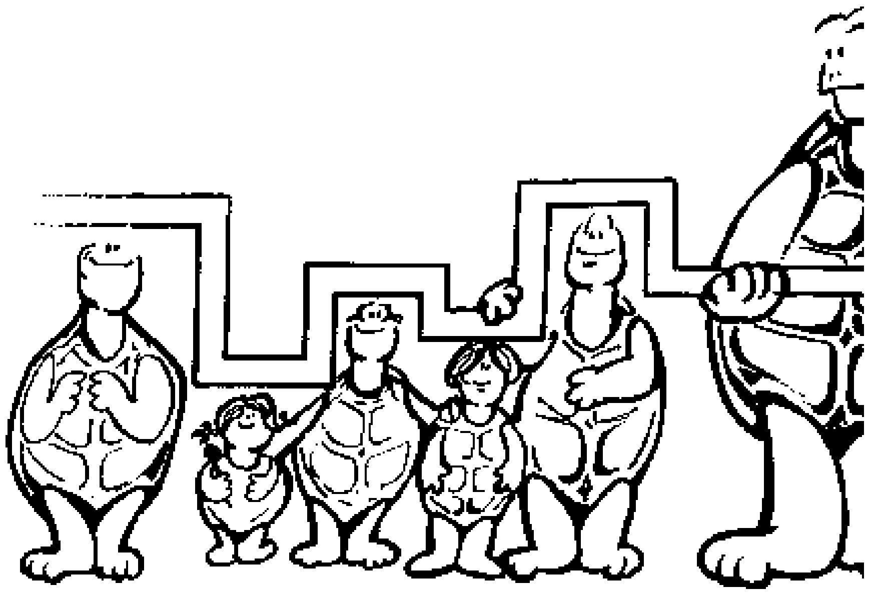
Chapter 6. Varying Variables

How many people are in your family? In your class? Are they all the same? Or are they **variable**?



Got one in on you, didn't I? There's that word, "Variable."

How many of your friends have the same color hair? The same color eyes? How many are the same age? How many were born in the same month as you? Anyone born on the same day?

How many things about you and your group are the same? How many are "variable?"

Variables in Logo

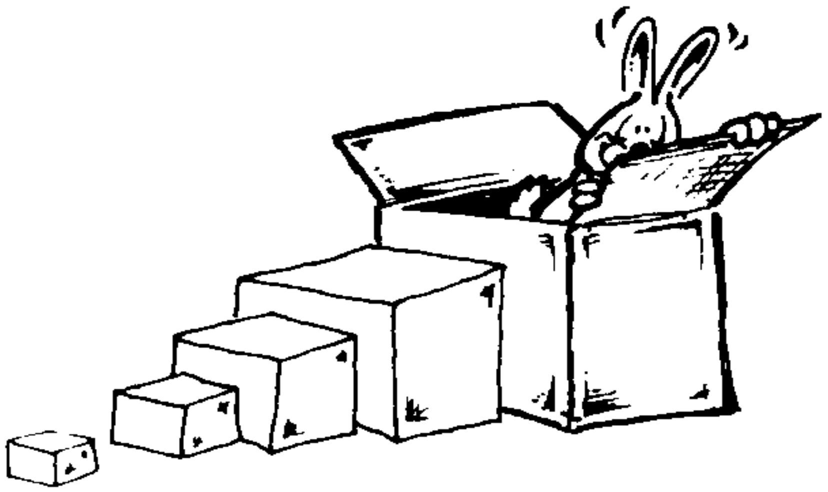
Now let’s talk about Logo.

By now, you should know what this procedure will look like after it's been run. What do you think?

```
TO BOXES
REPEAT 4 [FD 100 RT 90]
RT 90 PU FD 120 PD LT 90
REPEAT 4 [FD 100 RT 90]
END
```

Sure, that's a procedure to draw two boxes side by side.

But what if you wanted to draw 20 boxes? What if you want each box to be bigger than the last?



What if you want them smaller? In other words, what if you want to vary the size or the number of boxes?

No problem! This is where those things called “variables” come in. A variable is something you put into a procedure so you can change the procedure every time you run it.

Yes, that does sound confusing, doesn't it?

To help explain it, let's take another look at the experiment you did creating pictures using just one shape. Find a big sheet of paper and draw a picture using your favorite shape. Use triangles, squares, or rectangles — or even circles, if you've peeked ahead in this book.

Remember, you can only use one type of shape. But you can vary the size of the shape all you want.

There's that word again, "vary."

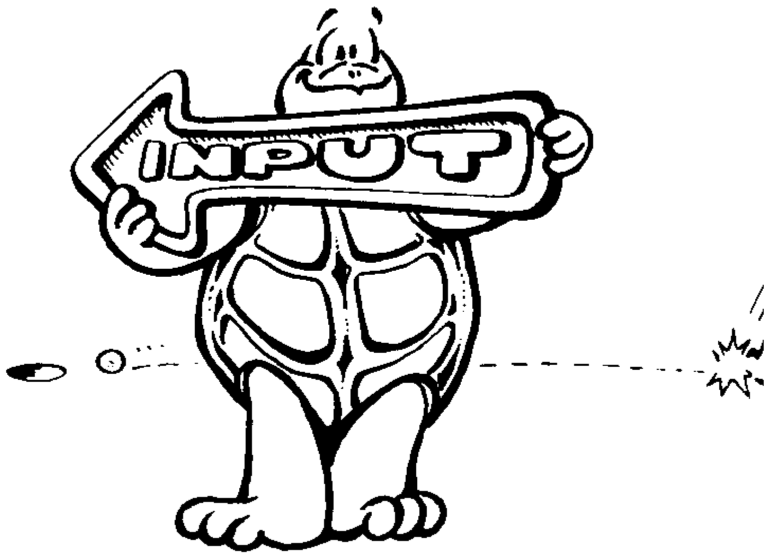
Remember the caterpillar example? That's a picture drawn using just squares (and a little piece of a straight line). And don't forget the cat.

OK, got your drawing done? Before you try to put your picture on the computer, let's take a look at the new BOXES procedure below. It may give you some ideas.

```
TO BOXES :SIZE
REPEAT 4 [FD :SIZE RT 90]
RT 90 PU FD :SIZE + 20 PD LT 90
REPEAT 4 [FD :SIZE RT 90]
END
```

You probably know what the variable is, don't you? It's the :SIZE. That's right.

Now when you type BOXES to run the procedure, you have to provide something new, an input.



Try it out. Type...

BOXES 20

BOXES 40

BOXES 60

BOXES 100

When you type BOXES 20, you tell the :SIZE variable to use the :SIZE of 20. What about BOXES 60. What will :SIZE be then?

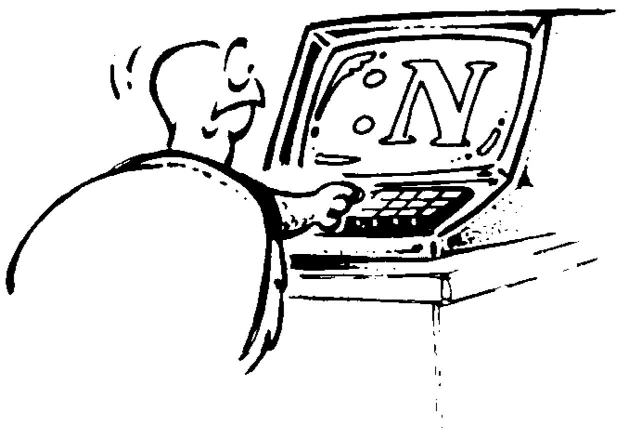
Variables must always have an input, or value. And they must also have the two dots in front so Logo knows it's a variable.

Yes, that's a colon. But in Logo, we call them "dots." You'll find they can save you a lot of time and typing.

Take a look. Remember the TRI procedure? Let's add a variable.

```
TO TRI :N
REPEAT 3 [FD :N RT 120]
END
```

See! You can name variables just about anything you want. Rather than call this one :SIZE, call it :N. The :N can stand for number. Of course, you could call it :X, :Z, or :WHATEVER.



