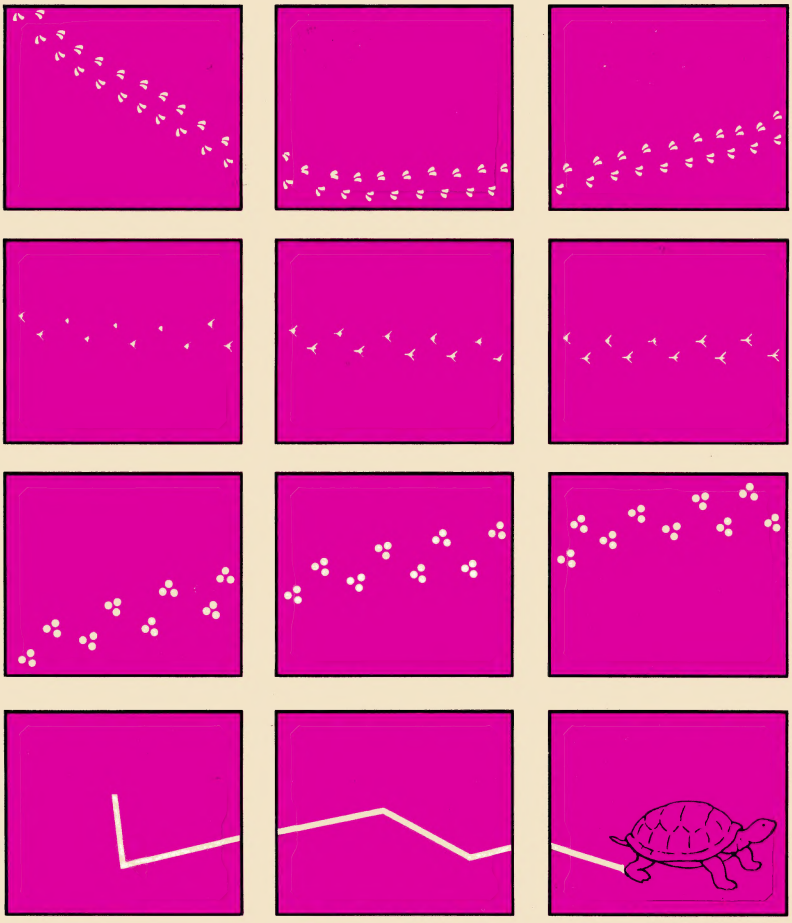
Apple II

Apple Logo

Introduction to Programming through Turtle Graphics



A Product of Logo Computer Systems, Inc.

Disclaimer of all Warranties and Liabilities
Logo Computer Systems, Inc., makes no warranties, expressed or implied, concerning the quality, performance, merchantability or fitness of use for any particular purpose of this manual and software described in this manual. This manual and the software described in this manual are being sold "as is". The entire risk as to the quality and performance of these goods is with the buyer; if the goods shall prove defective following their purchase, the buyer and not the manufacturer, distributor or retailer assumes the entire cost of all necessary servicing, repair and replacement and any incidental or consequential damages. In no event will Logo Computer Systems, Inc. be responsible for direct, indirect, incidental or consequential damages relating to the purchase or use of these goods, even if Logo Computer Systems, Inc. has been advised of the possibility of such damages. Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you.

Notice
Logo Computer Systems, Inc. reserves the right to make any improvements and changes in the product described in this manual at any time and without notice.

Apple Logo

Introduction to Programming through Turtle Graphics

Author: Cynthia J. Solomon

Editorial Board: Margaret Minsky, Seymour Papert, and the staff of LCSI.

Manuscript Editor: Annette Dula

Designers: Judith Richland and Nancy Gardner

Plotter Images: Peter Cann

Photographs: Ralph Mercer

Compositor: County Photo Compositing Corporation

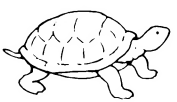
Printer: RBT Printing Ltd.

This Guide is set in Century Textbook and OCR-B on a Quadex/ Compugraphic 8600 typsetting system.

Logo Computer Systems, Inc.
9960 Cote de Liesse
Lachine, Quebec
Canada H8T 1A1

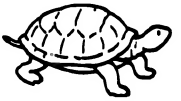
Printed in Canada

Table of Contents



Chapter 1

Logo on the Apple II: An Introduction



Chapter 1 Logo on the Apple II: An Introduction

Logo is a language for computers. Compared with natural human languages like English or French, it has a very small number of words and grammatical rules. But Logo can be extended easily. In fact this is what programming in Logo is all about: using what exists to make new things, and then using the new things to make more new things.

The initial vocabulary words, which we refer to as Logo *primitives,* deal with different kinds of computation including familiar ones such as adding and subtracting numbers and less familiar ones such as manipulating words and lists of words.

A different kind of computation is chosen as the focus of this guide. Technically, the area is known as computer graphics. It has become familiar through computer art, special effects, and video games. As an introductory route into programming, computer graphics has the advantage that you can *see* how your programs work. Our experience is that this is an excellent way to develop an intuitive understanding of what all programming is about.

Using This Guide

This Guide is an introduction to Logo and to programming. It is intended to highlight some activities that you can do; it is not intended to serve as a complete user's guide. It tries to present you with what you need to know to get started. It shows you how to program, to edit, and to save and retrieve your work. For more details on particular aspects and other uses of Logo read the appropriate sections of the *Reference Manual.* The index in the *Reference Manual* will help you locate the section you want to read.

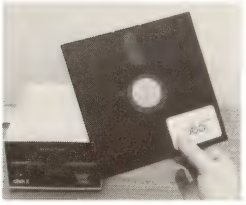
In this Guide we talk about the Logo environment as if it were a living thing.

Periodically, you will find a section in this Guide marked "Bug Box". A "bug" is a friendly name for things that don't quite work. The Bug Box tries to anticipate what might go wrong and suggests ways to fix things.

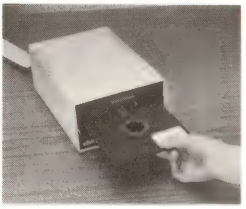
The best way to get to know Logo is to try things on your own. So relax, explore, and have fun!

Apple II System

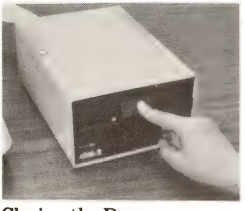
Disk Drive



Diskette



Putting in Diskette



Closing the Door

Setting Up

What you need to do now is to start up Logo and begin to program. To do this you have to learn to identify certain parts of the Apple system. If you are already familiar with the Apple computer then this section will be very simple.

Your Apple II computer system includes a disk drive and a TV monitor. As part of your Logo kit there is also a Logo diskette. You have to put the diskette into the disk drive before you can use Logo. To put in the diskette, open the door of the disk drive by lifting it outwards; a hinge is on top. You should see a slot for the diskette.

Hold the diskette so that the label is face up and slide it into the slot of the disk drive, then *close the door.* The disk drive is now ready to load information from the diskette into the Apple.

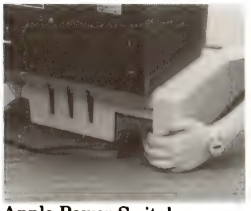
Starting Logo

To start Logo, turn on the Apple power switch, which is located on the back side of the computer. You can reach it from your left. If the power is already on, turn it off and then on again.

Turning on the Monitor

The disk drive light will go on. After a moment, the light will go off and if all is going well you should soon see a message on the screen saying

```
PRESS THE RETURN KEY TO BEGIN

IF YOU HAVE YOUR OWN FILE DISKETTE,
INSERT IT NOW, THEN PRESS RETURN
```



Apple Power Switch

You do not need a file diskette to use this Guide or to use Logo. If you do not have a file diskette, just press the RETURN key and go on.

Bug Box

A file diskette is one on which you can save your programs.

If you do not have a file diskette and want to continue without one, then just press the RETURN key.

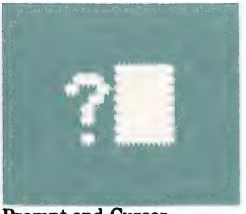
If you do not have a file diskette and want one now, look in the *Reference Manual* to find out how to make one.

After you press the RETURN key, the disk light will go on again. After a few seconds, it will go off and you should now see a message on the screen saying

```
WELCOME TO LOGO
?
```

Below the message you will see a ? (question mark) followed by a flashing ▮ (rectangle).

The ? (question mark) is the *prompt* symbol. When ? (question mark) is on the screen Logo is waiting for you to type something.



Prompt and Cursor

The flashing ▮ (rectangle) is the *cursor*. It appears when Logo wants you to type something and shows where the next character you type will appear.

Bug Box

If you have problems starting Logo, try the following:

1. Reinsert the Logo diskette into the disk drive.
2. Make sure the diskette is properly inserted. The label should face up, at the front of the drive (near your hand).
3. Turn off the Apple, then turn it on again.
4. Make sure any diskette you insert into the disk drive has its label facing up and at the front of the drive.

If you still have problems then perhaps your Apple does not have a 16K RAM or language card in slot 0 or the disk drive is not connected properly.

The Keyboard

The keyboard is like a typewriter. Type any word or sentence to get a feel for the touch of the keyboard. Logo probably won't understand what you say, but it doesn't matter.

For example, type

```
HELLO THERE
```

Then press the RETURN key. Logo will respond with a complaint like


```
I DON'T KNOW HOW TO HELLO
```

Logo is correct, but that's not important at the moment. Just play with the keys. There is no bad thing that you can do because you can always start Logo up again by turning the computer off and then on again.

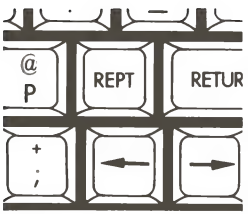
Details About the Keyboard

There are several different kinds of keys; you may be familiar with some.

Character Keys

Character keys — A, B, C, 7, ;, $, etc. — are like those on a typewriter. They include letters of the alphabet, numbers, and punctuation marks. You are probably most familiar with these keys.

The ← Key



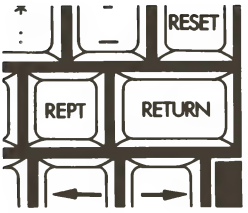
← Key

The ← key is extremely useful. Very few typewriters have such keys. This key causes Logo to erase the character before the cursor. Usually, this is the character you just typed. This key is often called the Delete key. Your Logo kit should have a

DELETE sticker for you to put on the front of this key to remind you of this. If not you might want to make one using adhesive tape.

The RETURN Key

The RETURN key is found on many electric typewriters (where it means "carriage return" and is used to move the type position to a new line on the paper). In Logo the RETURN key serves a function unknown on typewriters. It tells Logo: "Now do what I just typed." You use this key every time you want Logo to obey the instructions you give it.



RETURN Key

SHIFT Key

The SHIFT Key

The SHIFT key can change character keys. Pressed alone, nothing happens. But holding down this key changes the meaning of some of the character keys. For example, while holding the SHIFT key down press N. Logo prints a [on the screen.

We represent this two-key combination as SHIFT-N. SHIFT-M is a]. SHIFT-4 is $ and SHIFT-; is +. Some of these are marked on all Apple keyboards on the upper portion of the key pad.

Check to see whether SHIFT-M and SHIFT-N are marked as] and [on the fronts of the M and N keys. If not look in your Logo kit for stickers to mark them. If you can't find stickers, use adhesive tape. These *bracket* symbols are very important in Logo. Do not confuse them with parentheses, ( ), which are SHIFT-8 and SHIFT-9.

For the SHIFT key to have an effect on another key, always press the SHIFT key first and then *keep* it down *while* typing the other key.

The Space Bar

The space bar is a special character key. It prints an invisible (but very important) character called space. Logo uses spaces as word separators. For example, Logo would interpret THISISAWORD as a single word and would interpret THIS IS A WORD as four words.

Space Bar

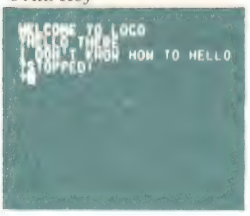
The Control Key

The CTRL key can change character keys into action keys. You use it like the SHIFT key. Press it alone and nothing happens; hold it down and press a certain character key, and something happens. These key combinations do not print out on the screen, but Logo responds to them.

CTRL Key

For example, while holding the CTRL key down press G. We represent this two-key combination as CTRL-G. When we type these two keys together Logo prints

```
STOPPED!
?
```



The CTRL-G key combination serves a very important function; it signals Logo to stop whatever it is doing.

There are several other control key combinations which will be introduced in later sections.

Chapter 2

The Print Command

A good way to start is to jump right in and swim! Type the following:

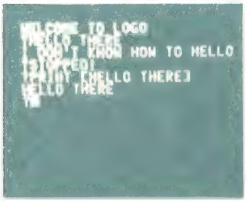
```
PRINT [HELLO THERE]
```

To type [press the SHIFT key and at the same time type N.

To type] press the SHIFT key and at the same time type M.

The instruction appears on the screen, but is not obeyed until you press the RETURN key. Logo will then respond

```
HELLO THERE
```

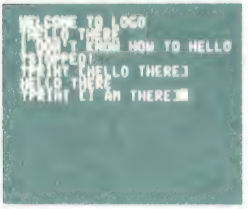


Suppose you wanted Logo to print another message like

```
I AM THE GREATEST
```

but you made a mistake and typed

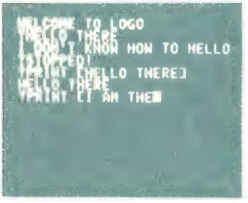
```
PRINT [I AM THERE]
```



Okay, hold on, do not press the RETURN key.

Press the ← key for each character you want to erase until the screen shows

```
PRINT [I AM THE
```



Now type the rest of the line.

```
PRINT [I AM THE GREATEST]
```

Press RETURN and your instruction is obeyed.

```
I AM THE GREATEST
```

The ← key is one of several editing actions Logo provides so that you can change what you have typed without having to type the entire instruction over again. You will be introduced to more editing keys in later sections of this Guide.

You can have Logo print some other sentences by typing PRINT and enveloping what you want typed in brackets as illustrated in the previous example.

Clearing off the Screen

A useful command is CLEARTEXT. It clears the screen of text and starts the text at the top of the screen.

```
CLEARTEXT
```

Press the RETURN key.

The screen clears and the ▮ (cursor) appears at the top of the text screen.

