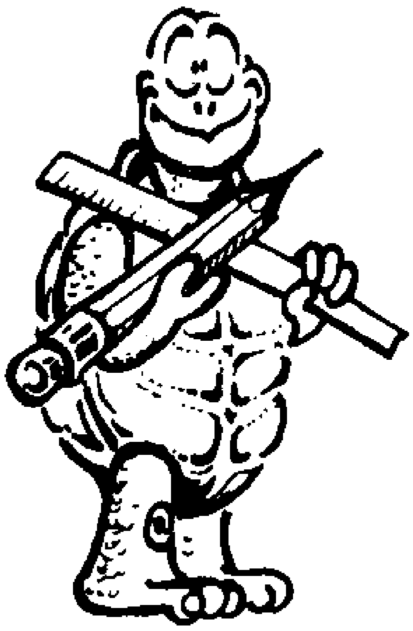
Chapter 10. The Great Math Adventure

"When I'm doing my arithmetic homework, it seems more like a pain in the neck than a Great Math Adventure."



True enough! But think about this for a moment! What part of your life does not involve mathematics, some form of calculation, counting, or measurement?

What about time, the counting and measurement of seconds and hours? Or distance, the measurement of space in inches and feet, or centimeters and meters? What about music, measured in frequencies?

Think about Logo and the computer for a moment. Everything you do on the computer is translated into electrical signals that become a mathematical code of zeros and ones.

Everything you have been doing so far in our Great Logo Adventure has been part of turtle geometry, right? I hate to tell you this, but that's mathematics.

Just what is mathematics? Is it just some number tables that you have to memorize? Is it just a bunch of formulas and equations?

Or is it a way to express ideas and relationships using common symbols such as +, -, *, /, and others? That sounds like a language, doesn't it?

Isn't it sort of like Logo? A way to explore and express ideas on and off the computer? What you read on the screen is expressed by numbers, by a mathematical code that is translated into electrical signals.

If you had to communicate with people who don't speak English, how would you do it? What if they were aliens who didn't speak at all?

One way to start is through the use of numbers, music, and mathematics.

Counting is something everyone does in one way or another. Musical tones all have numerical frequencies as you may remember. Sounds set up vibrations. So even if you can't hear the sounds, you can feel them. Believe it or not, that's how some deaf people enjoy music.

To carry this idea a bit further, humpback whales sing. The use of mathematics is helping scientists analyze their songs.

So look beyond your homework into all the realms where numbers and math are used. Who knows? Maybe by the time you finish this book, you may just agree that math *is* a universal language.

Come on! Let's start with arithmetic.

Logo Arithmetic

Remember back in the first Rabbit Trail where you took a Turtle Walk. You used the multiplication symbol then.

```
FD 10 * 5
```

What would happen if that line said

```
FD 10 + 5  or
```

```
    FD 10 - 5  or

    FD 10 / 5
```

Try these commands using variables.

```
    MAKE "A 50
    MAKE "B 100
    MAKE "C :A + :B
    FD :A + :B  or FD 100 + 50

    FD :C / :A * 10
```

That’s FD 50 + 100 divided by 50, or 3. 3 * 10 = 30 or FD 30.

How about FD :C / (:A * 3)

Does this instruction do the same thing? Why? Or why not?

When Logo looks at a command that uses arithmetic, it does the arithmetic in the standard mathematical order: multiplication and division followed by addition and subtraction.

So, when Logo reads that line, the first thing it sees is FD :C or FD 150. This is divided by 50 * 3 or 150. So you have FD 150 / 150 or FD 1.   Looks like the parentheses change things.

Parentheses are among the Logo delimiters that can be used to change the order of operations.

“Delimiters. That’s a funny word!”

It may seem a bit strange. But when you think about it, that's just what they do. Delimiters <u>*de*</u>fine the <u>*limit*</u>*s* of an operation. Take a look.

The commands listed below use arithmetic signs to tell the turtle to go FD 200.

```
FD 100 + 1000 / 10
FD 10 * (5 + 15)
FD (20 - 10) * (18 + 2)
```

Write down some other commands that will take the turtle FD 200. Make sure you use parentheses in your examples. Then test them out.

____

Positive and Negative Numbers

When you add or multiply two positive numbers together, what do you get? You get another positive number. That makes sense, doesn't it?

Add a positive and a negative number together. What do you get? Why not try it and see.

```
SHOW 10 + (-2)
8

SHOW 10 + (-12)
-2
```

What about multiplication?

```
SHOW 10 * -2
```

```
-20

SHOW -10 * -2
20
```

Adding and Multiplying Negative Numbers

How can that be? Multiply two negative numbers and the answer is positive?