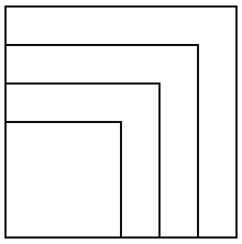
But you still have to use the dots.

Here are some examples that a 7-year-old had fun dreaming up. They use this SQUARE procedure.

```
TO SQUARE :N
REPEAT 4 [FD :N RT 90]
END
```

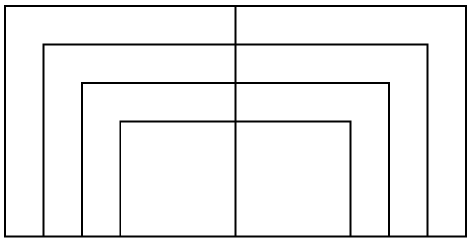
It started as a simple exercise to see what different squares would look like.

```
TO SQUARES
SQUARE 60
SQUARE 80
SQUARE 100
SQUARE 120
END
```



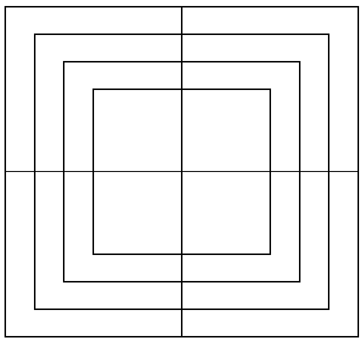
Then she added a left turn, and that reminded her of her mom’s stacking tables.

```
TO TABLES
SQUARES
LT 90
SQUARES
END
```



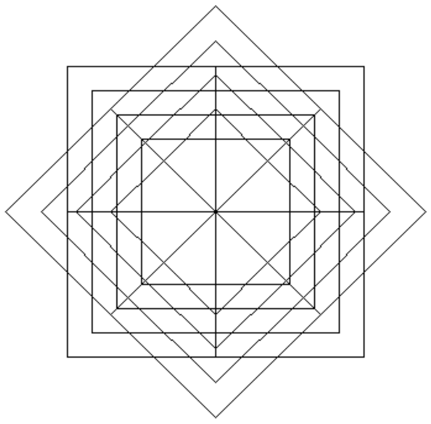
The more she looked at the tables, the more it looked like half of a decorative mirror.

```
TO MIRROR
TABLES
LT 90
TABLES
END
```



And what would happen if you stacked mirrors?

```
TO MIRRORS
MIRROR
LT 45
MIRROR
END
```

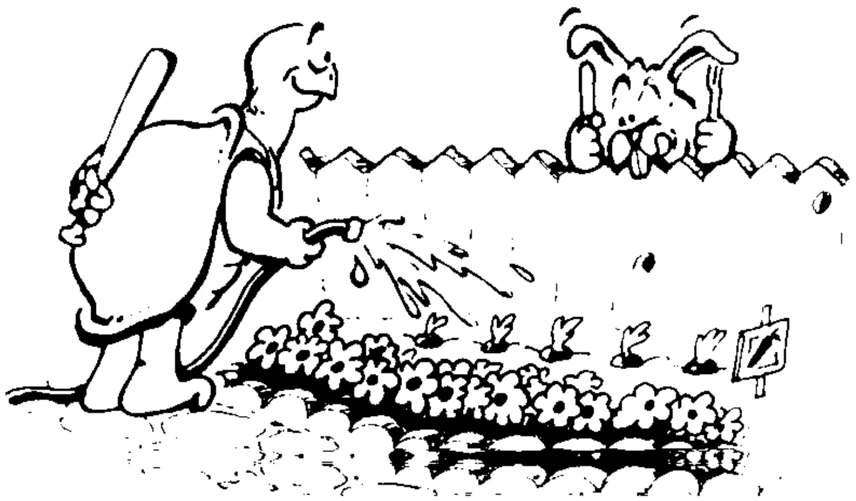


This is a lot to think about. So why not stop for a while and experiment using one shape in a design.

After you've had fun with one shape, try doing something with two shapes.

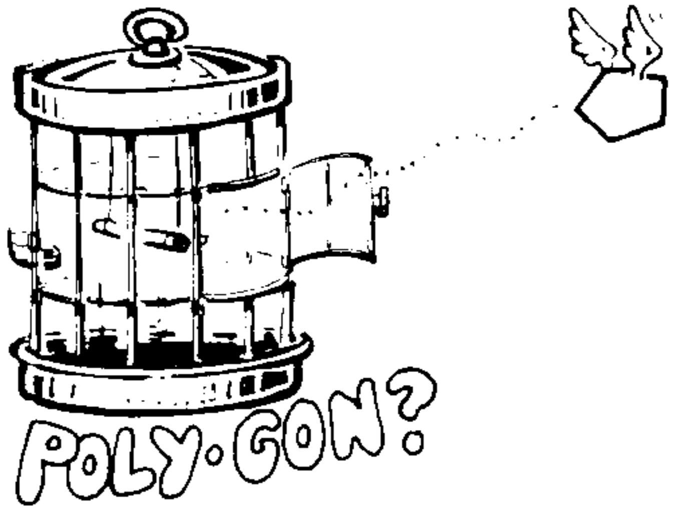
You've already seen what you can do with a square and a triangle. These were combined to make a house. Then they were used to make a wheel.

Since you've also made some flowers, maybe you can "plant another garden?"



Polygons and Things

Polygon? Now there's a new word for you. Do you know what it means? No, it doesn't mean that Poly flew away.



We'll talk lots more about polygons. But for now, think about this for a moment.

Squares, triangles, and rectangles are polygons. So are pentagons, hexagons, and octagons.

All of these shapes have one thing in common. They all enclose an area that has at least three sides. (You can't enclose anything with two sides, can you?)

Triangles have three sides, squares and rectangles have four, pentagons have five, and octagons have eight. Seems like there's a rule for polygons in there somewhere.

A polygon is a closed shape with at least three sides.

Rabbit Trail 16. Variable String Toss

Here's something else to explore. How about trying a variation of the game String Toss? It's called FD :N. (We're sneaking the variables in here, too.) The idea is to create a design by passing the ball of string back and forth. The :N variable can equal one step or as many as you want.

Let's say you want to create a square of string. That's really easy. One person plays the Turtle starting at Home. The Turtle holds one end of the string, gives the ball of string to the first person, and says FD :N times 5. The first person takes 5 steps.

The first person then turns RT 90, holds the string to make a corner, and gives the ball of string to the second person. That person goes FD :N * 5 and RT 90. A third person takes the ball of string and goes FD :N * 5 RT 90. And finally a fourth person takes the string and brings it HOME.

See how this works? The string is now in the shape of a variable square. Now try a hexagon, why don't you?

Then maybe you can connect six triangles to make a fancy hexagon.

It's more fun when you make crazy shapes. Try it.

If you find it hard to see the shapes, have everyone carefully put the string on the floor and then step back. Can you see the shape now?

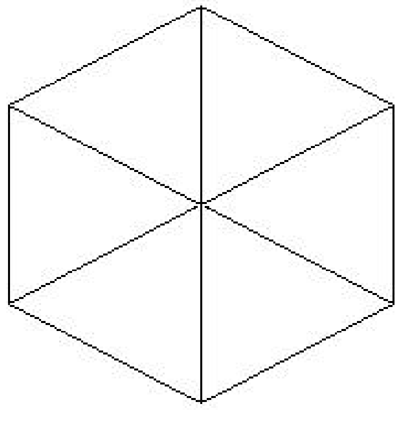
Hexagons and Spiderwebs

To make that String Toss Game design on the computer, you can use the TRI :N procedure you wrote earlier in this chapter.

```
TO TRI :N
REPEAT 3 [FD :N RT 120]
END
```

What would happen if you repeated the TRI :N procedure, turning after each triangle?

```
REPEAT 6 [TRI :N RT 60]
```



What do you call a shape that has six sides like this? That's a hexagon, right?