Writing Programs

You can make up *new* commands in Logo by defining procedures. You can use the command TO to signal your intention to Logo. On the same line you tell Logo what name the procedure will go by. Then, you tell Logo what you want the procedure to do. For example, let's define a procedure, GREET, so that whenever you type GREET Logo will type

```
HI THERE
BY NOW
```

Type TO followed by the name of the procedure to start defining the procedure.

```
TO GREET
```

Type a space to separate the words TO and GREET.

Press the RETURN key.

Logo now uses > instead of ? as the prompt symbol. This is to remind you that you are defining a procedure and not entering instructions to be carried out right away.

```
PRINT [HI THERE]
PRINT [BY NOW]
END
```

Press the RETURN key after each instruction.

The word END signals TO that you have finished the procedure definition. Logo will now type

```
GREET DEFINED
?
```

Logo again uses ? as the prompt symbol. Run the procedure by typing

```
GREET
```

Press the RETURN key.

Logo types

```
HI THERE
BY NOW
```

Bug Box

Suppose GREET does not work. Perhaps you typed something strange. Soon you will learn how to "edit" your procedure, that is, change the parts you don't like. In the meantime, just try writing another procedure. Give it another name; let's say GREET1. (Note that numbers can be used as characters in Logo commands.)

```
TO GREET1
PRINT [HI THERE]
PRINT [BY NOW]
END
```

Type a space between TO and GREET1.

Press the RETURN key.

Run this procedure by typing

```
GREET1
```

Logo should type

```
HI THERE
BY NOW
```

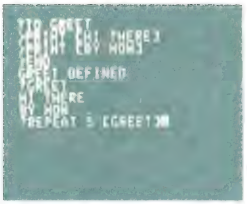
You can use the command REPEAT to run GREET over and over. For example you could have REPEAT run GREET five times.

```
REPEAT 5 [GREET]
```

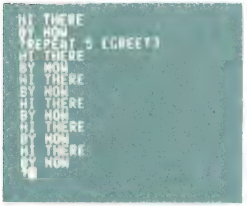
Type a space between REPEAT and 5.

Press the RETURN key.

If you have a GREET1 procedure, you can use it instead of GREET.



Logo responds




```
HI THERE
BY NOW
HI THERE
BY NOW
HI THERE
BY NOW
HI THERE
BY NOW
HI THERE
BY NOW
```

You could have REPEAT run GREET a thousand times.

```
REPEAT 1000 [GREET]
```

If you want to stop this process type CTRL-G.

Bug Box

If CTRL-G doesn't stop Logo from printing on the screen, make sure you are typing the key combination correctly. While holding down the CTRL key press G. Otherwise you can wait until REPEAT is finished.

A Note

In the next chapters of this Guide we will not always remind you to press the RETURN key or to type a space between words.

Logo Vocabulary

Here is a list of Logo words you have used.

PRINT
CLEARTEXT
TO
END
REPEAT

We have also used several special keys.

RETURN
←
CTRL-G
[
]
space bar

Chapter 3

Chapter 3 Meet the Turtle

This section introduces you to programming by learning to control a computer creature known as a turtle. The first Logo turtle was a robot; it looked like a big cannister and moved about on wheels. It was attached to a computer by a long cable and could draw lines on the floor, which normally was covered with paper.

Our turtle, however, lives on the computer screen. It also has a pen and can write with it on the screen. Logo has many commands which you can use to control the turtle. This section introduces you to some of the most important turtle commands.

To see the turtle, give the command SHOWTURTLE.

```
SHOWTURTLE
```

Press the RETURN key.

SHOWTURTLE

The turtle appears on the screen as a triangle. Notice that the turtle is shaped in a way that shows its position and its heading. The *position* and *heading* are called the turtle's *state*. At any time, the graphics turtle is at a specific position and is facing in a specific heading. The most important turtle commands are those that change its state.

The Turtle

At the start, the turtle is in the center of the screen heading straight up.

Notice also that the ? (the prompt symbol) and the ▮ (the cursor) are now near the bottom of the screen. The commands you type will now appear on the last four lines of the screen.

State Change Commands

FORWARD

Now let's get the turtle to do something using the command FORWARD. FORWARD needs an input. The input is a number indicating how many steps the turtle moves.

Type the following command and remember to press the RETURN key when you want Logo to "do it."

```
FORWARD 50
```

FORWARD 50

We chose 50 as the input, but you could use any number.

The space between FORWARD and 50 is very important. It distinguishes the word FORWARD from the word FORWARD50. Actually, you can type extra spaces between words and Logo will ignore them.

Notice that the turtle changes its position but not its heading.

Bug Box

Often as you communicate in Logo you will make goofs. Many of these will be typing errors. Perhaps the most common typing bug is not spacing between a command and its inputs. For example, FORWARD is a command that expects a number as its input. FORWARD is part of Logo's vocabulary. Thus FORWARD 50 is guaranteed to cause the turtle to move 50 steps. But FORWARD50 is a different word from FORWARD, and one that is probably not yet defined.

The difference between the two instructions is merely a space between words. The difference between FRWARD and FORWARD is merely an O, but to Logo these differences are very significant.

If you type

```
FRWARD
```

Logo responds

```
I DON'T KNOW HOW TO FRWARD
```

Check what you have typed with what you meant to type.

RIGHT

To change the turtle's heading we tell it to turn RIGHT or LEFT a specified number of degrees. You can, of course, tell it to turn any number of degrees.

RIGHT 90

In the following example, we tell the turtle to turn RIGHT 90 degrees.

```
RIGHT 90
```

The turtle turns 90 degrees to the right of where it had been heading previously. Notice the turtle changes its heading, not its position on the screen.

BACK 50

BACK

BACK is similar to FORWARD except that the turtle backs away from its current position; it changes the turtle's position only.

```
BACK 50
```

Again 50 is chosen as a concrete example, but you can choose any number.

LEFT

LEFT is similar to RIGHT except that the turtle turns in the opposite direction.

```
LEFT 45
```

The turtle turns 45 degrees to the left of where it had been heading. Again it changes only its heading, not its position. The effect of the turn is seen more clearly if you now tell the turtle to go FORWARD 25.

LEFT 45

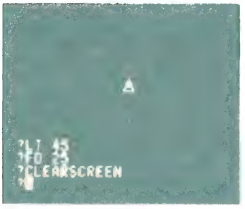
```
FORWARD 25
```

FORWARD 25

CLEARSCREEN

You might want to clear the screen and start again. The command CLEARSCREEN will do that. It will erase previous turtle tracks from the screen and put the turtle at its startup state in the center of the screen heading straight up.

```
CLEARSCREEN
```



CLEARSCREEN

Try experimenting with these state change commands on your own. Remember you can always use CLEARSCREEN to start over.

Logo Vocabulary

Here is a list of commands you have used along with their short names.

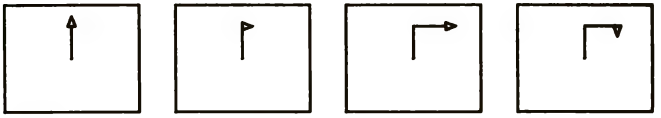
Full Name	Short Name
PRINT	PR
SHOWTURTLE	ST
CLEARSCREEN	CS
FORWARD	FD
BACK	BK
LEFT	LT
RIGHT	RT

Chapter 4

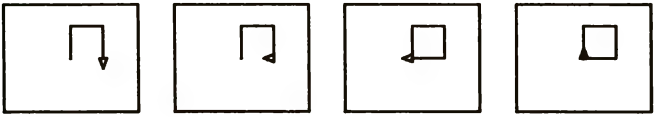
A First Procedure: Teaching the Turtle to Draw a Square

Chapter 4 A First Procedure: Teaching the Turtle to Draw a Square

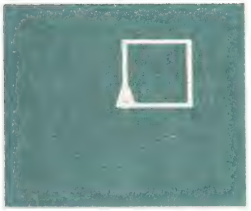
Using the commands FORWARD and RIGHT or LEFT we can make the turtle draw a square. For convenience we will often use the short names for commands.



```
FD 30
RT 90
FD 30
RT 90
```



```
FD 30
RT 90
FD 30
RT 90
```



SQUARE

If we used 50 instead of 30, the turtle would draw a bigger square. We could use any number.

Let's define a new command in Logo to get the turtle to draw a square. New commands in Logo are programs or procedures written by you. Each time you want a square you can use your new procedure rather than retyping the individual instructions.

To define a new command you should first choose a name. Let's use SQUARE because that seems natural. But any name will do. You could then use the Logo command TO and define SQUARE in the same way as you did GREET.

That is, you could type TO SQUARE, then the instructions, then END. This method is good if you

are careful about typing and you know just what you want to type in advance.

Introducing the Logo Editor

There is another way to define a procedure. You could use the Logo editor. Then, if you make typing mistakes you can remove them easily. When you are using the editor, Logo carries out only editing actions.

There is a disadvantage to using the editor. Your turtle drawings will be replaced by the editing screen and the picture will be lost.

On the other hand, the disadvantage of using TO to define your new procedure is that you will not be able to fix typing errors except on the line where the ▮ (cursor) rests.

EDIT

EDIT or ED signals Logo that you want to edit. Follow it by the name of the procedure you want to edit. You must prefix the name with a " (quote mark). Do not type a space between " and the name of the procedure.

```
EDIT "SQUARE
```

Remember the " (quote mark).

After you press the RETURN key you will be using the Logo editor and only editing actions will be carried out.

Bug Box

If you do not remember to prefix SQUARE with a " (quote mark) and type

```
EDIT SQUARE
```

Logo will respond

```
I DON'T KNOW HOW TO SQUARE
```

If you had turtle drawings on the screen they will go away when you start editing.

When the editor starts up, the *title line* of the procedure will appear at the top of the screen.

```
TO SQUARE
```

This is the title line.

TO informs Logo that the following text is part of a procedure definition.

SQUARE is the name of the procedure. You are free to choose another name.

The ▮ (cursor) is at the end of the title line.

Notice that Logo does not print ? or any other prompt symbol while you are using the editor.

Since you do not want to change the title line press the RETURN key.

Now type in the commands that make up SQUARE. They are the commands you used previously.

```
FD 30
RT 90
FD 30
RT 90
FD 30
RT 90
FD 30
RT 90
```

Think of everything you type in the editor as a stream of characters. If you want to add something to the stream, move the ▮ (cursor) to that place.

If you want to move the ▮ (cursor) backwards in the stream, use CTRL-B. You can move all the way to the beginning of the stream.

If you want to move the ▮ (cursor) forward, use CTRL-F.

Then type the characters you want and they will become part of the stream.

Thus, if you have made a typing error in a previous line, use CTRL-B to move the ▮ (cursor) backwards to where the bug is. When the ▮ (cursor) passes over the characters they remain unchanged.

If you want to erase a character the ← key will erase the one to the left of the ▮ (cursor) and will move the ▮ (cursor) to that position.

Notice if the ▮ (cursor) is at the beginning of a text line, pressing the ← key will move it along with the entire line of text to the end of the previous text line. Thus

```
FORWARD 30
RIGHT 90
```

becomes

```
FORWARD 30RIGHT 90
```

Press the RETURN key to separate the lines again.

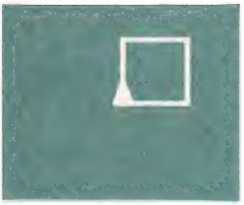
```
FORWARD 30
RIGHT 90
```

If you had pressed space instead of the RETURN key then you would see

```
FORWARD 30  RIGHT 90
```

You can move the ▮ (cursor) from the last character typed to the T in TO on the title line by typing CTRL-B several times.

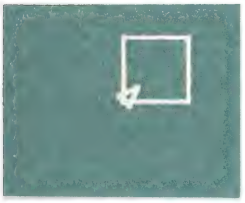
Type CTRL-C when you have completed your editing. If you do not type END, Logo will insert the word when you type CTRL-C.



SQUARE

Logo will now end the procedure definition and carry out the TO command by defining SQUARE. Logo will type

```
SQUARE DEFINED
```



RT 45

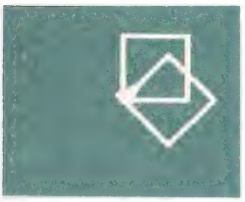
Using the New Command

Try your new command. Type

```
SQUARE
```

Again, type

```
SQUARE
```

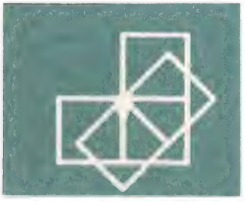


SQUARE

This time the turtle just retraced its path.

If you turn the turtle left or right and then type SQUARE again, a new drawing will appear. For example, tell the turtle

```
RIGHT 45
```



REPEAT 3 [RT 45 SQUARE]

and now type

```
SQUARE
```

Continue to repeat these two commands (RT 45 and SQUARE). To do this you could use the Logo command, REPEAT. REPEAT requires two inputs. For example,

```
REPEAT 3 [RT 45 SQUARE]
```

REPEAT 3 [RT 45 SQUARE]

The first input indicates how many times to repeat the enveloped instructions. The second input is a list of instructions. The instructions must be enclosed in brackets. Think of the brackets as making an envelope. Complete the design by typing

```
REPEAT 3 [RT 45 SQUARE]
```

Now let's make a procedure for this design and call it SQUARESTAR.

```
EDIT "SQUARESTAR
```

You will now be using the editor and the title line will be displayed on the screen.

```
TO SQUARESTAR
```

Notice that the ■ (cursor) is at the end of the title line. Since you do not want to change the title line press the RETURN key and type

```
REPEAT 8 [SQUARE RT 45]
END
```

Don't forget, you type CTRL-C when you are finished editing.

SQUARESTAR

It is always a good idea to try out your new procedure. Put the turtle in its startup state in the center of the screen facing straight up. Remember that CS will do this. Type

```
CS
```

and then

```
SQUARESTAR
```

Bug Box

If your squares look like rectangles the bug is in your TV, and not in Logo. The Logo command SETSCRUNCH allows you to change the aspect ratio of the screen. If you want to know what the current setting is type

```
PR SCRUNCH
```

Logo prints a number which is the current setting. Now try

```
SETSCRUNCH .8
```

Then type

```
CS SQUARE
```

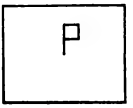
If your square looks worse try

```
SETSCRUNCH 1
```

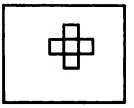
Try other settings until you are satisfied.