This is something worth talking about. How can two negatives make a positive? One way to look at the subject of positive and negative numbers is to see how the turtle handles such things. Check out the coordinate system again, especially the GRID.LGO procedure you saw in Chapter 8.

```
FD 50        That's easy. The turtle moves (+) 50.
BK 50        The turtle moves back (-) 50.
```

What happens when you ask the turtle to go

```
BK -50
```

Rabbit Trail 25. Positive and Negative Numbers

Here's another way to make some sense out of positive and negative numbers. Make a game out of them or just make up a demonstration. You'll need:

- some scissors.
- sheets of paper in at least two colors.
- markers to write on sheets of paper.

You're going to play a game of high finance.

Cut out a bunch of paper strips — at least five of each color for each player — about the size of a dollar bill. Next you need to mark the strips for what they are worth. There are four colors in the example shown below. Two colors will work also.

Color	***Description***	***Value***
White	Asset	+1
Red	Debt	- 1
Yellow	Asset	+5
Green	Debt	- 5

Assets are things that people own. They have a positive value. Debts have a negative value; like when you borrow money from the bank to buy a home or a car.

Now you're ready to start your game or demonstration. You can call it "Wall Street Trader," where you will buy and sell companies. Maybe you can play "Banker" where players trade for things in their own small town. Make up your own rules; as simple or as complicated as you want. The whole idea is to see what happens when you use positive and negative numbers.

Here's how it works:

3 * 5	I give you three of my yellow assets.
3 * -1	I give you three of my red debts.
-2 * 5	I take two of your yellow assets.
-3 * -5	I take three of your green debts.

After you have made several trades, how are you doing? Subtract your debts from your assets. Do you have more than when you started or less?

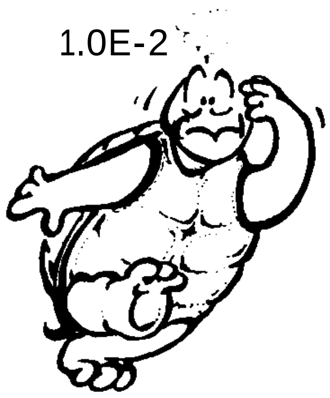
More important, does this help make sense out of positive and negative numbers?

____

Engineering Notation

"You're getting complicated again. What's with this notation stuff?"

Engineering notation — some people call it exponential notation — is simply a way of writing very big numbers.



Decimal numbers include a part of a whole number, such as 1.25, 3.24, 89.23. Logo lets you use decimals such as

```
FD 100.125  BK 21.75
```

Logo writes very big and very small numbers using engineering notation such as 1.0E-21.

COOKIE.LGO is a simple game that adds some fun to mathematics. It's a great test to see who can make the most money selling cookies. The full procedure is on the CD that came with this book. Only a few of the subprocedures are listed here. You'll need to look at the whole thing to understand what's going on.

Right now, let's just focus on that strange stuff in the Cost procedures:

```
TO COSTA
OUTPUT 1.E-2*(19 + RANDOM 7)
END

TO COSTB
```

```
OUTPUT 1.E-2*(12 + RANDOM 8)
END

TO COSTC
OUTPUT 1.E-2*(9 + RANDOM 7)
END
```

Engineering notation looks strange all scrunched together like that. But it's not all that complicated. It's really pretty easy.

Time to experiment.

Trying playing around with some engineering numbers.

```
SHOW 1.E+2*9
900

SHOW 1.E+5*9
900000

SHOW 1.E+14*128
1.28E+16
```

What kind of answer is 1.28E+16?

If you play around with engineering notation, you'll discover how it works. Try adding lots of numbers to 1.E. Subtract a bunch also. What happens?

1.28E+16 is 128 with 14 zeros, sixteen places to the right of the decimal point.

What's SHOW 1.E-14 * 128?

Mathematics doesn't have to be dull, meaningless stuff. It can be fun. It can even get exciting! It can even get weird!

Mathematical Operations

There are lots of other ways you can use arithmetic with Logo. Here are the math commands used in MSW Logo.

SUM	DIFFERENCE	MINUS
PRODUCT	QUOTIENT	REMAINDER
INT	ROUND	SQRT
POWER	EXP	LOG10
LN	SIN	RADSIN
COS	RADCOS	ARCTAN
RADARCTAN		

Let's take a look at some examples:

```
FD SUM 50 50
```

What do you think that means? You're right — FD 100. Forward the sum of 50 and 50 or 50 + 50. How about

```
FD DIFFERENCE 300 200
```

Forward the difference between 300 and 200 or 300 - 200.

```
FD PRODUCT 10 10
```

Forward the product of 10 times 10 or 10 * 10.

```
FD QUOTIENT 1000 10
```

Forward the quotient of 1000 divided by 10 or 1000 / 10.

```
FD REMAINDER 1000 300
```

Forward the remainder of 1000 divided by 300. How much is that?