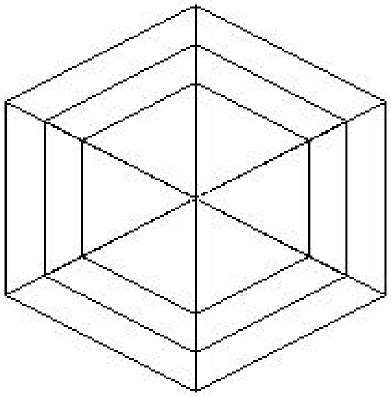
Hmmmm? That sort of looks like a see-through box — one of those optical illusions.

But back to hexagons for now.

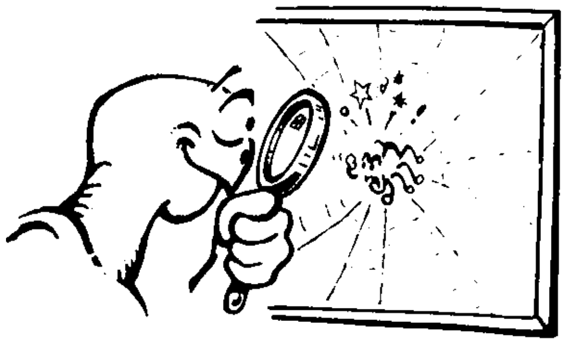
```
TO HEXAGON :N
REPEAT 6 [TRI :N RT 60]
END
```

Be sure to tell the turtle how big to make the hexagon. Try this:

```
HEXAGON 60
HEXAGON 80
HEXAGON 100
```



What does this look like? Of course, it's a spiderweb!



Can you think of another way to write this procedure so that the turtle will do the same thing? How about this!

```
TO SPIDERWEB :N
HEXAGON :N
HEXAGON :N + 20
HEXAGON :N + 40
END
```

Go ahead. Type the SPIDERWEB :N procedure and then try

```
SPIDERWEB 40
```

Play around with this idea to see what it can do. Make up some other shape procedures using variables.

Adding More Variables

Can you think of a way to use more variables in the SPIDERWEB procedure? What about substituting a variable for 10? For 20? For both?

```
TO SPIDERWEB :N :X :Y
HEXAGON :N
HEXAGON :N + :X
HEXAGON :N + :X * :Y
END
```

This is getting complicated.

:N gives you the size of each side.
:X tells you how much to add to :N
:Y tells you to multiply :X by this number

After you've typed in this procedure, see what happens when you try

```
SPIDERWEB 60 20 2
```

Does this look like the first spiderweb the turtle drew? It should. Let's change the variables to numbers and take a look.

```
TO SPIDERWEB 60 20 2
HEXAGON 60
HEXAGON 60 + 20
HEXAGON 60 + 20 * 2
END
```

Changing a Variable

Typing SPIDERWEB 60 20 2 is fine when you want to make three hexagons that have sides of 60, 80, and 100. But what if you want to do five hexagons? Seven hexagons? Seventy hexagons?

Let's try something! When you write a procedure, it becomes another command you can use, right?

OK. Then let's make the most of it. Tell SPIDERWEB to draw a hexagon using the variable :N. Then tell SPIDERWEB to add 10 to itself and do the same thing again.

```
TO SPIDERWEB :N
HEXAGON :N
SPIDERWEB :N + 10
END
```

Try it! What happens?

Wait a minute!

The last line of the SPIDERWEB procedure has the procedure using itself. That's strange!

No, that isn't strange, that's recursion. There's a whole chapter on what you can do with recursion. For now, let's just stick with the variables.

Local and Global Variables

Most versions of Logo use two types of variables: local and global. Global variables are used by any procedure. Take a look.

```
TO SHAPES :N
TRI :N
```

```
SQUARE :N
RECTANGLE :N
END
```

How about it? Can you write procedures for a triangle, a square, and a rectangle using :N to represent the distance forward.

```
TO TRI :N
REPEAT 3 [FD :N RT 120]
END

TO SQUARE :N
REPEAT 4 [FD :N RT 90]
END

TO RECTANGLE :N
REPEAT 2 [FD :N RT 90 FD :N * 2 RT 90]
END
```

Now, when you type SHAPES 100, each of the procedures will use 100 wherever there is an :N. The :N is a global variable. It's available to anyone who wants to use it.

Global variables tend to be a nuisance. Logo has to keep track of which procedures uses which global variable, what the value of the variable is, has it changed? This takes up valuable memory.

Of course, sometimes you have to use global variables. But it keep things neater if you can use local variables.

Local variables are "local" to the one procedure where it is used. So there isn't nearly as much record-keeping required, making it easier on Logo.

You write them like this:

```
TO TRI
LOCAL "X
MAKE "X 100
REPEAT 3 [FD :X RT 120]
END
```

Hey, there's a new command, MAKE. You'll learn more about that on the next page. In the meantime, go ahead. Change your TRI procedure. Change the TRI :N in the SHAPES procedure to just TRI. Now run the SHAPES procedure using SHAPES 100 again. What does the picture look like? Why?

You'll see lots more examples of local variables as you move through the rest of this book.

Outputting Variables

OK, local variables are good. Global variables are not so good. Is there another way to pass information between procedures without using global variables?

Sure is!

You can OUTPUT them. You remember, OUTPUT sends information to another procedure. Let's use the TRI procedure as an example. Here's what you need to do.

```
TO TRI
REPEAT 3 [FD X RT 120]
END

TO X
OUTPUT 100
```

```
END
```

In this example, X isn't really a variable. It's a procedure. So how would you add a local variable to this so that X would pass information to TRI?

How about this?

```
TO X
LOCAL "Z
MAKE "Z READWORD
OUTPUT :Z
END
```

Is this really the best way to run the TRI procedure? Of course not. The important lesson here is

Don't ever close your mind to new possibilities!

In other words, never say "Never."

Before we go, let's confuse the issue even more. Actually, here's a way to simplify the X procedure above. You can combine LOCAL and MAKE using the LOCALMAKE command.

```
TO X
LOCALMAKE "Z READWORD
OUTPUT :Z
END
```

Keep this in mind as you continue.

Making Variables

LOCAL "X is easy enough to figure out in the TRI procedure. But what's with the MAKE "X 100?

MAKE is a command that gives a value to the variable named "name." The name of a variable must always be what Logo sees as a word. That means it can be a letter, such as :X, or a word, such as :VARIABLE. Here's how it works.

```
MAKE "<name> <value>
```

In the TRI example, the goal was to MAKE the variable X have the value of 100. Then you can use the variable :X within that procedure whenever you want something to be equal to (have the value of) 100. In the TRI procedure, the variable :N was used as the side of the triangle, which in this case is 100 turtle steps long.

More Ways to Make Variables

You just got introduced to MAKE. Well, Logo gives you lots of other ways to vary your variables. Let's start with another look at MAKE.

```
MAKE "JOE 2
MAKE "TOM 4
MAKE "SAM :JOE + :TOM
```

So what does :SAM equal? If you said six, you get a Gold Star.

You can also NAME :JOE + :TOM "SAM

This does the same thing as MAKE "SAM :JOE + :TOM except that you NAME *<value>* "*<name>*.

If you want to see what :SAM equals, you can tell the computer to

```
PRINT :SAM
or
SHOW :SAM
```

You can also tell Logo to

```
SHOW THING "SAM
or
PRINT THING "SAM
```

THING does the same thing as the dots. It outputs the value of the variable named in the word that follows THING. Sure, that sounds confusing. Try it a few times and it will begin to make sense. That's why Morf likes to experiment so much.

Conditional Things

Remember the SPIDERWEB procedure?

```
TO SPIDERWEB :N
HEXAGON :N
SPIDERWEB :N + 10
END
```

The problem with this procedure is that it just keeps running, filling your screen with spiderwebs. Is there no way to stop it other than pressing the HALT button?

Well, there is a way. You just tell the turtle that IF the last hexagon that it drew was as big as you want the spiderweb to be, THEN stop drawing.

Here's how you use IF. Since IF knows what you mean, you don't have to use the word THEN.

```
TO SPIDERWEB :N
IF :N > 100  [STOP]
HEXAGON :N
SPIDERWEB :N + 10
END
```

Look at that first line in this new procedure. When the turtle reads this line, it learns that IF :N is greater than 100, then stop drawing.