Other Uses of SQUARE

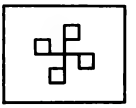
Once you have defined a procedure, you can use it as you would any Logo primitive such as FD, BK, LT, RT, etc. Thus, a procedure you define can be used as part of the definition of other procedures. This is one of the powerful features of Logo. For example, there are many designs which use SQUARE. Some more examples are:



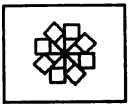
FLAG



CROSS

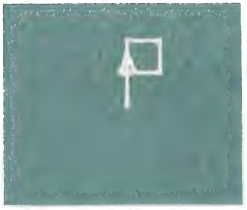


FLAGS



MANYFLAGS

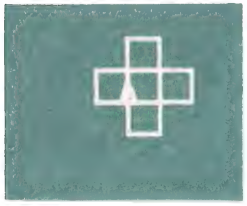
Both FLAG and FLAGBACK make the turtle draw the same design, but they leave the turtle in different states. Both procedures leave the turtle with the same heading as they found it, but FLAG leaves the turtle in a different position from the one it started in.



FLAG

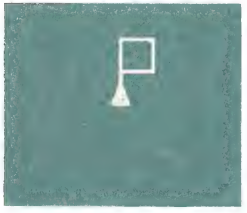
FLAGBACK, on the other hand, leaves the turtle in the same position on the screen as it was found. We can see the effect of these differences in CROSS and FLAGS. CROSS runs FLAG four times while FLAGS runs FLAGBACK four times.

```
TO FLAG
FD 30
SQUARE
END
```



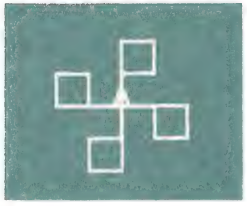
CROSS

```
TO CROSS
REPEAT 4 [FLAG RT 90]
END
```



FLAGBACK

```
TO FLAGBACK
FLAG
BK 30
END
```



FLAGS

```
TO FLAGS
REPEAT 4 [FLAGBACK RT 90]
END
```

MANYFLAGS uses FLAGS.

```
TO MANYFLAGS
FLAGS
RT 45
FLAGS
END
```

MANYFLAGS

Logo Vocabulary

Full Name	Short Name
REPEAT	
SETSCRUNCH	
SCRUNCH	
CLEARSCREEN	CS
SHOWTURTLE	ST
FORWARD	FD
RIGHT	RT
BACK	BK
LEFT	LT
TO	
END	
EDIT	ED
PRINT	PR

Special Keys

CTRL-G

CTRL-B

CTRL-F

RETURN

←

CTRL-C

Chapter 5

Saving and Retrieving Your Work

Chapter 5 Saving and Retrieving Your Work

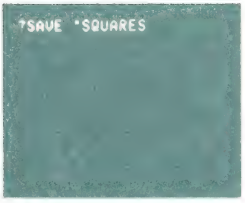
You can save your procedures on a diskette using the command SAVE.

Bug Box

Save your procedures on a file diskette. If you do not have a file diskette, skip this chapter and go on.

To make a file diskette consult the *Reference Manual.*

The diskette is divided into *files*. Each file has a name. To save your work, you must think up a name for the file. Let's use the name SQUARES.



SAVE "SQUARES

```
SAVE "SQUARES
```

SAVE is the command.

The name of the file is a word and prefixed by a " (quote mark). You can choose almost any name.

All of your procedures are now saved in the file SQUARES.

LOAD "SQUARES

Wait until the red light on the disk drive is off. Now that you have saved your work, you can turn off the Apple and remove the diskette. You can later retrieve your work from the diskette.

Loading a File from Diskette

To retrieve your work from the diskette, use the command LOAD, which takes as its input the name of the file you want.

```
LOAD "SQUARES
```

Files on the Diskette

You can see the names of files already on the diskette using the command CATALOG.

```
CATALOG
```

Notice that your file, SQUARES, has been given the name SQUARES.LOGO on the diskette. Each time Logo saves your work it adds .LOGO to the end of the name. But, when you refer to the file by name, you do not use that suffix.

New Files

If you make a new file, pick a name that is not already used as a file name.

Bug Box

In this manual we use LOAD and SAVE with one input, the name of a file. They can, however, take two inputs. We do not discuss that use in this manual, and so to avoid any problems we suggest that you use these commands by themselves with no other command following them on that line.

Logo Vocabulary

SAVE

LOAD

CATALOG

Chapter 6

The Turtle and Text on the Screen

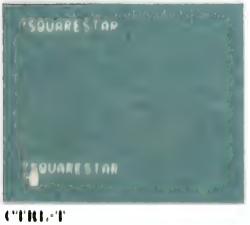
Before you give Logo any turtle commands, the whole screen is available for text. As soon as you give a turtle command, the screen is divided into a large turtle field and a small text field. In fact, only four lines at the bottom are available for text. There are different ways to get the whole screen back for text.

The command TEXTSCREEN will get you the whole screen for text. SPLITSCREEN will get you back to the turtle field and the four-line text field. Neither of these commands destroys what was on the two fields. They only change what is *visible* to you. Try going back and forth. There are special action keys which have the same effect as these commands. For example, CTRL-T is for TEXTSCREEN and CTRL-S is for SPLITSCREEN.

SQUARESTAR


CTRL-L


CTRL-S



CTRL-T

FULLSCREEN (or CTRL-L) gives the whole screen to the turtle. No text is visible. So if you type FULLSCREEN you will not see the characters typed on the screen, but they will be there all the same.

CTRL-T, CTRL-S, and CTRL-L can be typed while a procedure is running.

CTRL-L CTRL-S

Next we go back and forth between TEXTSCREEN and SPLITSCREEN.

CTRL-T CTRL-S

Note: When you edit a procedure, the turtle screen is erased; it becomes the edit screen.

Neither CTRL-S nor CTRL-L will switch to the turtle screen until you give a turtle command.

Logo Vocabulary

TEXTSCREEN

FULLSCREEN

SPLITSCREEN

Special Keys

CTRL-T

CTRL-L

CTRL-S

Chapter 7

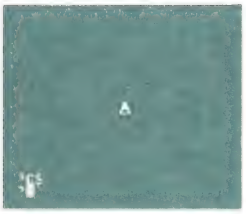
More Turtle Commands

You might want to clear the screen and put the turtle in the center of the screen heading straight up. So, type

```
CS
```

HIDETURTLE

You can make the turtle invisible by typing HT or HIDETURTLE.



CS

SHOWTURTLE

ST or SHOWTURTLE will make the turtle visible again.

HIDETURTLE

Pen Commands

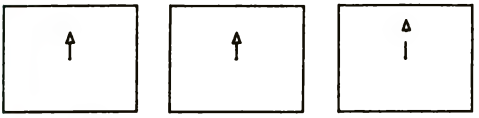
PENUP

PU or PENUP lets the turtle move without drawing any lines.

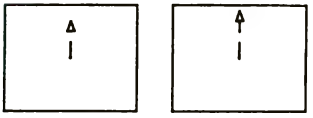
SHOWTURTLE

PENDOWN

PD or PENDOWN lets the turtle use its pen again.



```
FD 30
PU
FD 20
```



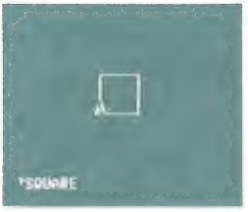
```
PD
FD 20
```

PENERASE

You can erase what the turtle has drawn by using the commands PENERASE or PENREVERSE.

If you give the command PENERASE the turtle becomes an eraser instead of a drawing instrument. If you now make it retrace a line it has drawn, the line will be erased.

For example, clear the screen, make sure the pen is down, and draw a square.



SQUARE

```
CS
PD
SQUARE
```

Now

```
PENERASE
SQUARE
```

PENREVERSE

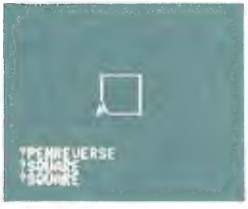
The turtle will erase any lines on the screen until you tell it to PENUP or to PENDOWN. Notice the turtle does not draw any new lines.

PENREVERSE

PENREVERSE is a mixture of PENDOWN and PENERASE. When this command has been given the turtle will draw a line whenever it moves over blank background. But if it moves over a previously drawn line it will erase it. This can be used to produce some spectacular effects. For example, type

SQUARE

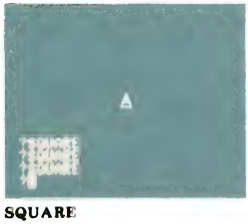
```
PENREVERSE
SQUARE
SQUARE
SQUARE
```



SQUARE

PENDOWN (or PD) returns the pen to its normal writing state. So, type

```
PD
```


SQUARE

Using Apple Color Graphics

This section describes commands and features which take advantage of Apple color. The Apple allows you to use six colors: black, white, green, violet, orange and blue. If you have a black and white monitor, your color selection includes gray tones as well as black and white.

There are two types of color changes you can make. You can change the color of the turtle field or BACKGROUND by using the command SETBG.

You can also change the color in the turtle's pen. SETPC is the command to change PENCOLOR.

Both SETBG and SETPC take one input, which must be a number. The colors are number-coded in the following way.

0 is black.
1 is white.
2 is green.
3 is violet.
4 is orange.
5 is blue.

Changing the BACKGROUND Color

Try

```
SETBG 1
SETBG 2
SETBG 3
SETBG 4
SETBG 5
```

Define a procedure that cycles through background colors.

To see the colors more clearly use the Logo command, WAIT. WAIT 60 makes Logo wait for 1 second before running the next command. CB is an example of such a procedure. Try different inputs to WAIT.

```
TO CB
SETBG 1 WAIT 20
SETBG 2 WAIT 20
SETBG 3 WAIT 20
SETBG 4 WAIT 20
SETBG 5 WAIT 20
SETBG 0 WAIT 20
END
```

Repeat CB a few times. Type

```
REPEAT 3 [CB]
```

You can always find the number code of the current color by printing BACKGROUND or its short name, BG. Type

```
PR BG
```

Logo responds

```
0
```

Note on "Artifacts"

You would think that changing the background color would not do anything to the lines already on the screen. But the way the Apple works, the colors of the lines are changed. These are "artifacts". We can either say "too bad", or look for ways to make use of these artifacts. These show up most often when you use PENERASE or PENREVERSE.

Changing PENCOLOR

Use the command SETPC. Try different pen colors. Start with the background color in black and a clear screen.

```
SETBG 0
CS
SETPC 2 SQUARE
RT 90 SETPC 3 SQUARE
RT 90 SETPC 4 SQUARE
RT 90 SETPC 5 SQUARE
```

Now type

```
SETPC 0 SQUARE
```

The square disappears! This should not surprise you since the pen color and the background color are the same. To set the pen color to white, type

```
SETPC 1
```

Bug Box
If you are using a black and white monitor one way to get thinner pen lines is to do the following:

```
SETBG 6
SETPC 1
```

PENCOLOR or its short form PC will give you the current number code for the pen color. Type

```
PR PC
```

Logo responds

```
1
```

Bug Box
Pen color green will not draw on background orange, and pen color orange will not draw on background green. Violet and blue will not show up on one another.

If you change the color of the background, lines already on the screen might change color.

Logo Vocabulary

Full Name	Short Name
HIDETURTLE	HT
SHOWTURTLE	ST
PENDOWN	PD
PENUP	PU
PENERASE	
PENREVERSE	
SETBG	
BACKGROUND	BG
SETPC	
PENCOLOR	PC
WAIT	

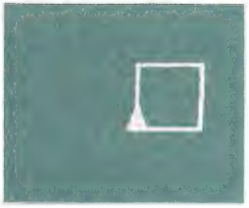
Chapter 8

Another Look at Editing Procedures

The Logo editor allows you to change already defined procedures as well as define new ones. You may wish to change one of your procedures to fix a bug or to alter what the procedure does.

First define a procedure to draw a diamond, but with a bug in it. For example type

```
TO DIAMOND
SQUARE
RT 45
END
```



DIAMOND

This procedure is supposed to draw a diamond. We try it and find it draws a square, not a diamond. The bug is obvious. The command RT 45 should be used *before* the turtle draws a square. To fix the bug we edit the procedure. Type

```
EDIT "DIAMOND
```

The text of the procedure, DIAMOND, is now displayed on the screen.

```
TO DIAMOND
SQUARE
RT 45
END
```

The ▮ (cursor) is positioned at the top left corner of the screen on the letter, T of the word TO.

To edit, move the ▮ (cursor) to where you want to add or delete characters. Move the ▮ (cursor) forward (to the right) by typing CTRL-F. Move it back (to the left) by typing CTRL-B.

So, to edit DIAMOND move the ▮ (cursor) to the end of the title line.

```
TO DIAMOND
```

Now press the RETURN key and type

```
RT 45
```

Move the cursor down to the line before END by typing CTRL-N twice.

```
RT 45
```

Press the ← key (five or six times) to erase the characters on the line.

Leaving the Editor
Type CTRL-C. This signals Logo that you have finished editing. Logo prints out a message saying

```
DIAMOND DEFINED
```

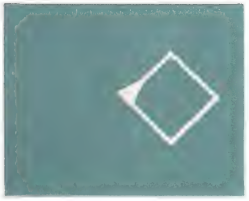
Bug Box
If you are editing and don't like the changes you are making or decide not to make changes and want to start again, type CTRL-G. Logo will "abort" editing and forget the changes that you made so far. The definition of the procedure will be the same as before you started editing.

Here are some useful editing actions. The *Reference Manual* describes more of them.

CTRL-N	moves the cursor down to the next line.
*CTRL-P	moves the cursor up to the previous line.
RETURN	moves the cursor and the following text to the beginning of the next line.
*CTRL-A	moves the cursor to the beginning of the current line.

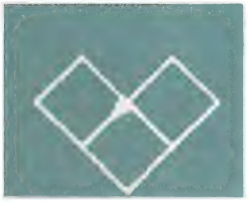
*CTRL-E	moves the cursor to the end of the current line.
CTRL-B	moves the cursor one space backward (to the left).
CTRL-F →	moves the cursor one space forward (to the right).
*CTRL-D	erases the character directly under the cursor.
←	erases the character to the left of the cursor.

*This editing action was not discussed in this section, but it is useful and will be used later.