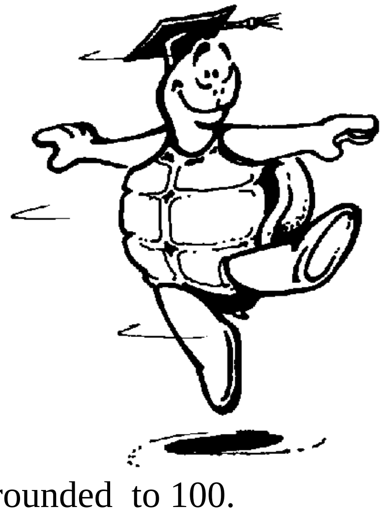
MSW Logo uses INT for integer. But what's an integer? That's a whole number, one without decimals or fractions.

```
FD INT (121.8 - 21.1)
```

Forward the integer of 121.8 - 21.1. 121.8 - 21.1 equals 100.7. But, since the command is FD INTeger, or whole number, the decimal is dropped and you're left with FD 100.

```
FD ROUND 121.8 - 21.5
```

Forward 121.8 - 21.5 rounded off. That equals 100.3, which rounds to 100.



Change that to

```
FD ROUND 121.8 - 22.1
```

That equals 99.7, which is rounded to 100.

All these examples would be much simpler if they just said FD 100. After all the arithmetic is done, they each tell Ernestine, the turtle, to go FD 100.

So what?

Well, what if you want to add or multiply a bunch of variables?

```
FD SUM (PRODUCT :A :X)(QUOTIENT :B :Y)
REPEAT (PRODUCT :A :B) [FD SUM :C :D RT 90]
```

You'll see some examples of this type of thing later on.

Factorials

Factorials give you the chance to explore recursion again as well as multiplication. A factorial is the product of all the whole numbers from 1 to whatever.

For example, factorial 5 is just another way of saying
5 * 4 * 3 * 2 * 1 which equals 120

To write that as a procedure:

```
TO FACTORIAL :N
IF :N = 1 [OUPUT :N]
OUPUT :N * (FACTORIAL :N - 1 )
END
```

The easiest way to make sense of this is to use the recursive pages approach like you did with AMAZE in the last chapter.

When you type FACTORIAL 5, this is what Logo sees.

```
TO FACTORIAL 5
IF 5 = 1 [OUPUT 5]
OUPUT 5 * (FACTORIAL 5 - 1 )
END
```

This is saved on the first page as the procedure is called again in the third line: OUPUT 5 * (FACTORIAL 5 - 1). This time, it reads

```
TO FACTORIAL 4
IF 4 = 1 [OUPUT 4]
OUPUT 4 * (FACTORIAL 4 - 1 )
END
```

This continues until the second line of the procedure reads

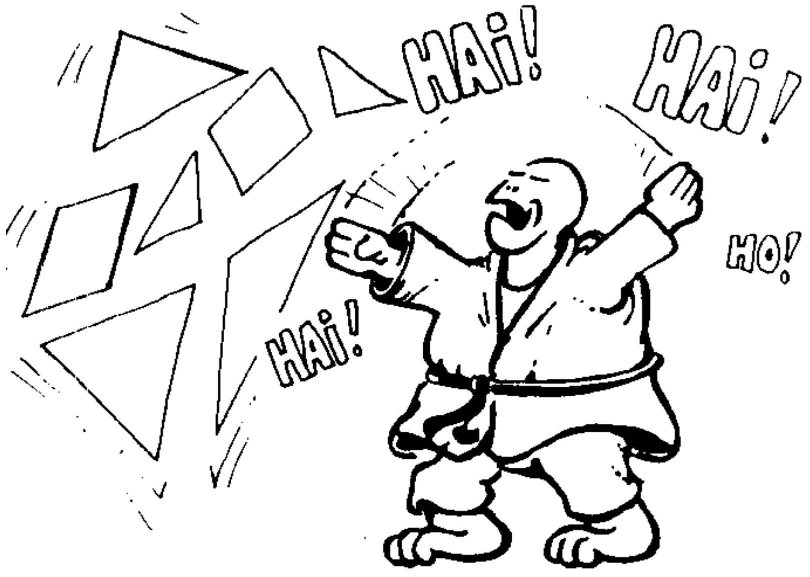
```
IF 1 = 1 [OUTPUT 1]
```

Then Logo reads back through the pages:

5 * 4 = 20 * 3 = 60 * 2 = 120 * 1 = 120

The Tangram Procedures

Now let's take a closer look at something else you've seen before. Do you remember the Tangram puzzles? Well, let's take a look at the procedures to draw the Tangram pieces you saw earlier. They're on the CD that came with this book in the \procs\chpt10 subdirectory.



```
TO TRIANGLE.RT :SIDE
TO TRIANGLE.LT  :SIDE
TO SQUARE.LT  :SIDE
TO SQUARE.RT  :SIDE
TO MED.TRI.RT  :SIDE
TO MED.TRI.LT  :SIDE
TO SMALL.TRI.RT  :SIDE
TO SMALL.TRI.LT  :SIDE
TO PARGRAM.LT  :SIDE
TO PARGRAM.RT  :SIDE
```