Greater Than, Less Than

That thing that looks like an arrowhead after the :N > is the symbol for "greater than." It means that if the value of :N is greater than 100, then STOP.

If > means "greater than," what does that other arrow symbol [<] mean?

You guessed it. It means "less than." An easy way to remember which symbol is which is that the arrow always points to the smaller value.

- IF :N > 100 means that the value of :N must be larger than 100, at least 101.
- IF :N < 100 means that the value of :N must be less than 100, no more than 99.

For our example, we picked 100 as a place to stop. You can select your own stopping point. Or you can make the stopping point another variable. How would you do that?

Go ahead. Give it a try. But remember, if you're going to use a variable like this, you have to add it to the procedure name.

```
TO SPIDERWEB :N ____
IF :N > ____ [STOP]
HEXAGON :N
SPIDERWEB :N + 10  ____
END
```

"OK, I understand IF. IF something is true, then Logo will carry out the next instruction. And that sits inside brackets. But what if that something is not true? What if I want Ernestine to do something if the answer is false?

TEST

Actually, there are two ways to handle that. Look at how SPIDERWEB has been changed below.

```
TO SPIDERWEB :N
TEST :N > 100
IFTRUE [CS CT PR [SORRY!] STOP]
IFFALSE [HEX :N]
SPIDERWEB :N + 10  ____
END
```

The first lines says to test :N to see if it is greater than 100. The next line says that if the test is true, clear the screen, clear the text, print SORRY!, and stop. The third line says that if :N is not greater than 100, go ahead and run HEX :N. (HEX :N is a new short name for HEXAGON :N.)

What do you think would happen if you left out IFFALSE? Then you'd have

```
TEST :N > 100
IFTRUE [CS CT PR [SORRY!] STOP]
HEX :N
```

Would that work? Try it and see. What did you learn from that?

You don't always have to have both IFTRUE (IFT for short) and IFFALSE (IFF for short) in your procedures.

IFELSE

Another way is to use the IFELSE command. Let's change the SPIDERWEB procedures and try it out.

```
TO SPIDERWEB :N
IFELSE :N > 100 [CS CT PR [SORRY!]STOP][HEX :N]
SPIDERWEB :N
END
```

The first line says that if :N is greater than 100

- clear the screen
- clear the text
- print SORRY!
- Stop

If :N isn't greater than 100, run the HEX procedure and move on the next line. You can think of IFELSE as

IF a condition is true, **THEN** do this or **ELSE** do this. Actually, this is just what some Logo packages let you do. For example:

```
TO L.OR.R
PR [SHOULD THE TURTLE GO LEFT OR RIGHT?]
IF RC = "L THEN LT 90 FD 100 ELSE RT 90 FD 100
END
```

You can also write the IF line as

```
IF RC = "L [LT 90 FD 100][RT 90 FD 100]
```

This works just fine in some versions of Logo — but not in MSW Logo. You need IFELSE.

Go ahead and explore. You'll see more of IFELSE.

When you've finished with spiderwebs, why not add variables to your procedures for drawing other shapes? See what you can do with squares, rectangles and things.

Remember, this is your own Great Logo Adventure!

More on Tessellations

Tessellations are really great places to use variables. These repeating patterns usually start with a basic shape that is repeated in varying sizes.

Do you remember the tessellation from Chapter 4 that used Diamonds? This gets a bit tricky so think this one through carefully. Can you combine DIAMOND, DIAMOND1, and DIAMOND2 to make one procedure using variables? How would this change the other procedures?

Here are the Diamond procedures.

```
TO DIAMOND
REPEAT 2 [FD 8 RT 60 FD 8 RT 120]
END

TO DIAMOND1
REPEAT 2 [FD 24 RT 60 FD 24 RT 120]
END

TO DIAMOND2
REPEAT 2 [FD 40 RT 60 FD 40 RT 120]
END
```

Look at the distances the turtle moves. Can you write one procedure for these that uses a distance variable?

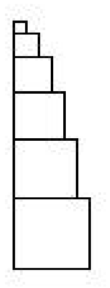
```
TO DIAMOND :DIST
REPEAT 2 [FD :DIST RT 60 FD :DIST RT 120]
END
```

Now, rather than use DIAMOND, DIAMOND1, or DIAMOND2, you can use DIAMOND 8, DIAMOND 24, or DIAMOND 40.

More Fun With Squares

Let's try a tessellation with squares. The first thing to do is draw a tower of squares, each square smaller than the last.

```
TO SQUARES :S
IF :S < 0 [ STOP]
REPEAT 4 [FD :S RT 90]
FD :S
SQUARES :S - 5
END
```

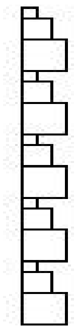


Try SQUARES now using different inputs. This is going to be the basic pattern in the tessellation. The picture above was made using 30 as the input to SQUARES.

Next, let’s make a TOWER of SQUARES.

TOWER takes two inputs: one that says how big the SQUARES are, and the second to tell the turtle how many times to repeat the SQUARES pattern.

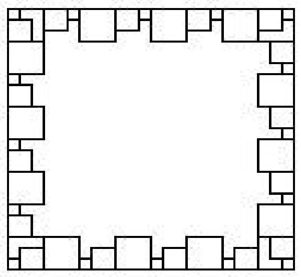
```
TO TOWER :S :T
IF :T = 0 [STOP]
SQUARES :S
TOWER :S :T - 1
END
```



Here’s the pattern made by using TOWER 15 5.

And this raises a question. Are you just going to make a tall, skinny tessellation? Or can you make the TOWER procedure turn the corner, maybe like a picture frame?

```
TO FRAME
PU LT 90 FD 100 RT 90 BK 40 PD
REPEAT 4 [TOWER 15 4 RT 90]
END
```



The first thing the FRAME procedure does is move the turtle over to the left. Then it draws the FRAME using the REPEAT command.

REPEAT 4 [TOWER 15 4 RT 90]

Now we're getting some where. Try different inputs. TOWER 15 4 seems to work pretty good.

To make this into an interesting tessellation, why not just fill up a frame with the SQUARES pattern? How are you going to do that?

Rabbit Trail 17. Tessellating Squares

Here's a quick and easy Rabbit Trail for you. It's a great way to discover what you can do with your SQUARES pattern. You can either use squares of different sizes or better yet, print a page full of the SQUARES pattern and cut them out.

Now move the patterns around to see what kind of patterns you can make.

Can you make the FRAME pattern using squares or your cutouts?

Once you figure that one out, then figure out what the turtle would have to do to fill the FRAME pattern after it draws the first TOWER pattern?

More Towers

When the turtle draws TOWER 15 4, it can’t just turn around a draw the pattern again, can it? What would happen? Why not try it and see?

After the turtle gets to the top of the first pattern, it is going to have to move over a bit to draw the TOWER pattern coming down the screen. But how far?

You know that the pattern is :S steps wide. In TOWER 15 4, :S is 15, right? So let’s write a procedure for the turtle to move at the top of the TOWER.

```
TO MOVE1
RT 90 FD _____ RT 90
END
```

When the turtle gets to the top of the TOWER, she’ll turn right, move over, and then turn right again. What happens if we use the value of :S or 15? Does that work?

No. The turtle ends up drawing the pattern over the original drawing. When the turtle turns at the top, it starts drawing the SQUARES pattern by moving to the right. This means the turtle has to move twice as far, or :S * 2.

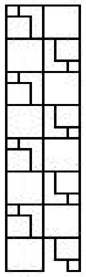
Try it. See what happens.

```
TO MOVE1 :S
RT 90 FD :S * 2 RT 90
END
```

Try this:

```
TOWER 15 4
MOVE1 15
```

It seems to work, doesn't it!



You're not out of the woods yet. Do you see that blank space at the bottom — to the right?

How are you going to fill that in? Also, what is the turtle going to have to do to draw the next TOWER?