DIAMOND

RT 45 DIAMOND



RT 45 DIAMOND

Editing Outside of the Editor

You can use most of the control-character actions to edit instructions you type to Logo when you are not in the editor. For example, type

```
DIAMOND
```

Now type CTRL-Y. This gets you a copy of the last line you typed. Each line of text on the screen is like a mini-editor, only one line high. Logo responds

```
DIAMOND
```

The ▮ (cursor) is at the end of the line.

You can now type CTRL-A to move the cursor to the beginning of the line.

```
DIAMOND
```

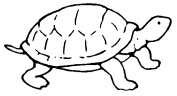
Now type

```
RT 45
```

Type a space.

Now press RETURN. Type CTRL-Y again and then press RETURN.

Experiment with other editing actions both in the editor and outside of it. For more details consult the *Reference Manual.*



Chapter 9

Your Workspace

As you interact with Logo and give words meaning Logo puts these new words in what we refer to as your *workspace*.

If you want to see what you have in your workspace, Logo provides several ways to do so. For example, you can print out the title lines of all the procedures you have written or you can print out their definitions.

Printing Out Procedures

POTS prints out the title lines of each of the procedures in the workspace. Type

```
POTS                    PrintOut Titles
```

POTS

Logo responds

```
TO SQUARE
TO SQUARESTAR
    .
    .
    .
```

and so on.

POPS prints out the definitions of all the procedures in the workspace. Type

```
POPS                    PrintOut Procedures
```

POPS

Logo responds

```
TO SQUARE
FD 30
RT 90
FD 30
RT 90
FD 30
RT 90
FD 30
RT 90
END
```

```
TO SQUARESTAR
REPEAT 8 [SQUARE RT 45]
END
    .
    .
    .
```

and so on.

You can print out the definition of any particular procedure with PO (PrintOut). For example, type

```
PO "SQUARESTAR
```

Logo responds

PO "SQUARESTAR

```
TO SQUARESTAR
REPEAT 8 [SQUARE RT 45]
END
```

PO can also be given a list of procedure names. For example, PO [SQUARE SQUARESTAR DIAMOND] would print out the definitions of the three procedures whose names are in the input list.

Erasing From the Workspace

You can erase procedures from your workspace. If you have not saved these procedures on a diskette, you will have to type them in again. So, make sure you really want to erase.

There are several Logo commands to erase your work. The most commonly used is ERASE, whose short form is ER.

```
ERASE "DIAMOND
```

erases the procedure DIAMOND.

```
ERASE [SQUARE SQUARESTAR]
```

erases all the procedures named in the input list.

```
ERPS
```

erases all procedures from the workspace

Bug Box

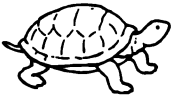
Some of these workspace commands can take an additional input. We do not discuss that use in this manual, but to avoid any problems we suggest that you use these commands by themselves with no other command following them on that line.

Logo Vocabulary

Full Name	Short Name
ERASE	ER
ERPS	
PO	
POTS	
POPS	

Chapter 10

A First Project: Drawing a Spider



Let's make a spider like this one; it has 4 legs on each side. A first step is to look more closely at a right leg and a left leg. Each leg is made by 2 lines joined to form a 90 degree angle.

SPIDER

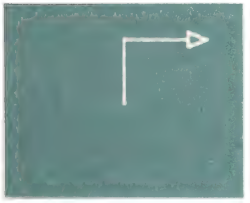
As a first step let's make a RIGHTLEG.

```
TO RIGHTLEG
FD 30
RT 90
FD 30
END
```

You can choose any number as input to FORWARD. We use 30 here.

Now type

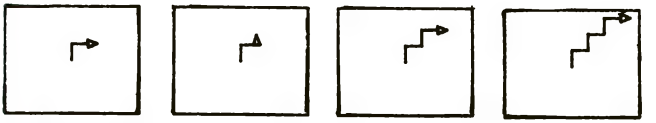
```
RIGHTLEG
```



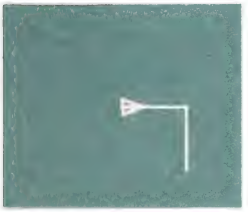
RIGHTLEG

Although this procedure makes a leg, the turtle stops at a funny place for making another spider leg.

At this point RIGHTLEG could be used to make stairs.



```
RIGHTLEG
LT 90
RIGHTLEG
LT 90
RIGHTLEG
```



RIGHTLEG Debugged

... but we want spider legs.

When in doubt about where you think the turtle should be when the procedure stops, put the turtle where it was before the procedure was run. Now fix or *debug* RIGHTLEG using the editor.

```
EDIT "RIGHTLEG
```

Now the Logo editor shows this procedure with the ▮ (cursor) on the T of TO.

```
TO RIGHTLEG
FD 30
RT 90
FD 30
END
```

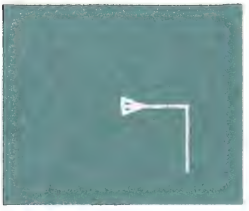
Now type in the new commands.

```
TO RIGHTLEG
FD 30
RT 90
FD 30
BK 30
LT 90
BK 30
END
```

These next 3 commands return the turtle to where it was at the start of RIGHTLEG.

Try RIGHTLEG.

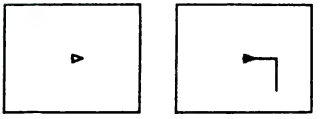
```
RIGHTLEG
```



RIGHTLEG

Now plan RIGHTSIDE, the procedure which will draw all the legs on the spider's right side. We want one leg horizontal, so ...

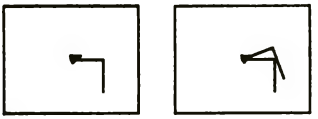
```
CS
```



```
RT 90
```

```
RIGHTLEG
```

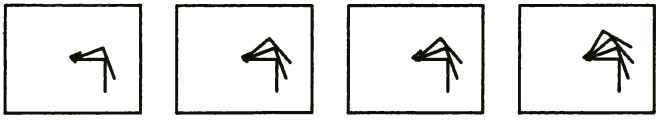
Now for the second leg ...



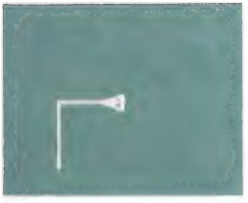
```
LT 20
RIGHTLEG
```

Good. Continue in this way until the turtle has drawn 4 legs. You can now make a procedure for RIGHTSIDE.

```
TO RIGHTSIDE
RT 90
REPEAT 4 [RIGHTLEG LT 20]
LT 10
END
```



Notice the last command in RIGHTSIDE, LT 10, returns the turtle to the same position and heading it was in at the start of the procedure. It is good practice to force procedures to adopt the rule of good behavior: "Leave the turtle in the state you found it."



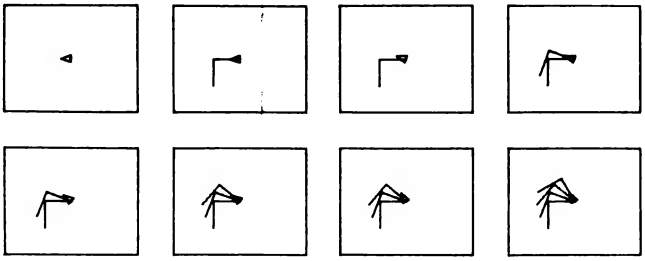
LEFTLEG

Now work on a left leg. LEFTLEG will be similar to RIGHTLEG.

```
TO LEFTLEG
FD 30
LT 90
FD 30
BK 30
RT 90
BK 30
END
```

Try it out. Use LEFTLEG to write LEFTSIDE.

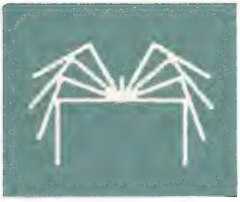
```
TO LEFTSIDE
LEFT 90
REPEAT 4 [LEFTLEG RT 20]
RT 10
END
```



and finally...

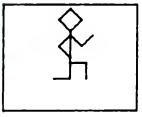
```
TO SPIDER
LEFTSIDE
RIGHTSIDE
FD 10 BK 10
HT
END
```

HIDETURTLE now that the job is done.

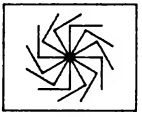


SPIDER

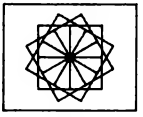
Other Designs Using RIGHTLEG and LEFTLEG.



MAN



SWIRL

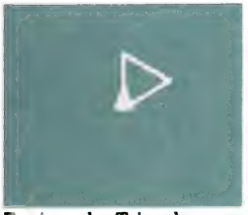


SPINSTAR

Chapter 11

Some Geometry: Triangles

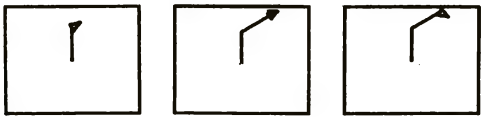
The turtle can draw different triangle shapes. The triangle we discuss below is like a square in that all its sides are equal and all its angles are equal. In this example, the turtle will take 30 steps forward, the same amount it took in SQUARE.



Equiangular Triangle

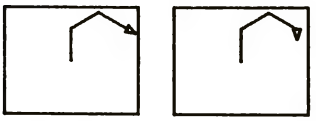
```
FD 30
```

Now comes the big decision. How many degrees does the turtle have to turn to draw this triangle? In school we learned that equiangular triangles have 60 degree angles. Look what happens when the turtle turns 60 degrees.

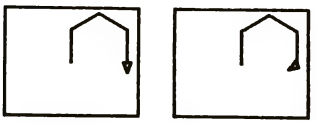


```
RT 60
FD 30
RT 60
```

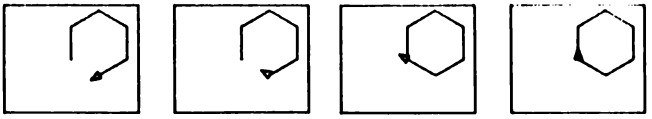
Interesting, but not a triangle! But we might as well finish it off.



```
FD 30
RT 60
```



```
FD 30
RT 60
```

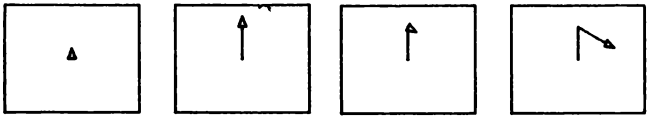


```
FD 30
RT 60
FD 30
RT 60
```

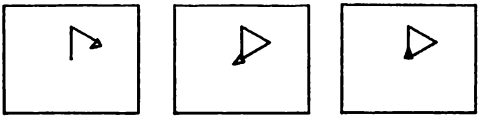
The figure is a hexagon; it has 6 rather than 3 sides. To make a triangle the turtle needs to turn 120 degrees at each corner.

Why 120 and not 60? The answer is simple. When the turtle starts its triangle trip, it must turn 360 degrees (a complete circle) before it returns to its starting state. It walks along an edge of the triangle of the triangle. The turtle does this 3 times. (3 times 120 is 360; 6 times 60 is 360.) When the turtle draws a square it turns four times instead of three; 4 times 90 is 360!

Now we can finish drawing the triangle.



```
CS
FD 30
RT 120
FD 30
```



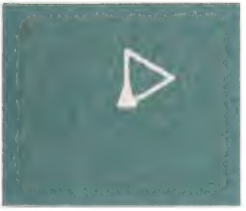
```
RT 120
FD  30
RT 120
```

Now you can define the word TRIANGLE. Let's use the editor to do this.

```
EDIT "TRIANGLE
```

and now type

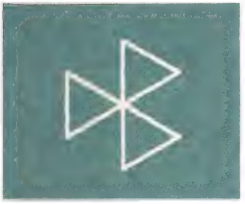
```
REPEAT 3 [FD 30 RT 120]
END
```



TRIANGLE

Play with this procedure a bit. For example

```
REPEAT 3 [TRIANGLE RT 120]
REPEAT 6 [TRIANGLE RT 60]
REPEAT 100 [TRIANGLE RT 30]
```

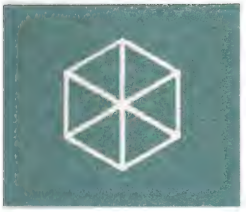


REPEAT 3 [TRIANGLE RT 120]

In this last example the turtle retraces its path many times. You can always stop the turtle by pressing CTRL-G.

You might want to figure out how many times the turtle needs to repeat a set of commands. For example if the turtle turns 30 degrees each round, it has to repeat the set of commands 360/30 or 12 times. Logo can do arithmetic and so can divide 360 by 30 for you.

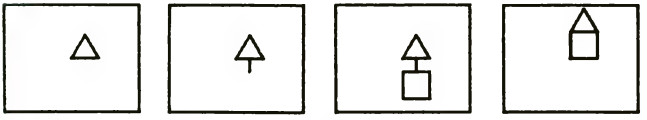
```
REPEAT 360/30 [TRIANGLE RT 30]
```



REPEAT 6 [TRIANGLE RT 60]

Here are some designs made by using TRIANGLE.

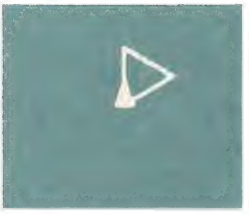
REPEAT 360/30 [TRIANGLE RT 30]



TENT TREE WELL HOUSE

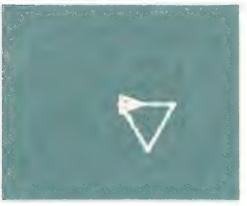
Let's make the turtle draw TENT. Just running TRIANGLE is not enough. The TENT will be tipped.

```
CS
TRIANGLE
```



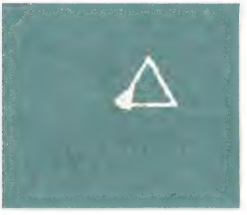
Turn the turtle RT 90 and run TRIANGLE. This time the tent is upside down.

```
CS
RT 90
TRIANGLE
```



Remember when we made TRIANGLE, the inside angle was 60 degrees. If we now turn the turtle RT 90 and then LT 60 the turtle is set up for drawing a tent. Of course, we could just turn the turtle RT 30.

```
CS
RT 30
TRIANGLE
```

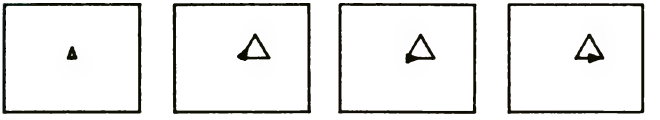


The procedure, then, is:

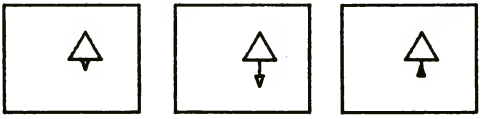
```
TO TENT
RT 30
TRIANGLE
END
```

TENT into TREE

Now you can use TENT to help with TREE.