To give you an idea of what you can do with tangram shapes, try this procedure.

```
TO TANGRAM  :SIDE
SETH 90 TRIANGLE.LT :SIDE
FD :SIDE SETH 0
TRIANGLE.LT :SIDE
FD :SIDE SETH 270
```

```
SMALL.TRI.LT :SIDE
FD :SIDE / 2 SETH 225
MED.TRI.RT :SIDE
SQUARE.LT :SIDE FD :SIDE1
PARGRAM.LT :SIDE
END
```

This procedure uses the different shape procedures to make one big shape. Which one?

You'll have to run the procedure to see.

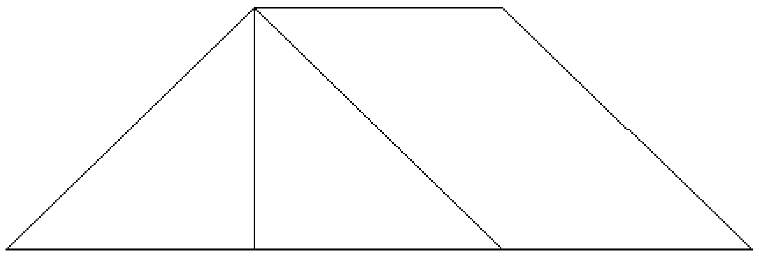
Fun With Tangrams

Enough of this stuff. Let's have some more fun with tangrams! Put two small triangles together. What shape do you get? Can you make a square from the two small triangles? How about a larger triangle? A parallelogram?

Put the parallelogram and two small triangles together. What shape is that? Can you make a square? What about a trapezoid?

"A what?"

"A Trap-e-zoid! That's another shape, Morf. It's like a parallelogram but it only has one set of parallel sides instead of two."



"That's no trap-e-whatever. That's a picture of the pup tent we use out in the back yard!"

"Get serious, Morf! Can you make a triangle using five pieces of the puzzle?

Check out the CD that came with this book for more ideas on what to do with tangrams.

Making Crazy Shapes

Why not have the computer think up some shapes for you?

These might come out a bit crazy. But who cares? That's the fun of having the turtle do things for you. That's why CRAZY.SHAPES was included in the TANGRAM.LGO procedure.

```
TO CRAZY.SHAPES  :SIDE
SHAPES :SIDE
MOVE :SIDE
CRAZY.SHAPES :SIDE
END

TO SHAPES :SIDE
MAKE "SHAPE RANDOM 10
IF :SHAPE = 0 [TRIANGLE.RT :SIDE]
IF :SHAPE = 1 [TRIANGLE.LT :SIDE]
IF :SHAPE = 2 [MED.TRI.RT :SIDE]
IF :SHAPE = 3 [MED.TRI.LT :SIDE]
IF :SHAPE = 4 [SMALL.TRI.RT :SIDE]
IF :SHAPE = 5 [SMALL.TRI.LT :SIDE]
IF :SHAPE = 6 [SQUARE.RT :SIDE]
IF :SHAPE = 7 [SQUARE.LT :SIDE]
IF :SHAPE = 8 [PARGRAM.RT :SIDE]
IF :SHAPE = 9 [PARGRAM.LT :SIDE]
END

TO MOVE :SIDE
MAKE "MOVE RANDOM 5
```

```
IF :MOVE = 0 [SETH HEADING + 45]
IF :MOVE = 1 [SETH HEADING + 90]
IF :MOVE = 2 [FD :SIDE]
IF :MOVE = 3 [FD :SIDE / 2]
IF :MOVE = 4 [FD (:SIDE / SQRT 2 ) / 2]
END
```

SHAPE :SIDE and MOVE :SIDE offer a very simple method for selecting something at random. Other methods are coming up.

RANDOM, Picking, and Shuffling

There's that RANDOM command again. It takes one input. Also, do you remember that in MSW Logo, RANDOM selects a number between zero and the number you input.

There may be times when you want to randomly select a sequence of numbers and then use that same sequence over again. Ordinarily the sequence of random numbers is different each time Logo is used. However, if you need the same sequence repeatedly, type RERANDOM and a seed number before you use the RANDOM primitive. For example:

```
(RERANDOM 1234)
REPEAT 2 [SHOW RANDOM 10]
9
3
```

When you need different sets of random numbers, you can give RERANDOM a new seed number as input. The seed number is simply a number that identifies the sequence you're working with. For example:

```
(RERANDOM 4321)
REPEAT 2 [SHOW RANDOM 10]
6
```

2

If you want to change the sequence, simply change the seed number.