How about this?

```
TO MOVE2 :S
RT 90 FD :S BK :S RT 90
END
```

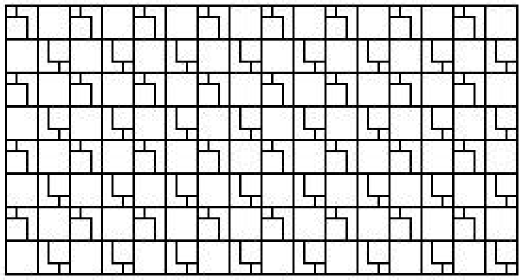
The turtle turns right, fills in the gap, backs up, turns right again (that's 180 degrees), and is ready to start again.

You didn't know this was going to be this complicated, did you?

There's one more
procedure that will cr

We'll call it COV

```
TO COVER :S :T :X
```



```
IF :X = 0 [STOP]
TOWER :S :T
MOVE1 :S
TOWER :S :T
MOVE2 :S
COVER :S :T :X - 1
END
```

You already know what the :S and :T variables are. What about the :X?

That's easy enough. Just like the :T variable, :X tells COVER the number of times to repeat itself.

Musical Variables

In the last chapter, we talked about making music. Now that you've read about variables, how about some musical variables?

Do you want to turn your keyboard into musical keys? Here's one way to do it.

```
TO MUSIC
MAKE "KEY RC
IF :KEY = "C [SOUND [262 100]]
IF :KEY = "D [SOUND [294 100]]
IF :KEY = "E [SOUND [330 100]]
IF :KEY = "F [SOUND [349 100]]
IF :KEY = "G [SOUND [392 100]]
IF :KEY = "A [SOUND [440 100]]
IF :KEY = "B [SOUND [494 100]]
IF :KEY = "S [STOP]
```

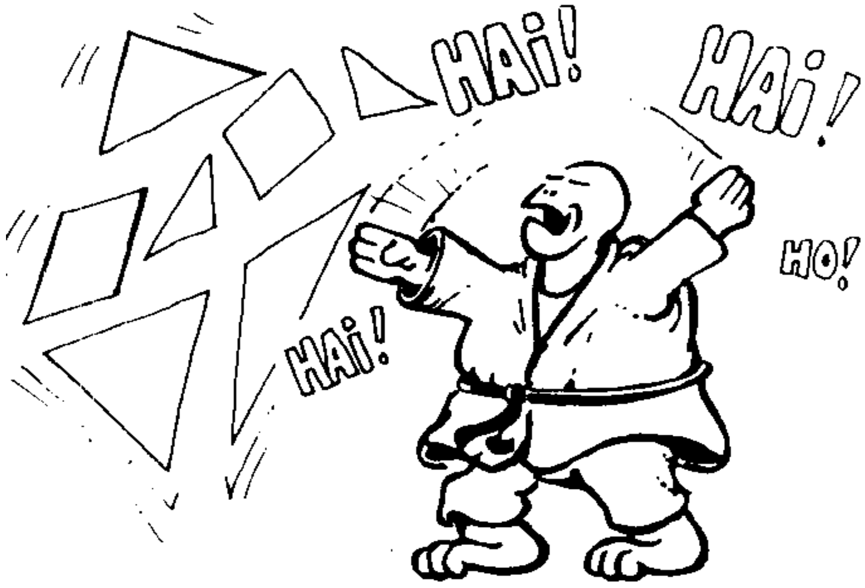
```
MUSIC
END
```

There's another new command, RC. That's short for READCHAR. When Logo sees the READCHAR or RC command, it stops and waits for you to type a character. In this case, the letter you type becomes the variable :KEY.

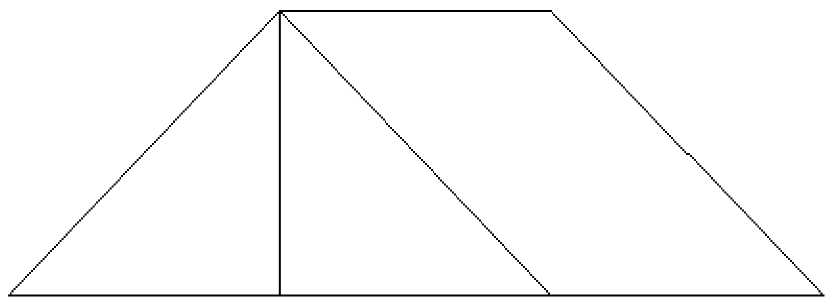
If you type one of the keys — A, B, C, D, E, F, G — you hear a note. Just make sure you use a capital letter. Otherwise Logo just runs the MUSIC procedure again and again until you hit one of the sound keys and press Enter.

Rabbit Trail 18. Tangrams

The Tangram is an Oriental puzzle with seven shapes of different sizes.

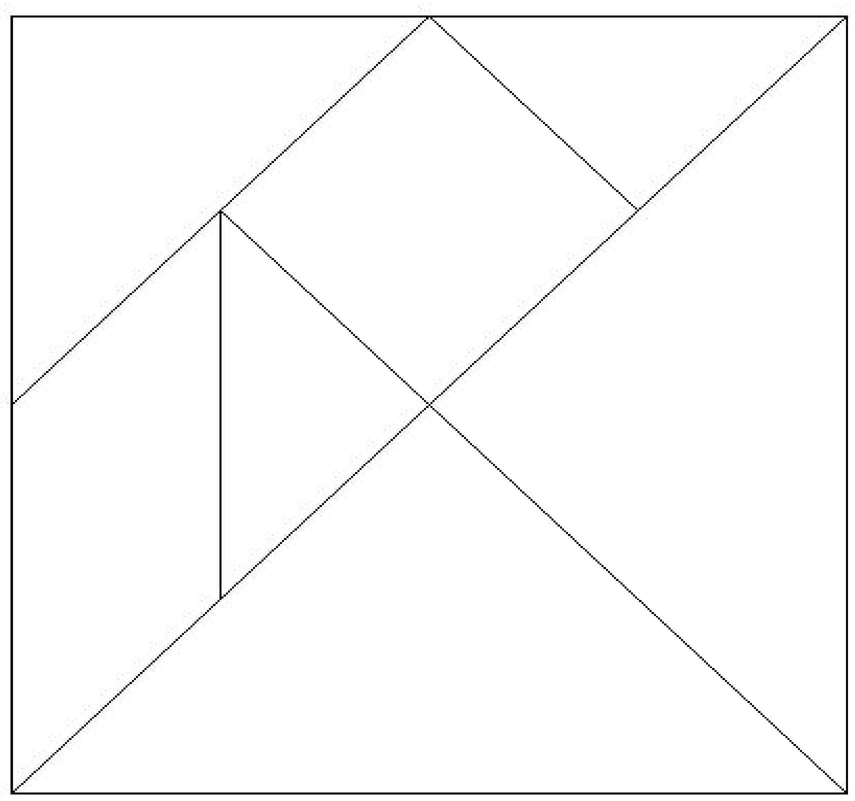


The puzzle is to use these shapes to make lots of different things. Here's my pup tent.

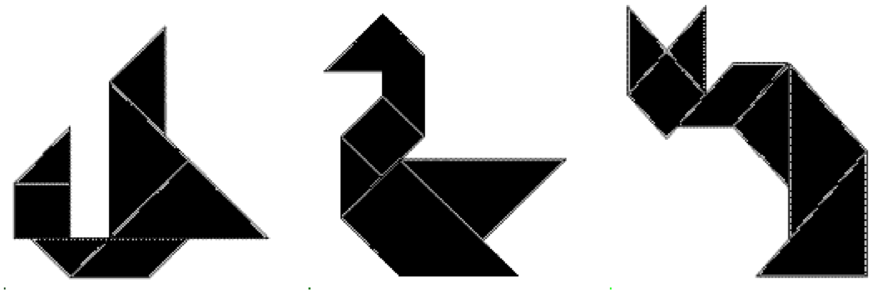


Why not visit your local library or bookstore? You'll find there are a number of books on tangrams that will give you lots of ideas of what to do with your new puzzle pieces.

There's a PCX file on the CD that came with this book called TANGRAM.PCX.