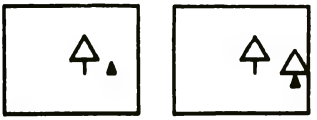
```
CS
TENT
RT 60
FD 15
```



```
RT 90
FD 15
RT 180
```

Now define TREE. You could then make three or four trees appear on the screen. For example,

```
RT 90
PU
FD 30
```



```
LT 90
PD
TREE
```

It is a good programming habit to put the setup instructions in a separate procedure. SETTREE will set up the turtle for drawing a new tree on the screen.

```
TO SETTREE
RT 90
PU FD 30
LT 90 PD
END
```

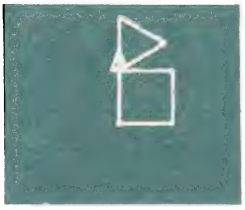
We can then use TREE and SETTREE repeatedly.

```
REPEAT 3 [TREE SETTREE]
```

If you want to change the distance between trees, then edit SETTREE and increase the amount the turtle goes FORWARD.

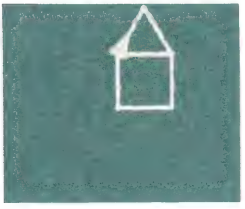
Turtle Makes a House

You have taught the turtle to make a square and a triangle. Now put them together to make a house.

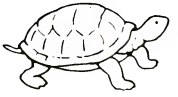


```
SQUARE
FD 3Ø
TRIANGLE
```

You put them together but the turtle didn't draw a house. There is a bug. The fix is simple. We should use TENT instead of TRIANGLE.



```
SQUARE
FD 30
TENT
```



Chapter 12

Variables: Big Squares and Small Squares

Chapter 12 Variables: Big Squares and Small Squares

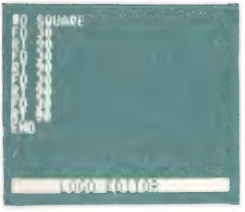
You might want the turtle to draw squares with sides of 60, 50, 100, 10 and so on. One way of doing this is to have many procedures: SQUARE100, SQUARE50, SQUARE33, etc. But there must be a better way, some kind of shortcut; and, of course, there is. We can change SQUARE so that it is like FORWARD in that it takes an input. Then we can tell SQUARE how long to make its sides by typing

```
SQUARE 50
SQUARE 40
SQUARE 30
```

So let's make a procedure for drawing variable sized squares. BOX might be a suitable name since it reminds us of squares. But we'll meet a new idea by calling it BOXR (usually pronounced like box-are) which makes us think of a right-turning square. We could also define a left-turning box and call it BOXL (usually pronounced box-ell).

A shortcut method for typing in the definition of BOXR is to modify SQUARE in the editor. If we change the name of the procedure before we leave the editor we will not change the definition of SQUARE.

```
EDIT "SQUARE
```



Now the Logo editor shows this procedure with the ■ (cursor) on the T of TO.

```
TO SQUARE
FD 30
RT 90
FD 30
RT 90
FD 30
RT 90
FD 30
RT 90
END
```

First let's change the name of the procedure from SQUARE to BOXR. The ▮(cursor) is sitting on the T of TO. Move it to the space after SQUARE. Use CTRL-E.

```
TO SQUARE
```

Now erase the word SQUARE using the ← key and type the word BOXR.

```
TO BOXR
```

Type CTRL-C. Logo responds

```
BOXR DEFINED
```

At this point BOXR and SQUARE have the same definition. Now we change BOXR. Type

```
EDIT "BOXR
```

The Logo editor now shows

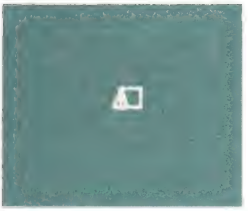
```
TO BOXR
FD 30
RT 90
FD 30
RT 90
FD 30
RT 90
FD 30
RT 90
END
```

We change BOXR so that it requires an input like FORWARD does. The procedure will be able to draw squares of various sizes. How do we tell Logo to do this?

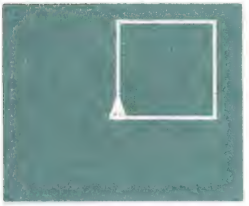
The first instruction has to be FORWARD some amount, but when we are writing the procedure we don't know what amount it will be. We handle this situation by giving the amount a name. For example, let's call it SIDE. If we choose this name,

we write FD :SIDE to mean FD whatever number happens to be called SIDE. So we can write

```
FD :SIDE
RT 90
FD :SIDE
RT 90
FD :SIDE
RT 90
FD :SIDE
RT 90
```



BOXR 20



BOXR 100

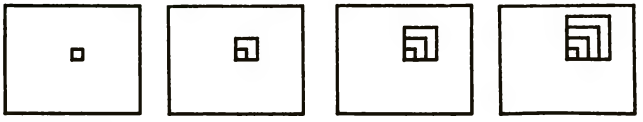
One more idea is needed to turn this into a procedure. When we use the command BOXR we will now have to follow it with an input like this:

```
BOXR 20
BOXR 100
```

To indicate that BOXR needs an input and that this input will be called SIDE we do the whole procedure like this:

```
TO BOXR :SIDE
FD :SIDE
RT 90
FD :SIDE
RT 90
FD :SIDE
RT 90
FD :SIDE
RT 90
END
```

BOXR makes the turtle draw a square of any size depending upon the number you give it as an input.



```
BOXR 10
BOXR 20
BOXR 30
BOXR 40
```

We have just used a powerful mathematical idea — the idea of variable. But instead of using a mysterious X for the variable as we did in school algebra, we have used a meaningful name, SIDE.

When you were defining this procedure you wanted to tell the turtle how to draw a square, but you did not know what size square you might need. Indeed, you wanted to be able to draw squares of all possible sizes. When you came to typing the FORWARD command you knew that FORWARD needed an input. You couldn't just type FORWARD without an input. You had to type FORWARD something. To give this something a name, we called it SIDE. In Logo the expression :SIDE means "whatever happens to be in the container called SIDE". If Logo is to carry out the command FORWARD :SIDE there must be something in the container.

The container is filled when you use BOXR and type BOXR 10 or BOXR 15. When Logo obeys that command, 10, 15, or whatever you typed as the input is put in the container named SIDE. BOXR can then look in the container at a later time.

Bug Box

Possible bugs:

1. You typed :SISE or some other spelling different from the way the input on the title line was spelled.
2. You forgot to use dots (:).
3. You inserted an extra instruction in BOXR.
4. You accidentally erased an instruction in BOXR.
5. You typed a space between : and SIDE.
6. You typed a : in front of a number.

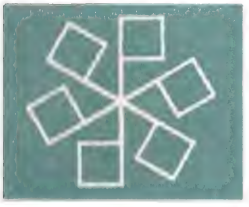
The character, : (dots), informs Logo that the word to which it is prefixed names a container which can have in it a number, another word, a list of words, or a list of lists.

Some Uses of BOXR

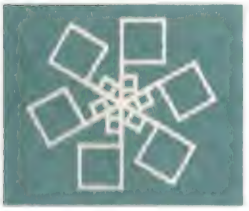
SQUARES

DIAMONDS

FLAGR 30



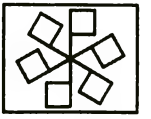
6FLAG 30



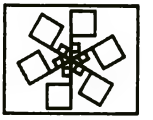
SPINFLAG 30

SQUARES

DIAMONDS



6FLAG 30



SPINFLAG 30

```
TO SQUARES
BOXR 10
BOXR 20
BOXR 30
BOXR 40
END

TO DIAMONDS
RT 45
REPEAT 4 [SQUARES RT 90]
END

TO FLAGR :SIZE
FD :SIZE
BOXR :SIZE
BK :SIZE
END

TO 6FLAG :SIZE
REPEAT 6 [FLAGR :SIZE RT 60]
END

TO SPINFLAG :SIZE
6FLAG :SIZE
6FLAG :SIZE - 20
END
```

Being able to control the size of a shape makes that procedure much more useful and interesting.

Big Triangles and Small Triangles

We can also define a triangle procedure which takes an input. Type

```
ED "TRIANGLE
```

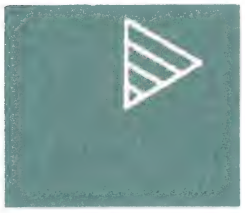
Now the Logo editor shows this procedure with the ▮ (cursor) on the T of TO.


```
TO TRIANGLE
REPEAT 3 [FD 30 RT 120]
END
```

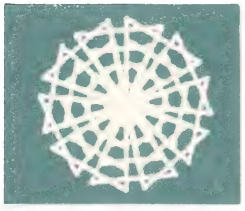
Then change TRIANGLE.


```
TO TRIANGLER :SIDE
REPEAT 3 [FD :SIDE RT 120]
END
```

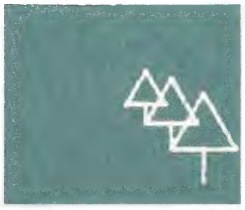
Use the procedure to make designs like:



TRIANGLES

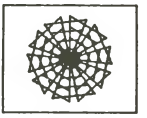


TRISTAR

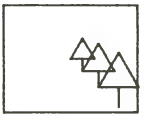


TREES

TRIANGLES



TRISTAR



TREES

```
TO TRIANGLES
TRIANGLER 10
TRIANGLER 20
TRIANGLER 30
TRIANGLER 40
END

TO TRISTAR
REPEAT 10 [TRIANGLES RT 36]
END

TO TREE :SIDE
RT 30 TRIANGLER :SIDE
RT 60 FD :SIDE / 2
LT 90 BK :SIDE / 2
END
```

```
TO TREES
TREE 30
TREE 40
TREE 50
END
```

Arithmetic

PR 5 +3

As you have learned from the examples above, you can do arithmetic in Logo. For example,

```
PR 5 + 3
```

Logo types

```
8
```

PR 4 * 23

```
PR 4 * 23
```

Logo types

```
92
```

PR 345 – 32

```
PR 345 - 32
```

Logo types

```
313
```

PR 25 / 5

```
PR 25 / 5
```

Logo types

```
5.
```

Logo Numbers

Notice that the result of 25 divided by 5 is a decimal number. Logo has both decimal and integer numbers. Logo will perform arithmetic with decimals or integers.

Some computations always result in a decimal answer. Division (/) is one of those. Other arithmetic operations depend on what they are given as inputs. For example, if you type

```
PR 4 * 2.3
```

PR 4 * 2.3

Logo types

```
9.2
```

For more discussion about arithmetic in Logo consult the *Reference Manual.*

Logo Vocabulary

The character : (dots) informs Logo that the word to which it is prefixed names a container which can contain a number, another word, a list of words, or a list of lists.

Arithmetic Operations

```
/
-
+
*
```

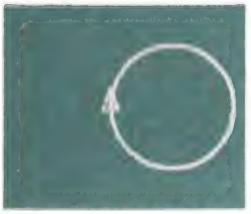
Chapter 13

Circles

The turtle can make curved as well as straight line designs. Curves are made by repeatedly taking a small step and then turning just a little bit.

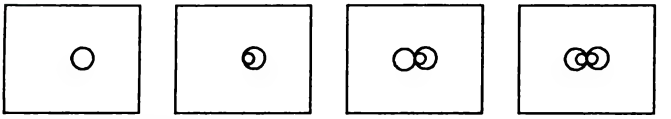
To make a complete circle the turtle has to turn 360 degrees. Try this:

```
REPEAT 360 [FD 1 RT 1]
```



REPEAT 360 [FD 1 RT 1]

If you want to vary the size of the circle, use the command CIRCLER (pronounced circle-are) or CIRCLEL (pronounced circle-ell) with an input for the size of the circle. The input is a number which is the radius, i.e., the distance from the edge of the circle to its center. Try



```
CIRCLER 20
CIRCLER 10
CIRCLEL 20
CIRCLEL 10
```

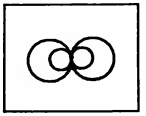
CIRCLER causes the turtle to draw a circle by turning right and CIRCLEL causes the turtle to draw a circle by turning left.

Bug Box

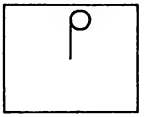
If Logo does not know how to CIRCLER or CIRCLEL skip this chapter. Appendix A offers an explanation of why Logo might not know how to do these procedures.

Projects Using Circles

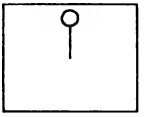
Here are some projects using circles.



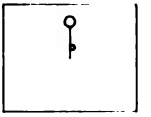
EYES



BALLOON



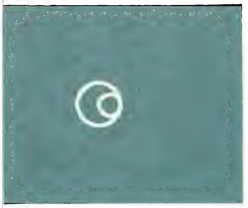
LOLLIPOP



FLOWER

REYE

```
TO EYES
REYE
LEYE
END

TO REYE
CIRCLER 20
CIRCLER 10
END
```

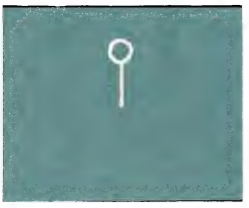


LEYE

```
TO LEYE
CIRCLEL 20
CIRCLEL 10
END
```

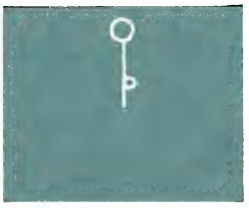
BALLOON

```
TO BALLOON
FD 40
CIRCLER 10
END
```



LOLLIPOP

```
TO LOLLIPOP
FD 40
LT 90
CIRCLER 10
END
```



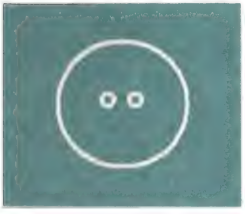
FLOWER

```
TO FLOWER
FD 20
CIRCLER 5
LOLLIPOP
END
```

In this example we define FACE with an input. It uses several procedures: HEAD, EYES, MOUTH and NOSE.