There are times that you want to randomly rearrange a list, like a sequence of numbers, letters, cities, or whatever — like shuffling a deck. To do that, you have to give RANDOM a little help.

Here's a procedure to do just that. It has some new commands that you'll get into a bit later. But since we're talking about RANDOM and RERANDOM, go ahead and try it out now. You'll find lots of uses for it.

```
TO SHUFFLE :DECK
MAKE "X []
REPEAT COUNT :DECK ~
   [CHECK MAKE "DECK BUTFIRST :DECK]
REPEAT (RANDOM 4) ~
   [MAKE "X LPUT FIRST :X BUTFIRST :X]
OP :X
END

TO CHECK
IFELSE (RANDOM 3) = 1 ~
  [MAKE "X FPUT FIRST :DECK :X] ~
  [MAKE "X LPUT FIRST :DECK :X]
END
```

Why not see what happens when you deal a shuffled list of shapes?

```
TO DEAL
CS
MAKE "LIST SHUFFLE [SQ TRI HEX]
RUN :LIST
```

```
END

TO HEX
REPEAT 6 [FD 50 RT 60] WAIT 60
END

TO SQ
REPEAT 4 [FD 50 RT 90] WAIT 60
END

TO TRI
REPEAT 3 [FD 50 RT 120] WAIT 60
END
```

Now that we've confused you with this procedure, let's confuse you even more. There's another way to shuffle things around. You saw that in the SHAPES :SIDE procedure.

```
TO SHAPES :SIDE
MAKE "SHAPE RANDOM 10
IF :SHAPE = 0 [TRIANGLE.RT :SIDE]
   on through...
IF :SHAPE = 9 [PARGRAM.LT :SIDE]
END
```

In effect, this procedure shuffles the Tangram procedures. It makes the variable :SHAPE a random number. It then matches the random number with the conditional statement to find a procedure to run.

Using SHUFFLE, the whole thing would be a bit easier.

One last point of confusion. There's also a PICK command that randomly selects an element from a word or list. For example:

```
MAKE "CHOICES [A B C D E]
```

```
SHOW PICK :CHOICES
D
```

Go ahead and explore how you could use all these choices. They all do things a bit differently. So you've got lots of choices for whatever you want to do.

Squares and Square Roots

Look back at the TRIANGLE.RT procedure. Got any idea what the SQRT 2 means?

That number is used to figure out how long the two short sides of the triangle are. The left side is the longest side, right? And you know that you have two equal sides connected by an angle of 90 degrees.

A long time ago, some mathematician figured out that when you know the long side of a triangle that has two equal sides and a right angle, then the short sides equal

<*Long side*> / SQRT 2

There are lots of rules like this for triangles and other shapes. You've already figured out a bunch of them.

"But what does SQRT 2 mean?"

Actually, it stands for the square root of 2. That sounds a lot worse than it really is. It doesn't have anything to do with the square shape. It's part of an arithmetic problem that asks what number, multiplied by itself, gives you the answer of 2.

What's SQRT 100? SQRT 9? SQRT 16?

Think about it for a minute. What number multiplied by itself equals 100?

10 * 10 = 100

What number multiplied by itself equals 9?

3 * 3 = 9

What number multiplied by itself equals 16?

4 * 4 = 16.

Now let's turn the square around. Square roots are like saying, "Here's the answer. Tell me what the question is. Here's 16, tell me what number multiplied by itself gives me that answer?"

So let's take a look at another question: 4 multiplied by itself equals what? MSW Logo has a POWER command.

FD POWER 4 2

That's like saying forward 4 to the power of 2, or 4 squared, or 4 times 4.

FD POWER 10 3

This is like saying Forward 10-cubed or 10 * 10 * 10 or 10 to the power of 3. The 2 in POWER 4 2 and the 3 in POWER 10 3 are called "exponents." This might make you think that POWER and EXP (or EXP) are the same. EXP is a trigonometric function that calculates the natural base *e* (2.7183. . .) raised to the power specified by its input.