1. Print the picture and paste it to a piece of cardboard.
2. Carefully cut out the pieces.
3. Now you can play with the pieces to create interesting shapes: birds, ships, dragons, and other interesting designs. Here's some to get you started.



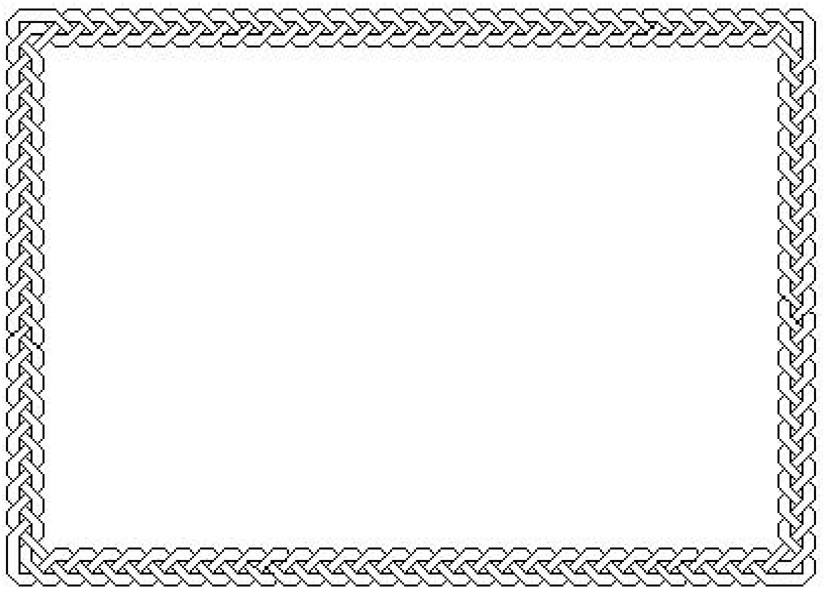
4. Now draw them on the computer.

There's a procedure on the CD that came with this book called TANGRAM.LGO. You can use that to create your Tangram shapes. We talk about it in *The Great math Adventure* chapter.

Now, why not see what you can do with Tangrams?

Adding Borders

Morf just loves to put borders around things, even the graphics window. Take a look!



Joe Power, a friend from California, taught Morf how to do that. It comes in real handy when you want to do a pretty card or announcement.

Here's the procedure.

```
TO BORDER
CS HT
PU LT 90 FD 200 LT 90 FD 100 LT 180 PD
BRAID
END
```

You can change the change this procedure to make the border larger or smaller. You also have to change the last line of the BRAID procedure.

```
TO BRAID
MAKE "SQR2 1.4          ;Square root of 2
MAKE "HFSQ2 0.7         ;Half the square root of 2
MAKE "S2 8.5            ;Square root of 2 * 6
MAKE "H2 4.2            ;(Square root of (2 * 0.5)) * 6
MAKE "S2H2 12.7         ; :S2 + :H2
PU FD 24 RT 45 FD 4.2 SETH 0 PD
REPEAT 2 [STRIP 20 CORNER STRIP 30 CORNER]
END
```

What’s that stuff out to the side?

Adding Comments

Those are comments. Programmers usually "annotate" their code, or programs. That means that they leave explanations written in their programs so that users will know what the program or procedure is supposed to do. In this case, the notes tell you what the variables mean.

You can add notes to your MSW Logo procedures by typing a semicolon followed by your notes. Logo doesn’t pay any attention to anything that follows the semicolon. If your version of Logo does not recognize the semicolon, use this procedure. It does the same thing.

```
TO ; :comment
END
```

As to the "square roots" in the comments, don’t worry about them right now. You’ll get into them in *The Great Math Adventure* chapter. You’ve got enough to think about just trying to figure out what the BORDER procedure is doing.

Varying the Border

To change the size of the BORDER, change the number of times that STRIP is repeated. Change it from STRIP 20 to STRIP 15, for example. Go ahead. Give it a try.

```
TO CORNER
LT 45 FD :H2 RT 45 FD 6
```

```
RT 45 FD :S2 RT 45 FD 18
RT 45 FD :S2H2 PU
RT 90 FD :H2 PD RT 90 FD :S2
LT 45 FD 18 LT 90 FD 6 PU
LT 45 FD :S2 PD LT 90 FD 17 PU
RT 90 FD :H2 PD RT 90 FD 17 PU
RT 45 FD 6 RT 90 FD 12 PD
RT 45 FD :H2 RT 45 FD 6
RT 45 FD :H2 PU RT 90 FD :H2 PD
RT 45 FD 6 PU BK 15 RT 90 FD 9 RT 90 PD
END

TO START
; Here's a simple procedure that puts a braided border
; around the edge of the screen.  Morf likes frames
; for his pictures.
; You can change the size of the border by changing the
; variable used by STRIP in the BRAID procedure.
BORDER
END

TO STRIP :N
REPEAT :N ~
  [
  LT 45 FD :H2 RT 45 FD 6 RT 45 FD :S2H2
  PU RT 90 FD :H2 PD RT 90 FD :S2 LT 45 FD 6 PU
  LT 45 FD :S2H2 PD LT 135 RT 45 FD :H2 LT 45
  FD 6 LT 45 FD :S2H2 PU LT 90 FD :H2 PD LT 90
  FD :S2 RT 45 FD 6 PU RT 135 FD :S2H2 RT 45
  FD 6 PD
  ]
END
```

Using the Tilde

The Strip procedure is actually one long line. But look how it's written.

```
REPEAT :N ~
```

What's that symbol after :N?

It's a tilde. In MSW Logo, that means that the instruction list is continued on the next line. There you find a single bracket:

```
[
```

When you have long lines and lists inside other lists, they can get confusing — very difficult to read. MSW Logo gives you some help. When MSW Logo sees a single bracket like that, it knows to look on the next line for the rest of the list.

The rest of the line in STRIP is simply a long list of commands. But what if you had lists within lists. Here's a simple example.

```
TO HEX
REPEAT 6 ~
  [
    REPEAT 3 ~
          [
            FD 100 RT 120
          ]
    RT 60
  ]
END
```

This is the same as

```
TO HEX
REPEAT 6 [REPEAT 3 [FD 100 RT 120] RT 60]
END
```

When procedures begin to get long and complex, you need a system that allows you to read and understand what's going on. As you will see in coming chapters, this can come in real handy.

Check out the procedures in the MSW Logo "Examples" directory for some other examples of multi-line procedures.

From Two to Three Dimensions

"Morf, do you remember Jamie, the six-year-old from that kindergarten class we worked with a few years ago?"

"The name's familiar. What did she do?"

"She was the one who told that newspaper reporter that she was smarter than the computer — because she could roller skate!"

Jamie was among the children at a private school near Dallas who enjoyed learning with Logo on and off the computer. What made her kindergarten class so special was the way they quickly and easily learned to visualize the differences between their three-dimensional world and Logo's two dimensional world.

challenge for you. Draw pattern of a soccer ball
n.

The first thing you see, looking at a soccer ball, is a bunch of hexagon shapes.

When some 3rd and 4th grade computer club members were asked to draw this pattern on the screen, they thought it would be easy.

```
TO SOCCER.BALL :DIS
REPEAT 6 [REPEAT 6 [FD :DIS RT 60] FD :DIS LT 60]
END
```

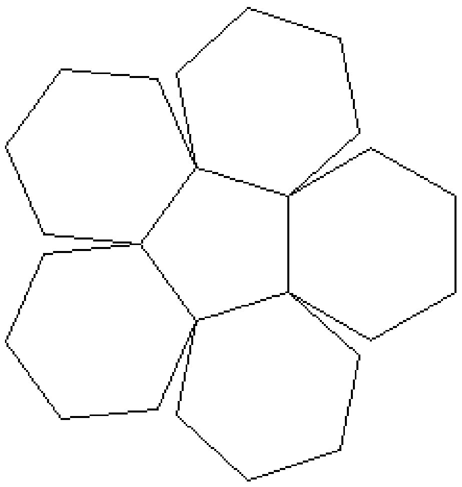
The boy’s team thought that all they had to do was draw a series of hexagons. But the center was a pentagon, not a hexagon. So their procedure didn’t quite work, did it.