FACE 30

HEAD 30

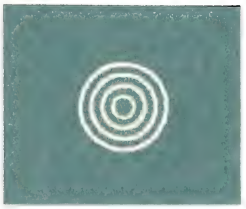


EYES 6

MOUTH 6

NOSE 6

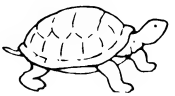
```
TO FACE :SIZE
HEAD :SIZE
EYES :SIZE / 5
MOUTH :SIZE / 5
NOSE :SIZE / 5
END

TO HEAD :SIZE
PU FD :SIZE
RT 90
PD CIRCLER :SIZE
PU LT 90
BK :SIZE PD
END

TO EYES :SIZE
LEYE :SIZE
REYE :SIZE
END

TO LEYE :S
PU LT 90 FD :S PD
CIRCLER :S / 2
PU BK :S RT 90 PD
END

TO REYE :S
PU RT 90 FD :S PD
CIRCLEL :S / 2
PU BK :S LT 90 PD
END

TO MOUTH :SIZE
PU BK 2 * :SIZE
RT 90 FD :SIZE / 2
PD BK :SIZE
PU FD :SIZE / 2
LT 90 FD 2 * :SIZE
END

TO NOSE :S
BK :S
END
```



BULLSEYE

Another one is BULLSEYE:

```
TO BULLSEYE
HEAD 10
HEAD 20
HEAD 30
HEAD 40
END
```



Chapter 14

Commands with Two Inputs: Polygons and Arcs

Just as you can vary the number of steps the turtle takes, you can also vary how much it turns. In fact, you can get some really beautiful and surprising designs by varying these two components of the turtle's state. The following procedure takes two inputs: one which specifies the number of turtle steps and one which specifies the amount of turn.

```
TO POLY :STEP :ANGLE
FD :STEP
RT :ANGLE
POLY :STEP :ANGLE
END
```

POLY 30 90

Now try it!

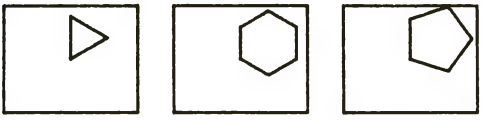
```
POLY 30 90
```

The turtle does not stop because POLY keeps telling it to go forward and turn. To stop POLY and the turtle press CTRL-G. Logo responds

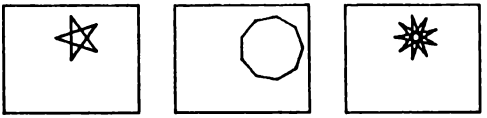
```
STOPPED!
?
```

You probably will want to clear the screen (CS).

Here are some suggestions for exploring POLY. You will no doubt want to try other inputs, too.



```
POLY 30 120
POLY 30 60
POLY 30 72
```

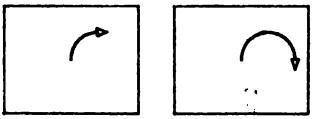


```
POLY 30 144
POLY 30 40
POLY 30 160
```

POLY is recursive. That is, instead of calling another procedure to help out, POLY calls itself. This is a simple example of recursion. We will meet other examples later.

Turtle Draws Arcs

Many projects require only pieces of circle. In those cases you can use the commands ARCRIGHT (ARCR) and ARCLEFT (ARCL). These procedures need 2 inputs: the first input is the radius of the circle from which the arc is taken and the second input is the number of degrees of the arc.



```
ARCR 30 90
ARCR 30 90
```

Bug Box

If Logo does not know how to ARCR or ARCL skip the rest of this chapter. Appendix A offers an explanation of why Logo might not know how to do these procedures.

Clear the screen and try some other inputs.

```
ARCL 40 120
LT 120
ARCL 40 120
```

A Fish!

```
ARCR 40 90
RT 90
ARCR 40 90
RT 90
```

A Petal!

Arcs, Petals, Flowers, Swans

```
TO PETAL :SIZE
ARCR :SIZE 90 RT 90
ARCR :SIZE 90 RT 90
END
```

Try

```
PETAL 40 PETAL 30
```

```
REPEAT 8 [PETAL 40 PETAL 30 RT 45]
```

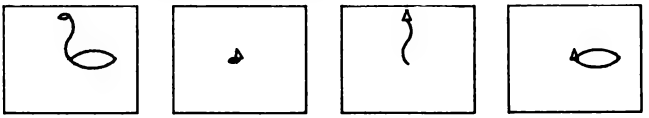
```
REPEAT 4 [PETAL 30 RT 45 PETAL 40 RT !
45]
```

PETAL 40 PETAL 30

FLOWER

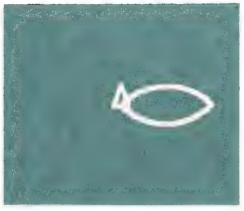
Notice that Logo lines can extend beyond one screen line. Logo marks continuation lines by putting a ! (exclamation point) in the last character position of the line. The rest of the text flows onto the next screen line.

You might want to make a FLOWER out of one of these designs. Now let's do a SWAN.



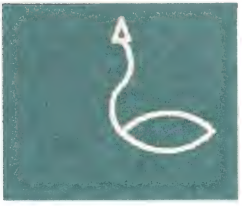
SWAN HEAD NECK BODY

There are really only 2 different shapes, PETAL and NECK. We can use PETAL to make BODY and HEAD. So



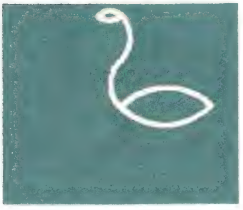
BODY 60

```
TO BODY :SIZE
RT 45
PETAL :SIZE
LT 45
END
```



NECK 30

```
TO NECK :SIZE
LT 45
ARCR :SIZE 90
ARCL :SIZE 90
RT 45
END
```



SWAN 60

```
TO HEAD :SIZE
LT 135
PETAL :SIZE
RT 135
END
```

SWAN, the controlling procedure, is

```
TO SWAN :SIZE
BODY :SIZE
NECK :SIZE / 2
HEAD :SIZE / 4
END
```

Chapter 15

Some More Advanced Techniques: Spirals

The POLY procedure makes the turtle draw closed figures. An exception occurs when the turtle turns 0 or 360 degrees (or a multiple of 360) on each round. Then, it walks in a straight line.

```
TO POLY :STEP :ANGLE
FD :STEP
RT :ANGLE
POLY :STEP :ANGLE
END
```

The turtle draws closed figures because it goes forward and rotates a fixed amount and it eventually gets back to where it started. We can easily have the turtle draw a spiral by increasing its forward step on each round of the procedure. Let's change POLY and name it SPI and thus make a spiral drawing procedure.

```
EDIT "POLY
```

Now the Logo editor shows the POLY procedure.

```
TO POLY :STEP :ANGLE
FD :STEP
RT :ANGLE
POLY :STEP :ANGLE
END
```

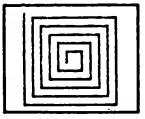
Now change the name of the procedure from POLY to SPI.

```
TO SPI :STEP :ANGLE
FD :STEP
RT :ANGLE
SPI :STEP :ANGLE
END
```

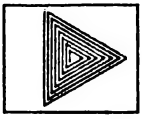
We make one more change to the recursion line. We tell SPI to add 2 steps to :STEP. Thus,

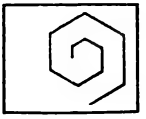
```
TO SPI :STEP :ANGLE
FD :STEP
RT :ANGLE
SPI :STEP + 2 :ANGLE
END
```

Now try SPI.



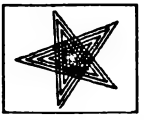
```
SPI 5 90
SPI 5 120
SPI 5 60
```



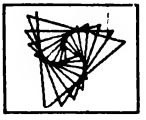


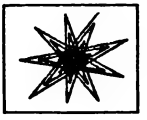
Remember CTRL-G stops.

CTRL-L shows the whole turtle field.

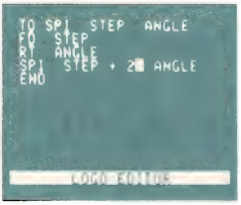


```
SPI 5 144
SPI 5 125
SPI 5 160
```





Try SPI with other inputs. Change SPI and give it a third input, :INC, which SPI will add to :STEP instead of 2. Then you can change how much the turtle's step increases by choosing different numbers for the third input.




```
ED "SPI
```

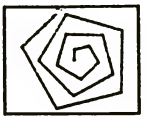
The Logo editor shows this procedure.

```
TO SPI :STEP :ANGLE
FD :STEP
RT :ANGLE
SPI :STEP + 2 :ANGLE
END
```

Now change the procedure so that it looks like this.

```
TO SPI :STEP :ANGLE :INC
FD :STEP
RT :ANGLE
SPI :STEP + :INC :ANGLE :INC
END
```

Now try



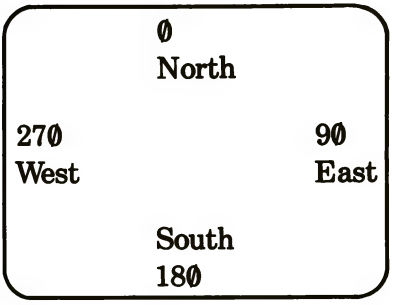
```
SPI 5 75 1
SPI 5 75 2
```

You might want to stop the turtle at different places. Try other inputs.

Chapter 16

The Turtle's Field

The turtle has a *position* and a *heading*. The turtle's heading is described in degrees like a compass reading with 0 or north at the top of the screen. Then 90 degrees is directly east, 180 degrees is directly south, and 270 degrees is directly west. We could mark the screen:



When the turtle starts up its heading is 0. After CS the turtle has a heading of 0. At any time you can get a compass reading. Try

```
RT 90
PR HEADING
```

Logo responds

```
90.
```

HEADING outputs the turtle's direction.

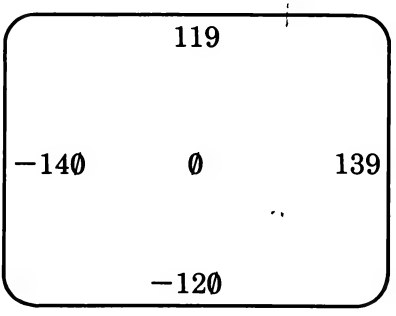
HEADING is part of Logo's vocabulary. It is different from PRINT or FORWARD. It is not a command; it is an operation. It does not cause something to happen, but rather outputs something which can be used as an input. In this section several other operations are introduced.

The turtle's position is described by two numbers which indicate how far the turtle is from the center. For example, POSITION at the start is [0 0].

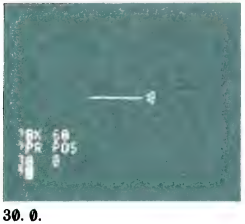
The first number indicates the turtle's location along the horizontal or x-axis. If the turtle is west of the center, then the number will be negative.

The second number indicates the turtle's location along the vertical or y-axis. If the turtle is south of the center, then the number will be prefixed by a − (minus sign).

The turtle screen can be represented by a grid divided into coordinates. The x coordinates run along the horizontal and the y coordinates run along the vertical. The turtle at the center has both XCOR and YCOR equal to 0. The screen dimensions are approximately:



−30. 0.



30. 0.

For example, if you type

```
LT 90
FD 30
PR POSITION
```

Logo responds

```
−30. 0.
```

The turtle is 30 steps west of the center along the horizontal.

```
BK 60
PRINT POS
```

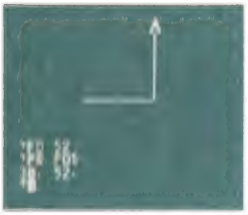
POS is the short name for POSITION.

and Logo responds

```
30. 0.
```

Now the turtle is 30 steps east of the center along the horizontal.

```
RT 90
FD 52
PR POS
```

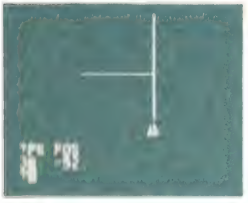


And Logo now responds

```
30. 52.
```

The turtle is 30 steps east of the center and 52 steps north of the center.

```
BK 104
PR POS
```



Logo responds

```
30. -52.
```

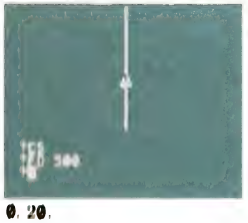
The turtle is now southeast of the center 30 steps east and 52 steps south.

WRAP, FENCE, and WINDOW

The turtle starts out being able to WRAP; it can walk off one edge of the screen and on at the opposite edge. It does not change direction.

Type

```
CS
FD 500
PR POS
```



Logo responds

```
0. 20.
```

Notice that the turtle is 20 steps and not 500 steps from the center.

The screen boundaries can be set up so that the turtle cannot move off the screen by typing

```
FENCE
```

Now type

```
CS
FD 500
```

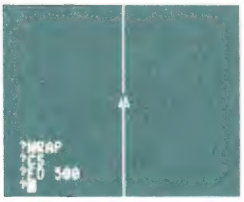
FENCE

Logo responds

```
TURTLE OUT OF BOUNDS
```

The turtle screen will act this way until you type

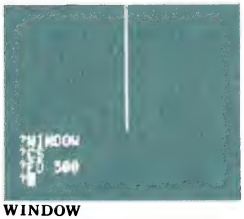
```
WRAP
```



WRAP

Now the turtle will wrap around the screen.

The command WINDOW allows the turtle to move off the screen, but the turtle does not wrap. Thus the turtle might often be invisible to you, but still be carrying out your orders. The x and y coordinates can be very large. CS always restores the turtle to its center position on the screen.



WINDOW

```
CS
WINDOW
FD 500
PR POS
```

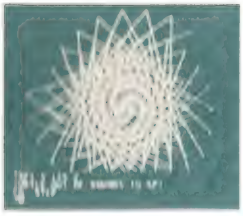
Logo responds

```
0. 500.
```

The turtle is now 500 steps from the center and out of view.

Try using SPI after typing FENCE, and typing WINDOW, and after typing WRAP.