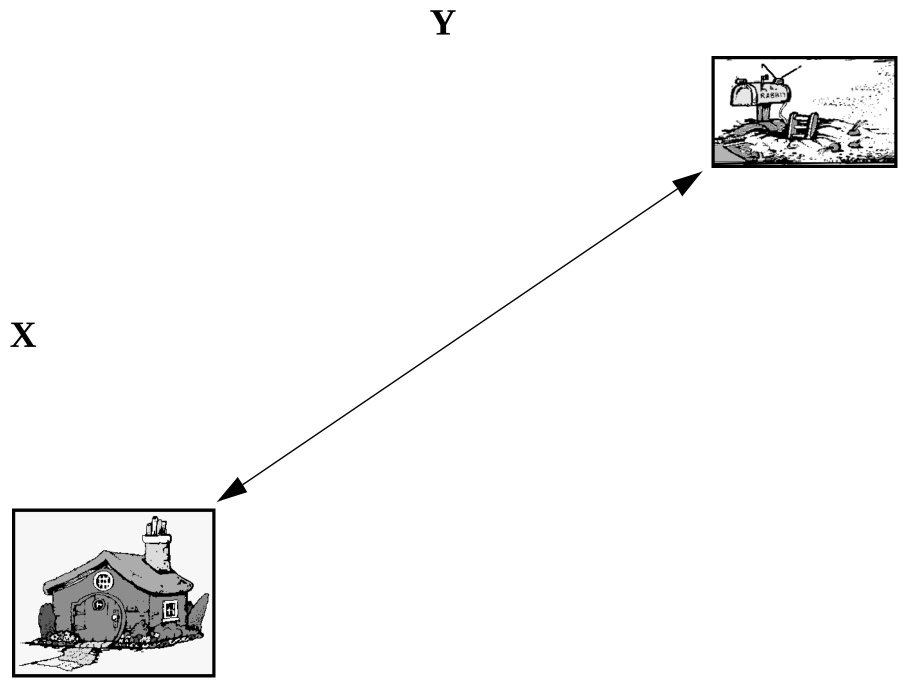
"I didn't understand a word of that."

Walking Distance

Let's first do some coordinate calculations to follow up on a Chapter 8 project.

Do you remember in Chapter 2 when you explored Turtle Town? You were asked to draw a map to your friend's house.

Now let's figure out the distances between two houses. Let's use Logy's and Morf's homes.



If you know the coordinates of each home, it's easy to use Logo and trigonometry to calculate the distance between the two homes. (You'll get into trigonometry in the next section.) Here's a variation of the DISTance procedure.

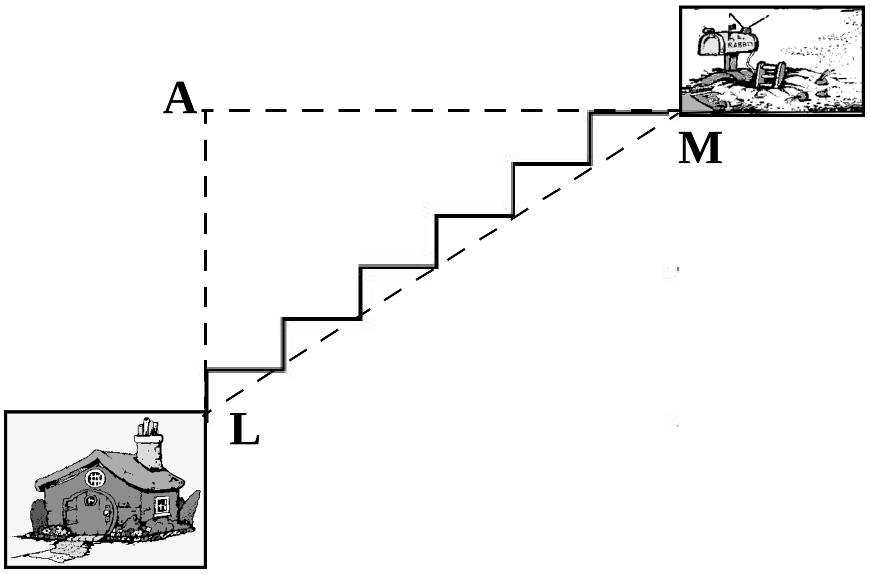
```
TO DIST  :X1 :Y1 :X2 :Y2
OP DIST1 :X1 - :X2 :Y1 - :Y2
END

TO DIST1  :DX :DY
OP SQRT (:DX * :DX ) + (:DY * :DY )
```

```
END
```

Logy's home is at [-100 -80] and Morf's Den is at [150 60]. What's the distance between the two homes?

```
SHOW DIST -100 -80 150 60
286.530975637888
```

This answer is fine if you're flying between homes. But since you're walking, you've still got some figuring to do.



Walking between the two homes means that you have to go up one block, turn right, go another block, turn left, and so forth. That's not the same as the straight line distance from L to M.

So how can you calculate the walking distance?

1. Start with what you know.

2. Define what you need to know.

3. Go find it.

You know the coordinates of each home, and you calculated the distance between the two homes.

- Logy’s Home is at [-100 -80].
- Morf’s Home is at [150 60].
- The direct distance between the two homes is 286.53.

You also know that turn from one block to the next, that the total distance you must walk going North (top of the screen) is the distance from L to A. The total distance you will walk going East (right side of the screen) is the distance from A to M.

Think about this. You also know the coordinates of point A. Take the Y coordinate of Morf’s home and the X coordinate of Logy’s home, and where are you?