The girl's team was the first to figure out that they could not make the soccer pattern on the screen as it appears on the ball. They had to flatten it out. At first, they thought this procedure was wrong. But then they discovered it was really correct.

```
TO SOCCER :DIS
REPEAT 5 [REPEAT 6 [FD :DIS RT 60] FD :DIS LT 72]
END
```

The girls printed twelve of their patterns, colored them, cut them out, taped them together, and made their own soccer ball. When they were finished, they decided it made a better pinata.

So they filled it with candy and had a party.

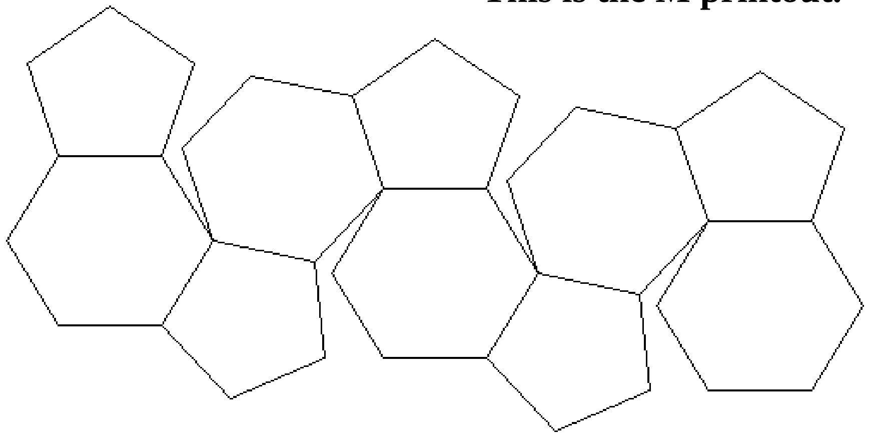


Adam's Soccer Ball

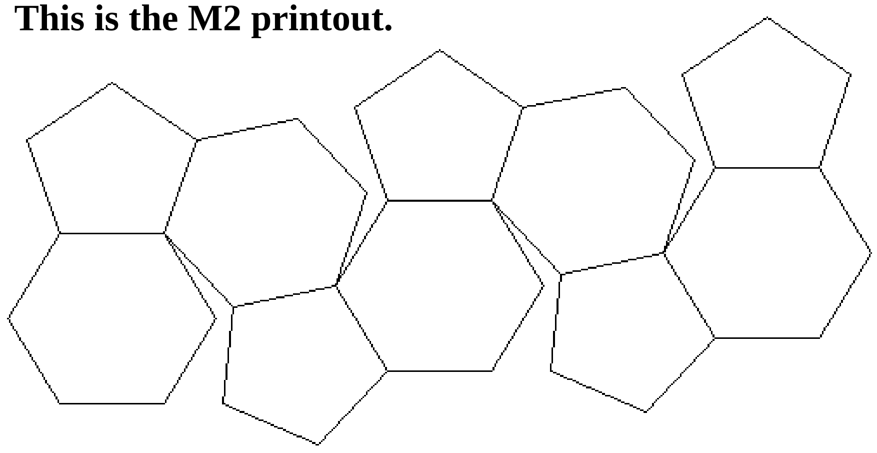
One young man decided to see if he could produce the entire soccer ball pattern in just one printout. Two was the best he could do.

Here's a picture of Adam's soccer ball. The procedure is on the CD that came with this book as SOCCERM.LGO. Maybe you can figure out a way do it all at once.

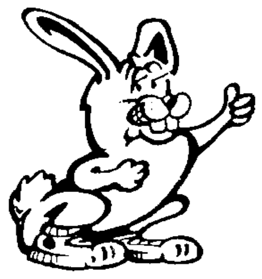
This is the M printout.



This is the M2 printout.



Rabbit Trail 19. Folded Paper Fun



Making the soccer ball out of paper is just one of many things you can do with Logo and folded paper. The computer club that made the first flattened soccer ball pattern found that you can make all sorts of three dimensional objects from folded paper.

How about a simple cube? This takes you from the two dimensional square to a three dimensional cube.

```
TO CUBE  :D
CS HT
REPEAT 4 [SQUARE :D RT 90 FD :D LT 90]
PU HOME REPEAT 2 [RT 90 FD :D] RT 180 PD
REPEAT 3 [SQUARE :D FD :D]
END

TO SQUARE  :D
REPEAT 4 [FD :D RT 90]
END
```

The group first cut out a number of cardboard squares. Then they taped them together to see what kind of shapes they could make. The next step was to transfer the pattern to the computer.

Making 3-D shapes from triangles really got interesting

```
TO TETRAHEDRON  :D
RT 30 TRI :D MOVER :D TRI :D
MOVEL :D TRI :D
END

TO MOVER  :D
RT 60 FD :D LT 60
END

TO MOVEL  :D
LT 60 FD :D RT 60
END

TO TRIR  :D
RT 60 FD :D TRI :D
END

TO TRI  :D
REPEAT 3 [FD :D RT 120]
END

TO OCTAHEDRON  :D
LT 30 TRI :D RT 30 TETRAHEDRON :D
LT 60 TRI :D TRIR :D TRIF :D
END

TO TRIF  :D
```

```
FD :D RT 60 TRI :D
END
```

TETRAHEDRON and OCTAHEDRON are just the beginning of what you can do with Logo and a printer.

Go ahead. Try these. Print them. Fold them up. And then design your own 3-D figures.

The whole idea is to explore, to discover what you don’t know and then go find the answers.

FOR

"No, Morf, this isn’t a golf match. FOR is a new command to explore. It can be a big help sometimes. Here, watch what this one-liner does."

```
FOR [N 0 2200] [FD 3 RT (:N * :N)]
```

That’s not nearly as bad as it looks. There’s just a bunch of stuff to remember. Maybe it would help to look at the procedure below. It does the same thing.

```
TO CRAZY.CIRC :N
IF :N > 2200 [STOP]
FD 3 RT (:N * :N)
CRAZY.CIRC :N + 1
END
```

Here’s another look at it as a different kind of one-liner.

```
MAKE "N 0 REPEAT 2200 [FD 3 RT (:N * :N) ~
   MAKE "N :N+1]
```

This tells you exactly how it works.

N is the name of the variable used in the crazy circle.
0 is the starting value of :N
2200 is the final value of :N
[FD 3 RT (:N * :N)] The list of instructions to carry out.

In MSW Logo, FOR looks for two lists. The first list "sets the rules" for what's supposed to happen. The second is a list of what is going to happen.

Logo looks for a word as the first element in the first list. Yes, N is a word in Logo even though it's only one letter. The rest of the list includes two or three numbers.

The first number is the starting value for the variable, N. The second number is the final value for :N. There can be a third number that tells Logo how to count from the first value to the final value of the variable. Normally Logo will count by 1 as it did in CRAZY.CIRCLE. How about this one:

```
FOR [N 0 100 5] [SHOW :N]
```

In this case, Logo counts by five. This line says:

For the variable :N, start at 0 and go to 100, making each step 5. Now show (or print) :N.

Here's some other examples to play with. These came from an on-line contest to find the prettiest one-liner.

```
FOR [X 1 150] [FD :X RT 89]
FOR [I 0.01 4 0.05] [REPEAT 180 [FD :I RT 1]]
FOR [X 10 200] [SETPENSIZE SE :X :X ~
   REPEAT 36 [FD 20 RT 15]]
```

DEFINE Your Procedures

Speaking of oneliners, here's another way to define procedures and variables. Use the DEFINE command.

```
DEFINE "SQUARE [[SIDE] [FD :SIDE RT 90]]
```

Try it. You'll see that this line is the same as:

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

Keep in mind that DEFINE does what it says it's going to do: define a procedure. It doesn't run the procedure. You have to tell Logo to do that.

The nice thing about it is that DEFINE can be used as a command within another Logo procedure, whereas TO requires you to use the Mode window or the editor.

The first thing that DEFINE looks for is a word that says what the name of the procedure is to be. In this case, the name of the procedure is SQUARE. Next, DEFINE looks for a list that includes any variable inputs followed by lists of instructions. Each line of instructions is put inside brackets.