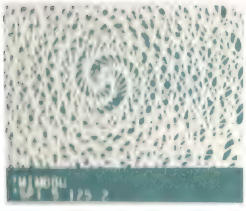
```
CS
FENCE
SPI 5 125 2

CS
WINDOW
SPI 5 125 2
```



FENCE

You will probably want to interrupt SPI by typing CTRL-G.

```
CS
WRAP
SPI 5 125 2
```

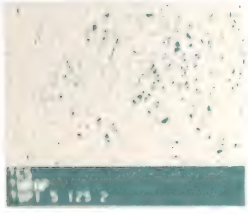


WINDOW

Using POSITION to Draw

There is an easy way to draw a right triangle provided you know the lengths of the two sides joining in the right angle. We record the starting position of the turtle. We do this by using the Logo command MAKE.

```
MAKE "START POS
```



WRAP

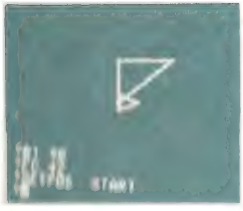
MAKE does two things. It puts the output from POS in your workspace and makes START its name. Thus if you say

```
PR :START
```

and the turtle was in the center of the screen when you typed MAKE "START POS, Logo will respond

```
0. 0.
```

Now have the turtle draw the two sides:



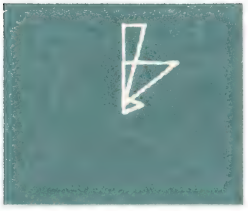
```
FD 33
RT 90
FD 42
```

and now we use the command SETPOS

```
SETPOS :START
```

The turtle will be moved to :START and since the pen is down a line will be drawn. A procedure for this is

```
TO TRI :SIDE1 :SIDE2
MAKE "START POS
FD :SIDE1
RT 90
FD :SIDE2
SETPOS :START
END
```



TRI 40 50
TRI 75 20

Try

```
CS
TRI 40 50
TRI 75 20
```

Notice that HEADING and POSITION are Logo words which cause Logo to output information. We call this kind of word an operation. You can make your own operations.

Logo Vocabulary

The following are operations:

Full Name	Short Name
HEADING	
XCOR	
YCOR	
POSITION	POS
BACKGROUND	BG
PENCOLOR	PC

Some commands that directly affect what these operations output are:

SETHEADING	SETH
SETX	
SETY	
SETPOS	
SETBG	
SETPC	
FORWARD	FD
BACK	BK
RIGHT	RT
LEFT	LT

Chapter 17

A Game Project

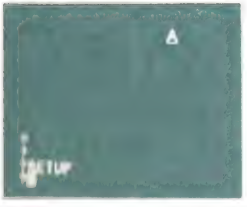
Creating a Game

Our game works like this. A target and a turtle appear somewhere on the screen. The player tries to get the turtle into the target with the smallest number of moves.

For a first version the moves will be regular Logo commands like LT 45 or FD 80. Later we will refine the game by assigning keys or paddles to direct the turtle. Developing the game in stages illustrates a kind of "project management" to which Logo is well-suited.

For our target game we need to set up a target. Then we need to set up the turtle. We can write one procedure that will be good for both tasks. An example of a setup procedure is printed below. SETUP sets the turtle up in a random position on the screen. It leaves the turtle heading in the same direction as it was at the start of SETUP.

```
TO SETUP
PU
RT RANDOM 360
FD RANDOM 100
SETHEADING 0
PD
END
```



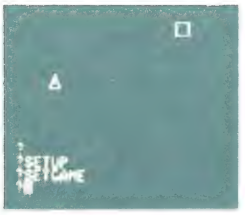
SETUP

The Logo operation RANDOM outputs a number which Logo makes up. This number is always less than the one RANDOM is given as its input.

In SETUP, for example, the turtle turns some angle which can be as small as 0 degrees or as large as 359 degrees. The actual number is computed each time RANDOM is used. The input to FD is also a random number. Here the number can be no larger than 99. Notice that SETUP leaves the turtle facing north.

SETUP can be used to set up the turtle as well as the target. It's a good idea to first put the turtle back in the center.

The following procedure, SETGAME, sets up the game.

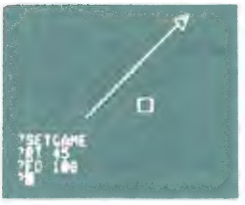


SETGAME

```
TO SETGAME
CS
SETUP
TARGET
PU
SETPOS [0 0]
SETUP
END
```

SETGAME calls TARGET

```
TO TARGET
BOXR 10
END
```



A Miss!

Use SETGAME a few times. It is hard at first. For example,

```
SETGAME
RT 45
FD 100
```

A miss!

Making a Key Into a Game Button

There are many kinds of interactive programs that you can write. You can have Logo ask questions and receive answers in words or sentences. Sometimes you want to trigger Logo into action by a touch of a key. This requires using the operation READCHAR or RC. Type

```
PR RC
```

Logo waits for a key to be pressed.

Type the character A.

Notice that the character does not appear on the screen when you type it. In other words Logo does not "echo" what you type to it.

When the PRINT command is carried out Logo puts an A on the screen.


```
A
```

Logo does not wait for you to type anything else. It acts immediately. Try READCHAR or RC a few more times. Note that if you type RC (followed by RETURN) and then type in a character like X Logo responds

```
I DON'T KNOW WHAT TO DO WITH X
```

RC is an operation like HEADING or POSITION. It is used as an input to another command or operation. For example, we could name RC's output using MAKE.

```
MAKE "KEY RC
```

Now type the character Z.

:KEY will be the character Z. To check this out type

```
PRINT :KEY
```

and Logo will respond

```
Z
```

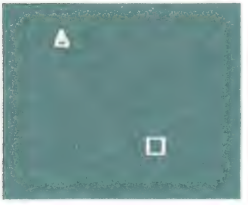
We can use this idea of giving things names so that we can talk about them. Imagine we have a procedure called PLAY. If you type F, L, or R, the following happens.

F makes the turtle move forward 10 steps.
R makes the turtle turn right 15 degrees.
L makes the turtle turn left 15 degrees.

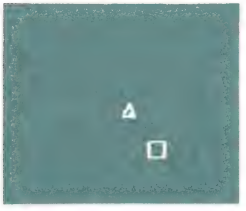
```
TO PLAY
MAKE "ANSWER RC
IF :ANSWER = "F [FD 10]
IF :ANSWER = "R [RT 15]
IF :ANSWER = "L [LT 15]
PLAY
END
```

PLAY

In PLAY :ANSWER is what RC outputs. PLAY then checks :ANSWER using the Logo primitive, IF. IF requires two inputs. The first input is either TRUE or FALSE. The second input is a list of instructions to be carried out when the first input is TRUE.

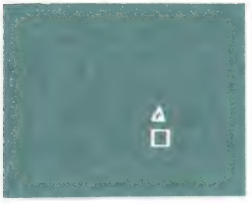


In this example, we use the Logo operation =, which compares its two inputs. This operation outputs TRUE when the inputs are the same; it outputs FALSE when the two inputs are not the same. This operation, like the arithmetic operations (+ − / *), comes between its inputs.



Notice that PLAY is recursive. That is, the last line of the procedure PLAY calls PLAY. PLAY does not stop unless it has a bug or you press CTRL-G.

Try it. Make turtle scribblings all over the screen.



Expanding the Game Project

In this section we build a better target game out of SETGAME and PLAY. Some of the techniques used in this game are unfamiliar to you others are not. We can make a procedure, GAME, which uses SETGAME and then PLAY.

```
TO GAME
SETGAME
PLAY
END
```

Try GAME.

```
GAME
```

Perhaps we should raise the turtle's pen. It would also be nice if GAME printed some instructions.

GAME

```
TO GAME
RULES
SETGAME
PU
PLAY
END

TO RULES
SPLITSCREEN
PR [HIT THE TARGET WITH THE TURTLE]
PR [TYPE R OR L TO TURN AND F TO AD!
VANCE]
END
```

Try GAME now.

```
GAME
```

This is much better, but there is room for improvement. The game plays too slowly. Let's make it more challenging.

Let's give the player only one chance to land on the target. The player can turn the turtle many times, but will have only one chance to tell the turtle how far to go forward.

Here is the plan: after Logo sets up the scene for the game, we want it to let you play the game. Once you've made your try, you can see if you landed in the target. Logo should leave unchanged the screen for a little while and then start the game again with a brand new target and position.

There is one liberty we have taken: your Apple screen can show 40 characters on a line; in this guide we can show only 37 characters across the page.

We use a "top-down" approach to plan this game. That means we plunge in and write the overall structure of the game before we know how we are going to write all the details.

```
TO GAME
RULES
SETGAME                    This sets up each game.
PU
PLAY
WAIT 100                   Logo waits a little while.
GAME                       Now start a new game.
END
```

We edit the procedure PLAY to give you *only one chance* to move the turtle forward into the target. The point of the game is to judge the distance. When you press the T key (T for try), you get your only chance to land in the target.

```
TO PLAY
MAKE "ANSWER RC
IF :ANSWER = "R [RT 15]
IF :ANSWER = "L [LT 15]
IF :ANSWER = "T [TRYLANDING STOP]
PLAY
END
```

The STOP command is very important. It makes the procedure stop after you have tried landing.

Now edit RULES and change F to T.

```
TO RULES
PR [HIT THE TARGET WITH THE TURTLE]
PR [TYPE R OR L TO TURN AND T TO TRY!
LANDING]
END
```

We've used the "top-down" approach again; we've changed PLAY to use a procedure named

TRYLANDING which we haven't defined yet! Let's define it now:

```
TO TRYLANDING
PR [HOW FAR DO YOU WANT TO MOVE FOR!
WARD]
FD READWORD
END
```

READWORD is like RC except you can type a word instead of a single character (in this case a number). It waits for you to press the RETURN key to signal that you are done. READWORD outputs the word you typed.

Now we have written the whole game. To try it, type

```
GAME
```

Remember that you can give the commands R and L to turn the turtle, and T to try landing on the target. After you type T, Logo will wait for you to type a number and then RETURN.

You may be able to adapt this game and the techniques used in it to make other interactive games. You can also add many improvements to this game. For example, have Logo figure out whether you landed on the target. Logo could also keep track of your score.

Bug Box

If Logo does not know READWORD, read this section.

READWORD is not a primitive. It is written in Logo. The definition is

```
TO READWORD
OUTPUT FIRST READLIST
END
```

As we already know, READWORD is an operation and outputs a word. READLIST is a primitive. It too is an operation, but it outputs a list. So READWORD uses READLIST, but takes only the first word you type.

Operations output a word or a list. The command OUTPUT or OP outputs something, and stops the procedure at that point.

Comments on Using IF

IF expects its first input to be either TRUE or FALSE. The = is a special kind of operation. It outputs either TRUE or FALSE. We call this kind of operation a *predicate*. Predicates are used as the first input to IF.

When the first input to IF is TRUE Logo carries out the commands or the operation contained in the second input, which is a list. When the first input is FALSE the second input is ignored.

A Note on Logo Grammar

Procedures can be commands or operations (or both). Commands are imperatives and order Logo to do something. Operations output something and thus are used as inputs either to commands or to other operations.

Logo Vocabulary

Full Name	Short Name
IF	
STOP	
OUTPUT	OP
RANDOM	
READCHAR	RC
READLIST	RL
FIRST	
=	

Chapter 18

Recursive Procedures

One of the most powerful features of Logo is that you can divide a project into procedures, each of which is a distinct entity that has its own name. A procedure can be called (or used) by any other procedure; it can also call other procedures. Some procedures call on themselves; these procedures are *recursive*. We have already used recursive procedures. For example, POLY and SPI are both recursively defined.

```
TO POLY :STEP :ANGLE
FD :STEP
RT :ANGLE
POLY :STEP :ANGLE     This is the recursive call.
END

TO SPI :STEP :ANGLE :INC
FD :STEP
RT :ANGLE
SPI :STEP + :INC :ANGLE :INC
END
```

POLY calls POLY as part of its definition; and SPI calls SPI.

Let's think about this process. Imagine that Logo has an inexhaustible supply of helpers who are computer creatures living in the computer. Every time a procedure is called a helper is called upon to look up the definition of the procedure. The helper then begins to carry out the instructions. It does this by calling on other helpers. Usually several helpers are needed to carry out one procedure.

For example, when POLY is called its helper calls a FD helper. After the FD helper finishes, a RT helper is called. When it finishes, a POLY helper is called. The second POLY helper calls a FD helper, a RT helper, and a POLY helper. Meanwhile, the first POLY helper is still around waiting for the second

POLY helper to finish. In the process the FD helpers and the RT helpers finish their jobs (we don't know who they call for help). The POLY helpers never finish; they keep on calling for new POLY helpers.

When you use POLY the process continues until you type CTRL-G. Not all recursive procedures work this way. They can be made to stop. In fact, making up appropriate "stop rules" is a major part of writing recursive procedures.

Stopping Recursive Procedures

In this section we will discuss recursive procedures that stop. We will use as an example a procedure that counts backwards from a given number. Assume we have a procedure called COUNTBACK and we type

```
COUNTBACK 55
```

Logo will respond

```
55
54
53
52
51
```

and so on, until it reaches 1.

We can quickly sketch out part of the COUNTBACK procedure.

```
TO COUNTBACK :NUMBER
PR :NUMBER
```

Now we want COUNTBACK to do the same thing using :NUMBER – 1. As part of a recursive definition we can have COUNTBACK call on another

COUNTBACK for help, but this time give it :NUMBER – 1 as input.

```
COUNTBACK :NUMBER - 1
END
```

Let's try it with a small number like 5.

```
COUNTBACK 5
```

Logo responds

```
5
4
3
2
1
0
-1
-2
```

and so on until you type CTRL-G.

The procedure definition is almost complete. It needs a stop condition. Let's make a stop rule for COUNTBACK. For example, if the input is 0 then stop.

```
IF :NUMBER = 0 [STOP]
```

Now COUNTBACK looks like this:

```
TO COUNTBACK :NUMBER
IF :NUMBER = 0 [STOP]
PR :NUMBER
COUNTBACK :NUMBER - 1
END
```

Try it by typing

```
COUNTBACK 5
```

Logo responds

```
5
4
3
2
1
```

Everything stops. The last number that is printed is 1.

Let's now think about this process in more detail and try to understand how Logo's helpers interact.

A Model for Recursion: Helpers

There is an inexhaustible supply of helpers readily available to Logo. Each time a procedure is called, whether it be PRINT or MAKE or POLY or COUNTBACK, a helper is put on the job. It looks up the definition of the procedure and starts reading it. The helper carries out what it can and then calls other helpers as they are needed. For each procedure call, a helper is put on the job. Whenever a helper finishes its job it reports back to whoever called it and goes away.