Point A is at [-100 60].

The easy way to calculate the walking distance would be to use the DIST procedure. Rather than do that, let’s write a procedure to calculate the total walking distance.

```
TO WALK.DIST
(LOCAL "MX "MY "LX "LY "WALK)
MAKE “MX 150
MAKE “MY 60
MAKE “LX -100
MAKE “LY -80
MAKE “WALK ABS (:LX - :MX) + (:LY - :MY)
(PR [THE WALKING DISTANCE IS] :WALK)
END
```

The walking distance is 390 steps.

You can use the DIST procedure to prove this. First calculate the distance from Logy’s Home to A.

```
DIST -100 -80 -100 60 = 140
```

Now from Point A to Morf's Home.

```
DIST -100 60 150 60 = 250
```

One last test: does 286.53 squared equal 140 squared plus 250 squared.

```
SHOW SQRT 140 * 140 + 250 * 250
286.530975637888
```

Works for me!

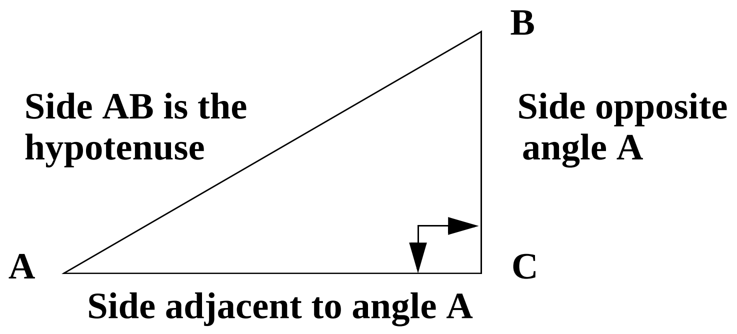
A Quick Look at Trigonometry

"You know, you can't really get away from trigonometry if you're going to explore angles. In simplest terms, trigonometry is the study of triangles. And what's a triangle other than three connected angles?"

Trigonometry — *trig* for short — also includes the study and use of what they call *trigonometric functions*. These functions include strange names such as sine, cosine, tangent, arctangent, and cotangant. They describe the functions of an angle that is described in terms of the ratios of pairs of sides or angles in a right triangle. You remember the right triangle, don't you?

A right triangle includes one right angle — right angles are 90 degrees — and two acute angles. Acute angles are less than 90 degrees. Obtuse angles are more than 90 degrees.but less than 180 degrees.

So here’s a right triangle:



A gentleman for Ancient Greece named Pythagoras came up with the rule that lets you determine the length of any side of a right triangle if you know the length of any two sides. The relationship that Pythagoras came up with says that side AC squared plus side BC squared equals the hypotenuse, side AC, squared.

$$A^2 + B^2 = C^2$$

Remember the DIST procedures?

```
TO DIST  :X1 :Y1 :X2 :Y2
OP DIST1 :X1 - :X2 :Y1 - :Y2
END

TO DIST1  :DX :DY
OP SQRT (:DX * :DX ) + (:DY * :DY )
END
```

Gee, when you look at those procedures, it seems as if trigonometry does have something to do with the coordinate system — which, of course, it does. Does that DIST1 procedure look familiar?

```
SQRT (:DX * :DX ) + (:DY * :DY)
```

is the same as

SQRT $:DX^2 + :DY^2$

which looks just like the Pythagorean theorem above. If you take the

SQRT $A^2 + B^2$

What will you get? Morf got the square root of C.

Defining Trig Functions

Once you know how the different sides of a right triangle relate to each other, you can define the trigonometric functions in terms of their right triangle relationships. Using the angles A, B, and C in the right angle shown previously, here's the definitions of functions with MSW Logo commands:

Sine (sin):	Sine of angle A = the ratio between the side opposite angle A and the hypotenuse. BC / AB.
Cosine (cos):	Cosine angle A = the ratio between the adjacent side and the hypotenuse. AC / AB
Tangent (tan):	Tangent angle A = the ratio between the opposite side and the adjacent side. BC / AC
Arctangent: (Arctan)	Arctangent angle A = The inverse of the tangent or the ratio between the adjacent side and opposite side. AC / BC

Using this information, you can define just about any trig function you may need. It also may help you understand some of the more complex procedures and commands in MSW Logo.

There are many books on trigonometry, if you really want to dig into it. There are also books on advanced Logo that talk about it much more than space allows in this book. For now, let's take a quick look at trigonometry in action.

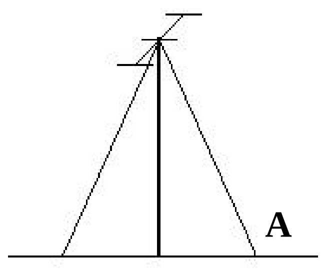
Morf's TV Antenna

Here's a problem that uses the sine function.