You don't have to use variables to use the DEFINE command. Both of these examples work just fine.

```
DEFINE "SQUARE [[][REPEAT 4 [FD 50 RT 90]]]
DEFINE "HELLO [[][PR "|I'M LOGY!|][PR "|I'M MORF!|]]
```

Remember the SHAPES procedure? Here's another way to write the shapes procedures using DEFINE within a superprocedure.

```
TO SHAPES
DEFINE "SQUARE [[][REPEAT 4 [FD 50 RT 90]]]
DEFINE "TRI [[][REPEAT 3 [FD 50 RT 120]]]
DEFINE "REC [[][REPEAT 2 [FD 50 RT 90 FD 100 ~
    RT 90]]]
SQUARE TRI REC
END
```

Here's one to have some fun with:

```
DEFINE "FRAC [[N] [IF :N > 1 [FRAC :N RT 60 ~
    FRAC :N FD :N]]
```

is the same as defining this procedure like this:

```
TO FRAC :N
IF :N > 1 [FRAC :N * .6 RT 60 FRAC :N  *.6 FD :N]
END
```

Now that's weird! You've got a procedure calling itself — not just once, but twice. That's recursion, which is discussed in the *Recursion* chapter.

This little monstrosity is a fractal procedure. To really understand what's going on, you'll have read the *Recursion* chapter and *The Great Math Adventure* chapter. In the meantime, why not have some fun with it.

Try FRAC 100

Try different numbers to see what it does. Then change .6 to another number, like .7 or .4. Change RT 60 to RT 90 or RT 72. What happens?

To really understand this procedure, you're going to have to read the recursion and math adventure chapters.

Copying Definitions

Hey, do you want to play a trick on your parents? Maybe on your teacher? I just love playing tricks on Logy.

COPYDEF and REDEFP are commands that let you rename your own procedures as well as your Logo primitives. Don't worry, these new names are not saved. And while these commands can be useful at times, they sure can be fun. They use variables, too!

Let's start with COPYDEF. This one's easy.

```
COPYDEF "FRACTAL "FRAC
```

This copies the new name FRACTAL to the old name FRAC.

Now type

```
EDIT "FRACTAL
```

You get

```
TO FRAC :N
IF :N > 1 [FRAC :N * .6 RT 60 FRAC :N  *.6 FD :N]
END
```

What happened to the new name FRACTAL? Actually, it's buried, something you'll read about in the next section.

Some versions of Logo copy the whole procedure with the new name. Then you'll get:

```
TO FRACTAL :N
IF :N > 1 [FRAC :N * .6 RT 60 FRAC :N  *.6 FD :N]
END
```

This shows you that you have to be careful using COPYDEF. What would happen if you erased the FRAC procedure?

You'd be in big trouble, that's what. So what good is this new command?

Suppose you wanted to run a procedure that uses the SETPOS command but your version uses the SETXY command. One of the ways to get around this difference is to simply type

```
COPYDEF "SETPOS "SETXY
```

Now you've got a SETPOS command that acts the same as the SETXY command. Get the idea?

Redefining Primitives

Now that you know how to copy a new name to a procedure, let's try it on a Logo primitive. To do that, you have to make REDEFP true. That means to turn it on. Here's how:

```
MAKE "REDEFP "TRUE
```

Now you can go ahead and change the Logo primitives.

```
ERASE "FD
```

Now try FD 100. What happens? You get

I don't know how to FD

Now try this one.

```
COPYDEF "FD "BK
FD 100
```

Remember, you COPYDEF <*new name*><*old name*>. So what happened? Does this give you any ideas?

____

Bury and Unbury

When you COPYDEF a new name over an old name, the old name stays around just as you saw above. The new name gets buried.

BURY is one of those Logo primitives that is often ignored. But it can be very useful.

Let's try something.

1. Load any procedure.
2. Type BURYALL and press Enter.
3. Type POALL and press Enter.

   Where'd the procedures go?

4. Try to run the buried procedure. What happened?
5. Now load another procedure.
6. Type UNBURYALL and press Enter.
7. Type EDALL and press Enter.

Both the procedures are now visible in the Editor, aren't they?

What this means is that when you bury something, it moves from your workspace into another part of the computer’s memory. It’s like it’s buried!

Why not bury the color procedures from the last chapter? First load the color procedures. Then type

```
BURY "COLORS
```

If you type POTS, nothing is displayed, right? Now type

```
SETSC BLACK
```

The screen color turns black.

You don’t have to remember color numbers anymore. Use the names.

If you ever want to see what’s buried, just say

```
UNBURY "COLORS or

UNBURYALL
```

This "digs up" everything that’s buried.

Planting Another Garden

Early in this chapter, you had the chance to "plant another garden." Before you leave this chapter on variables, how about planting another garden by adding a twist to the Anyshape procedure. This also adds a twist to running procedures automatically and shows you something else about variables.

In the FLOWERS procedure, you run procedures from within another procedure. Take a look.

```
TO FLOWERS :REPEATS :LIST
REPEAT :REPEATS ~
   [RUN :LIST RT 360 / :REPEATS ]
END
```

RUN is a command that tells Logo to run a list of commands. Your remember what a list is, don't you? The GARDEN procedure gives you a pretty good idea. Lists can contain words, commands, or other lists.

Take a look at the first line of GARDENS. After you clean the screen, you have FLOWERS 5. That means that the :REPEATS variable has a value of 5. Then you have a list [FD 50].

```
TO GARDEN
CS FLOWERS 5 [FD 50] WAIT 60
CS FLOWERS 5 [FD 60 SHAPE 50 5] WAIT 60
CS FLOWERS 5 [FD 50 LT 30] WAIT 60
CS FLOWERS 7 ~
       [FD 50 LT 60 FD 50 RT 120 FD 50 LT 60 FD 50]
WAIT 60 CS FLOWERS 8 [SHAPE 100 5] WAIT 60
CS FLOWERS 8 [SHAPE 100 3] WAIT 60
CS FLOWERS 8 [SHAPE 100 4] WAIT 60
CS FLOWERS 8 [SHAPE 80 6] WAIT 60
CS FLOWERS 5 ~
           [FD 80 FLOWERS 8 [SHAPE 80 3] BK 80]
END

TO SHAPE :SIZE :REPEATS
REPEAT :REPEATS [FD :SIZE RT 360 / :REPEATS]
END
```

Waiting

Do you remember when we mentioned "waiting" before? There are times that you want to slow down the computer so you can see what's going on, or when you just want it to wait a few seconds. That's where the WAIT command comes in.

WAIT <*time in 60ths of a second*>

There's another way to slow the computer down or to have it take a pause. Write your own WAIT command. Because WAIT is a primitive already, call your new procedure TIME or TIMER.

```
TO TIMER :T
IF :T = 0 [STOP]
TIME :T - 1
END
```

You can make this procedure as precise a timer as you need because you can make :T whatever you want. After all, it is a variable. You can also change :T - 1 to :T - 0.25 or whatever. It's another way to get Logo to do exactly what you want it to do. You can get it to wait in hundreths or even thousandsth of a second.

"What if I just want to pause for a moment while running a procedure? Can I do that?"

Sure, you can. That's what the Pause button is for; the one over to the right in the Commander Box. Try it and see what happens.

Last Minute Ideas

The GARDEN procedure is OK. But have you ever seen a black and white garden? Try adding some color to it.

GARDEN shows you a number of individual flower shapes. Maybe you want to change those shapes. Or maybe you want them to stay on the screen for a longer time. Add a WAIT command.

Remember the last FLOWER picture that is displayed?

```
FLOWERS 5 [FD 80 FLOWERS 8 [SHAPE 80 3] BK 80]
```

Why not add some variations of this to the GARDEN procedure so you have different groups of flowers in your garden. Here's one idea:

```
FLOWERS 12 [SHAPE 30 8]
```

Also, why not have your flower garden "grow" when it loads.

```
Make "startup [GARDEN]
```

Whatever you do, have fun with your new garden.
