are in the third and the sixth position. So, Logo then adds

$$3 + 6 = 9$$

Finally, Logo counts to the ninth position in the list and prints that word as the answer: NINE.

Logo does one more thing with this procedure; that's to determine if there if the number is a 'teen. In the problem above, it isn't, so Logo simply printed the answer. But let's take a look at another example:

```
WORDSUM "EIGHT "SIX
```

We know the answer is 14. To print it, Logo went to the TEENS procedure and output

The answer is fourteen.

Take a look at the full procedure; WORDSUM.LGO in \procs\chpt12. Look it over and then let's take it apart.

The first question, as you start to look at WORDSUM, is what do :NUMS1 and :NUMS2 equal.

Turn on TRACE and then run the procedure again. This allows you to follow the recursive calls of SET1 and SET2. Then you go to ADDNUMS.

Another way is to just explore the procedure on paper. Start with what you know.

```
WORDSUM "EIGHT "SIX
```

```
:NUM1 = EIGHT  
:NUM2 = SIX  
:NUMS = [ZERO ONE TWO THREE FOUR  
        FIVE SIX SEVEN EIGHT NINE]
```

From the WORDSUM procedure:

```
MAKE "NUMS1 SET1 :NUM1 :NUMS  
MAKE "NUMS2 SET2 :NUM2 :NUMS
```

If you have trouble stepping through these first two lines, just add the following before ADDNUMS in WORDSUM.

```
MAKE "NUMS1 SET1 :NUM1 :NUMS  
MAKE "NUMS2 SET2 :NUM2 :NUMS  
PR :NUMS1  
PR :NUMS2  
IGNORE RC
```

This will print the variables :NUMS1 and :NUMS2 and stop until you press a key to continue. This gives you a chance to check on what's going on.

```
:NUMS1 equals EIGHT NINE  
:NUMS2 equals ZERO ONE TWO THREE ~  
        FOUR FIVESIX
```

Make sure you understand how those variables were selected before proceeding.

```
TO ADDNUMS  
MAKE "NUMS1 SE :NUMS1 :NUMS  
MAKE "NUMS3 (FIRST :NUMS)
```

Talk To Your Computer

```
REPEAT (COUNT :NUMS2) - 1 [MAKE "NUMS1 BF
:NUMS1 IF (FIRST :NUMS1) = (FIRST :NUMS)
[MAKE "NUMS3 FIRST BF :NUMS]] IF :NUMS3 =
"ZERO [TYPE [THE ANSWER IS] PR (FIRST
:NUMS1)] [TEENS]
PR [PRESS ANY KEY TO CONTINUE.]
END
```

Knowing :NUMS1 and :NUMS2, you can write down what the first two lines of ADDNUMS will equal. Again, if you can't figure it out, add the print lines again:

```
MAKE "NUMS1 SE :NUMS1 :NUMS
MAKE "NUMS3 (FIRST :NUMS)
PR :NUMS1
PR :NUMS3
IGNORE RC
```

The third line is where things get a little complicated. But you know all those commands. So just start from the right and move through the line, one command at a time.

```
REPEAT (COUNT :NUMS2) - 1
```

Let's say you typed WORDSUM "EIGHT "SIX. What would :NUMS2 be?

```
[ZERO ONE TWO THREE FOUR FIVE SIX]
```

So, COUNTS :NUMS2 - 1 is what? I get six, what do you get? So the line starts with REPEAT 6. Now what?

The next command is

```
MAKE "NUMS1 BF :NUMS1
```


First of all, what's :NUMS1?

EIGHT NINE ZERO ONE TWO THREE FOUR FIVE
SIX SEVEN EIGHT NINE

What is BF :NUMS1? Remember, BF is BUTFIRST. So
BF :NUMS1 is the list above without the first element, or

NINE ZERO ONE TWO THREE FOUR FIVE
SIX SEVEN EIGHT NINE

Next, you come to:

IF (FIRST :NUMS1) — that's now NINE, right?

IF (FIRST :NUMS1) = (FIRST :NUMS)
[MAKE "NUMS3 FIRST BF :NUMS]

:NUMS is defined in the SETUP procedure. What's
FIRST :NUMS? Does it equal FIRST :NUMS1? No. So the
procedure repeats again.

On the sixth repeat, what is FIRST :NUMS1?

Four, right?

What has :NUMS3 become? It became ONE on the third
repeat cycle. Since :NUMS3 does not equal ZERO, the
TEENS procedure is called.

Since FIRST :NUMS is four, the answer is FOURTEEN.

Awfully simple? Or simply awful?

“Wow! What can’t you do in Logo?”

“Well, Morf, the most important lesson is “Never say never.” You and I aren’t experts, but have you ever found anything you can’t do if you put your mind to it?”

There is a WDTEEN.LGO procedure on the diskette that comes with this book that gives you another look at how this same type of addition can be done. You may like it better than the one used here.

But, after all, that’s what this book is all about. You can almost always find another way to do something, and it just may be a lot easier.

Another thing you’ve got to realize is that we have just scratched the surface of what you can do with Logo. There is so much, much more you can do.”

“OK, we’re wasting time! What’s next?”

Logo and Artificial Intelligence

The whole idea of artificial intelligence gets very confusing. It makes you wonder, just what is intelligence? And how can it be artificial?

This is not the place to discuss whether computers can or ever will be able to really learn. Leave that to the computer scientists and philosophers.

For our purposes, computers of today don’t really learn. It is the software that computers run that make them appear to learn. So, if you look at it, the learning is really kind of

artificial. Maybe we can agree that this is a type of artificial intelligence.