Let's revive the image of named things like :SIDE or :START as contents of a container. Thus, if a procedure takes an input the helper puts that input in the container named on the title line. If something is already there, the helper leaves it there, and puts the new thing on top of it to be available later. These containers can hold a lot. When the helper has finished carrying out the procedure it disposes of the input and then what was under it in the container is now on top and accessible.

Let's play computer by sketching out what the helpers do. To do this give COUNTBACK a small number as input. So

```
COUNTBACK 2
```

Immediately a helper is called on and given the procedure COUNTBACK with an input of 2. Let's draw a diagram of the interaction.

```
COUNTBACK 2
```

COUNTBACK :NUMBER
  COUNTBACK 2 is called.
  The input, NUMBER, is 2.
  COUNTBACK 2 stops if NUMBER = 0.
  But NUMBER is 2.
  COUNTBACK 2 continues.

```
2
```

PR :NUMBER
    COUNTBACK :NUMBER − 1
    COUNTBACK 1 is called by
    COUNTBACK 2.
    NUMBER is 1.
    COUNTBACK 1 stops if NUMBER = 0.
    But NUMBER is 1.
    COUNTBACK 1 continues.

```
1
```

PR :NUMBER
        COUNTBACK :NUMBER − 1
        COUNTBACK 0 is called by
        COUNTBACK 1.
        NUMBER is 0.
        COUNTBACK 0 stops if NUMBER = 0.
        So it stops.
    COUNTBACK 1 now continues, but it
    has finished its job and stops.
  COUNTBACK 2 now continues, but it has

```
?
```

finished its job and stops.

Logo responds

```
5
4
3
2
1
```

Everything stops. The last number that is printed is 1.

Let's now think about this process in more detail and try to understand how Logo's helpers interact.

A Model for Recursion: Helpers

There is an inexhaustible supply of helpers readily available to Logo. Each time a procedure is called, whether it be PRINT or MAKE or POLY or COUNTBACK, a helper is put on the job. It looks up the definition of the procedure and starts reading it. The helper carries out what it can and then calls other helpers as they are needed. For each procedure call, a helper is put on the job. Whenever a helper finishes its job it reports back to whoever called it and goes away.

Let's revive the image of named things like :SIDE or :START as contents of a container. Thus, if a procedure takes an input the helper puts that input in the container named on the title line. If something is already there, the helper leaves it there, and puts the new thing on top of it to be available later. These containers can hold a lot. When the helper has finished carrying out the procedure it disposes of the input and then what was under it in the container is now on top and accessible.

Let's play computer by sketching out what the helpers do. To do this give COUNTBACK a small number as input. So

```
COUNTBACK 2
```

Immediately a helper is called on and given the procedure COUNTBACK with an input of 2. Let's draw a diagram of the interaction.

```
COUNTBACK 2     COUNTBACK :NUMBER
                  COUNTBACK 2 is called.
                  The input, NUMBER, is 2.
                  COUNTBACK 2 stops if NUMBER = 0.
                  But NUMBER is 2.
                  COUNTBACK 2 continues.
2                 PR :NUMBER
                        COUNTBACK :NUMBER - 1
                        COUNTBACK 1 is called by
                        COUNTBACK 2.
                        NUMBER is 1.
                        COUNTBACK 1 stops if NUMBER = 0.
                        But NUMBER is 1.
                        COUNTBACK 1 continues.
1                       PR :NUMBER
                                COUNTBACK :NUMBER - 1
                                COUNTBACK 0 is called by
                                COUNTBACK 1.
                                NUMBER is 0.
                                COUNTBACK 0 stops if NUMBER = 0.
                                So it stops.
                        COUNTBACK 1 now continues, but it
                        has finished its job and stops.
                  COUNTBACK 2 now continues, but it has
                  finished its job and stops.
```

A helper stops when there is nothing more to do or when STOP is encountered.

Let's change the procedure so that the print action occurs in a different place. Make a new procedure, CB.

```
TO CB :NUMBER
IF :NUMBER = 0 [STOP]
CB :NUMBER - 1
PR :NUMBER
END
```

Type

```
CB 5
```

Logo responds

```
1
2
3
4
5
```

This is probably a surprise. But remember when the first CB helper calls the second CB helper, it waits for the second to do its job, and then the first helper continues with its own job.

Let's trace through the process giving CB the number 2 as input.

```
CB 2                CB :NUMBER
                      CB 2 is called.
                      NUMBER is 2.
                      CB 2 stops if NUMBER = 0.
                      But NUMBER is 2 so CB 2 continues.
                    CB :NUMBER - 1.
                          CB 1 is called by CB 2.
                          NUMBER is 1.
                          CB 1 stops if NUMBER is 0.
                          But NUMBER is 1 so CB 1 helper continues.
                    CB :NUMBER - 1
                                CB 0 is called by CB 1.
                                NUMBER is 0.
                                CB 0 stops if NUMBER = 0.
                                So it stops.
                          CB 1 continues.
1                   PR :NUMBER
                          CB 1 has finished its job and stops.
                      CB 2 continues.
2                   PR :NUMBER
?                     CB 2 now has finished its job and stops.
```

Note that the stop rule has to come before the recursion line. Otherwise, the recursion will go on forever and the stop rule will never be reached. Often, the stop rule will be the very first line in the procedure.

There are other kinds of stop rules we could make up. For example, in SPI

```
TO SPI :STEP :ANGLE :INC
FD :STEP
RT :ANGLE
SPI :STEP + :INC :ANGLE :INC
END
```

We could decide that SPI should stop if :STEP is greater than 200. So, we include the line

```
IF :STEP > 200 [STOP]
```

After editing SPI it looks like

```
TO SPI :STEP :ANGLE :INC
IF :STEP > 200 [STOP]
FD :STEP
RT :ANGLE
SPI :STEP + :INC :ANGLE :INC
END
```

Try SPI. If you don't like the stop rule, change it.

Designing a stop rule for POLY can be a little trickier. POLY completes a figure when the turtle returns to its starting state, which may require the turtle to turn only one complete rotation, that is, only 360 degrees. But sometimes the turtle turns a multiple of 360 degrees.

Actually, we only need to know what the turtle's heading was when it started, and then compare it to the turtle's heading after each turn. So before POLY is called we have to find the turtle's heading.

```
MAKE "START HEADING
```

Then POLY can check to see if the turtle's current heading is the same as :START.

```
IF HEADING = :START [STOP]
```

The edited POLY looks like:

```
TO POLY :STEP :ANGLE
FD :STEP
RT :ANGLE
IF HEADING = :START [STOP]
POLY :STEP :ANGLE
END
```

Now try POLY.

There is a problem here. Each time you call POLY you have to save the starting heading. It would be better to put that action in a procedure. Let's rename POLY and call it POLY1. Then we can make a new POLY which sets up :START and then runs POLY1.

```
TO POLY :STEP :ANGLE
POLY1 :STEP :ANGLE HEADING
END

TO POLY1 :STEP :ANGLE :START
FD :STEP
RT :ANGLE
IF HEADING = :START [STOP]
POLY1 :STEP :ANGLE :START
END
```

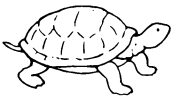
Now POLY will do the whole job.

There are many designs you can create with SPI and POLY. There are many ways of combining polygons to make new and dazzling designs.

This introduction to Logo is finished, but we hope you will explore other turtle projects on your own.

The *Reference Manual* describes other features of Logo which you might want to try now.

Have fun.



Appendix A

Logo Startup File

Appendix A Logo Startup File

On the Logo diskette or your file diskette there is a file named STARTUP. It is loaded into your workspace whenever Logo starts up. We have put the following procedures in that file:

Full Name	Short Name
ARCLEFT	ARCL
ARCRIGHT	ARCR
CIRCLEL	
CIRCLER	
READWORD	RW

Although we have been using these procedures as if they were Logo primitives, they are procedures we have written in Logo. If your Logo doesn't know about these procedures, you have probably rewritten your STARTUP file. You can type in the definitions below and put them in your STARTUP file.

We saved them in the STARTUP file in such a way that, even though they are loaded into your workspace, they do not appear when you use POTS (printout titles), POPS (printout procedures), or POALL.

These procedures as well as ARC1 and ARCL1 are organized into a package named AIDS. We use the command PACKAGE to do this.

```
PACKAGE "AIDS [ARCLEFT ARCL ARCRIGHT !
ARCR ARCR1 ARCL1 CIRCLEL CIRCLER RW !
READWORD]
```

When this package of procedures is loaded into your workspace, the procedures are available to you as if they were Logo primitives. They are not printed out with the other procedures in the workspace. This is because the entire package of procedures has been buried from view by the command

```
BURY "AIDS
```

If you want to know what procedures are in a package, type

```
POTS "AIDS
```

and Logo will respond

```
TO ARCLEFT :RADIUS :DEGREES
TO ARCL :RADIUS :DEGREES
TO ARCRIGHT :RADIUS :DEGREES
TO ARCR :RADIUS :DEGREES
TO CIRCLEL :RADIUS
TO CIRCLER :RADIUS
TO READWORD
TO RW
TO ARCR1 :STEP :TIMES
TO ARCL1 :STEP :TIMES
```

The file STARTUP contains the information that these procedures are in the AIDS package. This is how we made the file.

```
SAVE "STARTUP [AIDS]
```

We saved in the file STARTUP only those procedures in the package AIDS.

You can put more procedures in STARTUP and then they will be loaded whenever Logo starts up. If you wish to do so put them in another package using the PACKAGE command. For example,

```
PACKAGE "MINE [POLY SPOLY SPI]
```

Logo does not let you use the same file name again. Since STARTUP already exists, you will have to erase it.

```
ERASEFILE "STARTUP
```

Then when you save, type

```
SAVE "STARTUP [AIDS MINE]
```

Notice that we include AIDS in the list of packages to be saved. We do this so that STARTUP will continue to contain the procedures in the AIDS package.

Bug Box

To save your procedures and other Logo things you need a formatted Apple diskette on which you can write. If you do not have one you can make one or ask someone to help you make one.

Using the Apple DOS 3.3 SYSTEM MASTER you can format a diskette. Put the diskette in the disk drive and start it up by repowering your Apple. Then when the disk drive light goes off, remove the SYSTEM MASTER diskette and replace it with the blank diskette you want to format. Now type

```
INIT HELLO
```

When the disk drive light goes off the diskette will be formatted. You can test this out by typing

```
CATALOG
```

The file name HELLO should be displayed.

Now put the Logo diskette in the disk drive. Repower (reboot) the Apple. When Logo prints

```
PRESS THE RETURN KEY TO BEGIN

IF YOU HAVE YOUR OWN FILE DISKETTE,
INSERT IT NOW, THEN PRESS RETURN
```

press the RETURN key. Do not take the Logo diskette out of the drive yet. Let Logo load in the STARTUP file from the diskette.

After Logo prints

```
WELCOME TO LOGO
```

remove the Logo diskette and insert your file diskette. Now type

```
SAVE "STARTUP "AIDS
```

Now you have your own file diskette with its STARTUP file. You can, of course, change it.

ARCS and CIRCLES

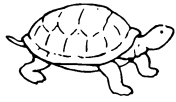
```
TO ARCRIGHT :RADIUS :DEGREES
ARCR1 .174532 * :RADIUS :DEGREES / 10
IF 0 = REMAINDER :DEGREES 10 [STOP]
FD .174532 * :RADIUS / 20 / REMAINDER
:DEGREES 10
RT REMAINDER :DEGREES 10
END

TO ARCR :RADIUS :DEGREES
ARCRIGHT :RADIUS :DEGREES
END

TO ARCLEFT :RADIUS :DEGREES
ARCL1 .174532* :RADIUS :DEGREES / 10
IF 0 = REMAINDER :DEGREES 10 [STOP]
FD .174532 * :RADIUS / 20 / REMAINDER
:DEGREES 10
LT REMAINDER :DEGREES 10
END

TO ARCL :RADIUS :DEGREES
ARCLEFT :RADIUS :DEGREES
END

TO ARCR1 :STEP :TIMES
REPEAT :TIMES [RT 5 FD :STEP RT 5]
END

TO ARCL1 :STEP :TIMES
REPEAT :TIMES [LT 5 FD :STEP LT 5]
END

TO CIRCLEL :RADIUS
ARCL1 .174532 * :RADIUS 36
END
```

```
TO CIRCLER :RADIUS
ARCR1 .174532 * :RADIUS 36
END
```

Comments: These arc and circle procedures actually draw a 36-sided polygon. (The number .174532 is the result of computing 2 * π / 36; π is rounded to 3.1416; and 36 is the number of sides of the polygon.)

READWORD or RW

```
TO READWORD
OP FIRST READLIST
END

TO RW
OP READWORD
END
```



Acknowledgments

This Guide owes a lot to the children who originated and debugged the programming examples used here. Many of these children are now grown up. We thank them still.

All of the staff of Logo Computer Systems, Inc. have contributed in various ways to the production of this guide.

This Guide was significantly improved by the helpful comments of Seymour Papert.

Margaret Minsky also played an important role in the preparation of this Guide.

Discussions with Gary Drescher on Logo and its new implementation were invaluable in its conceptual development.

Jim Davis, Brian Silverman, Greg Gargarian, Ed Hardebeck, Steve Hain, Tom Polucci, Billy Weinreb, Erric Solomon, Glenn Forrester, Florence Buxton-Thomas, Nancy Smith, Larry Davidson, Alison Birch, Kiyoko Montpetit, Bob Lawler, John Berlow, Andy diSessa, Brian Harvey, and Puff have all been important to this presentation. Special thanks to Peter Cann, who made the plotter images, and to Annette Dula, who spent many hours editing these pages. Judy Richland and Nancy Gardner, our production and graphics design consultants, have been spectacular in turning this manual into a beautiful book.

The manuscript was prepared on a text processing system put together by Max Behensky. It uses Mark of the Unicorn software so that we could have an EMACS-like text editing system.

The detailed definition of our Logo — how it functions and how it is presented to the user — was evolved collectively by our staff including Seymour Papert.

The implementation of the language, based on a model written in LISP by Gary Drescher and Ed Hardebeck, was designed and written by Jim Davis, Steve Hain, Ed Hardebeck, and Brian Silverman. The screen text editor was written by Tom Polucci.

Index

Logo Computer Systems, Inc.

9960 Cote de Liesse
Lachine, Quebec
Canada H8T 1A1