There are some examples of procedures that "learn" on the CD that came with this book. There's the well-known Animal guessing game in `\procs\chpt12` — `ANIMAL.LGO`. This is a procedure where you teach Logo to recognize animals by the descriptions you type.

Have you ever played the game States and Capitals? Someone names a state. You have to name the capital of that state. This edition of the game shows how the computer can appear to be learning (`STATES.LGO` in `\procs\chpt12`).

By now you should have little if any trouble figuring out how this procedure works. Sure, it might take some time. But you can do it.

The main feature of this game is "lists within lists within lists." First, there is the list of States and Capitals — `SLIST`. If you ever want to erase this list and start over, type `INIT`.

Secondly, there is a list that matches each state with its capital — `GROUP`.

Thirdly, there is a list of each state generated from the variable `:QUEST` and one for each capital that comes from `:ANSWER`.

Together, these look like this:

```
MAKE "SLIST [[[Oklahoma] [Oklahoma City]]
              [[New York] [Albany]]
              [[Texas][Austin]]
              [[Massachusetts][Boston]]
```

```
[[California][Sacramento]]]
```

Logo "learns" new states and capitals from the TEACH procedure.

The first thing that TEACH does is ask you to create the variables, :QUEST and :ANSWER. It then creates a new empty list named GROUP.

```
MAKE "GROUP []
```

Next, it adds the state (:QUEST) to the :GROUP list.

```
MAKE "GROUP LPUT :QUEST :GROUP
```

LPUT and FPUT are interesting commands. They are used to add words or other lists to a list. For example:

```
LPUT "Logo [MSW]  
results in the list [MSW Logo].
```

```
FPUT "MSW [Logo]  
also results in the list [MSW Logo].
```

In the case of States and Capitals, LPUT tells Logo to add :QUEST at the end of the list :GROUP. Once you have the state listed, you need to add the capital.

```
MAKE "GROUP LPUT :ANSWER :GROUP
```

This line adds :ANSWER as the second list within the list :GROUP.

```
MAKE "SLIST LPUT :GROUP :SLIST
```

And finally, this line adds the list of two lists to the master list :SLIST.



Time to experiment.

Change the TEACH procedure to add a third element to the GROUP list , maybe the county of the capital or the population of the state?

How would you change the other procedures to ask about that third element?

Go ahead — try it. It's really not that hard.

Rather than look for the LAST element of the GROUP list, you might want to look for the LAST BUTLAST element, or the FIRST BUTFIRST element, or select an ITEM from a list that matches something else.

What's Next

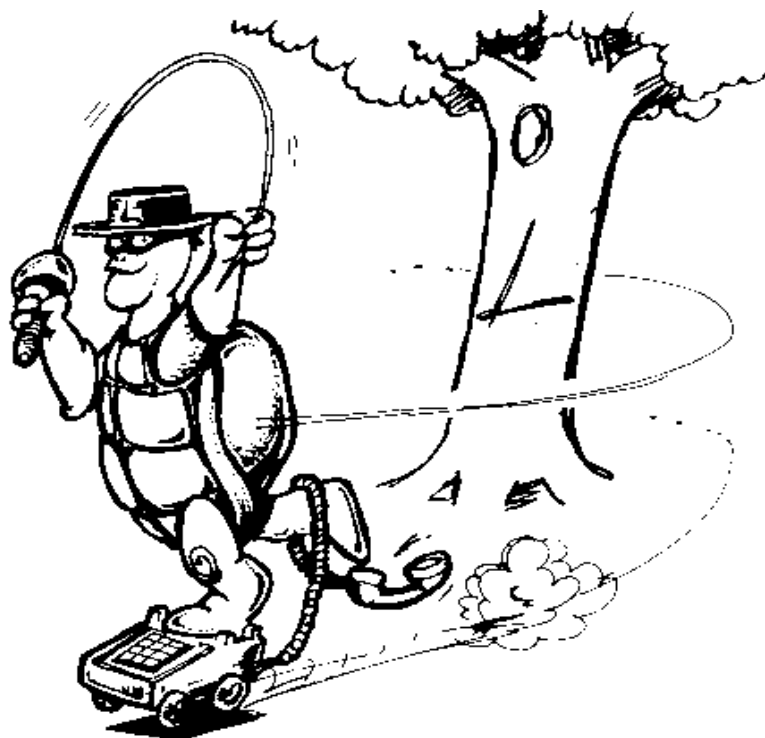
"Seems we just got started and here we are. And there are so many other things to do."

Well, there has to be something left for you to explore on your own. You can start with the projects on the CD that came with this book. This is what Logy and Morf like best, exploring new ways to do things, finding new and better ways to make things work.

By now you can do just about anything you want with MSW Logo. What you don't know, you can find—in the on-

Talk To Your Computer

line help files or in the Example procedures that came with MSW Logo.



Want to explore some more? You'll find a number of interesting Logo sights on the Internet. There's Logy and Morf's Home Page at

<http://www.cyberramp.net/~jmul>

Other addresses include:

<http://www.softronix.com>

for information on MSW Logo and other products from George Mills.

For more information about UCB Logo, use the Logo news group or discussion group as described below. Brian Harvey is a regular participant in these forums.

There is a Logo news group at comp.lang.logo. There's also an on-line Logo mailing list, actually a discussion group. To subscribe, send

subscribe logo-l to majordomo@gsn.org

One of your best sources of information about Logo and Logo products is the non-profit organization:

The Logo Foundation
250 West 85th Street
New York, New York 10024
(212)579-8028
E-mail: michaelt@media.mit.edu
<http://el.www.media.mit.edu/groups/logo-foundation>

Most important! What ever you do, enjoy your very own

GREAT LOGO ADVENTURE!

Talk To Your Computer