Morf has a TV antenna that is twelve feet tall. The instructions say that to keep it steady, he has to use four wires, each at a 65 degree angle from the ground.

How much wire does he need to buy?

Here's what the antenna looks like. Just remember that there are four wires and not just two as shown here.



The sine of angle A equals the opposite side (the height of the antenna) divided by the hypotenuse (the length of the wire).

SINE A = 12 / WIRE LENGTH

or

TOTAL WIRE = (12 / SINE A) * 4

In Logo, that's written as

```
SHOW (12 / SIN 65) * 4
52.9621401101996
```

Looks like Morf needs about 13-1/4 feet of wire for each of the four wires.

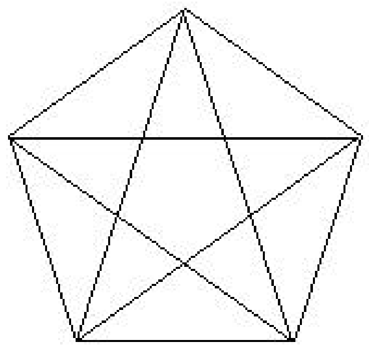
Side of the Star

Do you remember the problem with the stars back in Chapter 7? You started with a pentagon like this:

```
REPEAT 5 [FD 100 RT 72]
```

How did Ernestine know that the side of the star was 160? Maybe you can figure it out now.

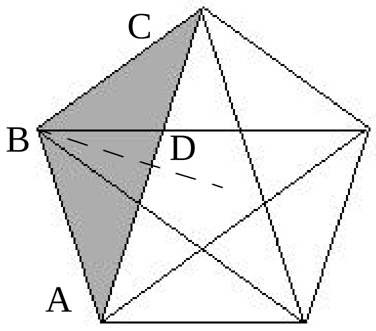
What do you know about that pentagon and the star?



The sides of the pentagon are 100.

The three angles at each corner equal 36 degrees.

Now, how can you prove that the long side of the shaded triangle is 160?



From HOME in the lower left, type

```
LT 18 FD 100 RT 108 / 2 FD 80
```

You already know that the big angle in the shaded triangle is 108. In Chapter 7, you proved that it's 180 - 72. So you cut that angle in half and go FD 80 — any distance, just so long as you cross the long side of the triangle.

You're left with two right angles, ABD and CBD.

Now you can calculate half of the long side of the shaded triangle using this trigonometric function.

COSINE 36 = ADJACENT / 100 (hypotenuse)

To put this in Logo terms:

```
SHOW  100 * COS 36 or
80.9016994374947
```

Now let's do some checking.

Using the Pythagorean theorem of $A^2 + B^2 = C^2$, you can calculate the length of the short side of the right triangle BD.

$100^2 = 80.9^2$ + What?

```
SHOW SQRT (POWER 100 2) - (POWER 80.9 2)
58.7808642331839
```

To check this out, try this:

```
REPEAT 5 [FD 162 RT 144]
LT 36
REPEAT 5 [FD 100 RT 72]

FD 100 RT 72 + 54  (Half the inside angle of 108)
FD 58.78
RT 90
FD 80.9
```

Where are you?

The length of the long side of the shaded triangle is 80.9 * 2 or 161.8, which is just about 162. Ernestine likes to work with rounded numbers. So now you know where she got it.

This is a bare taste of trigonometry. But maybe as you look at some of the procedures in the \projects\chpt10 and the Logolib subdirectory, they won’t seem quite so strange now.

You never know. You might just be able to figure them out and do some really neat things with them.

Here’s a challenge for you. Do you remember the "From the Center" exercises. You found the center point of different polygons. Now that you’ve had some experience with trigonometry, can you find out the distance from the edge to the center of any of the shapes?

But first, let’s check out a few other Logo commands.

Counting Numbers and Stuff

COUNT is another very useful Logo command. It outputs the number of elements in its input. That input can be a word or a list.

```
SHOW COUNT “LOGO
4   (There’s four letters in LOGO.)
```

```
SHOW COUNT [LOGY AND MORF]
3   (There’s three words in the list.)
```

Here’s an example from a procedure we talk more about in Chapter 12. Since you’re only interested in the first command, the one using COUNT, we left that second part off. Let’s see if we can use the REPEAT command to help make some sense out of COUNT.

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

You have a variable named :NUMS2. So for our explorations, let's make :NUMS2 equal to a list of numbers.

```
MAKE "NUMS2 (LIST 22 11 30 567 982)

SHOW :NUMS2
Result: [22 11 30 567 982]

SHOW COUNT :NUMS2
Result: 5

REPEAT ( COUNT :NUMS2 ) - 1 [FD 100 RT 90]
```

What would this command draw? You should know that. You learned about this shape back in Chapter 3.

____

Items, Members, and Things

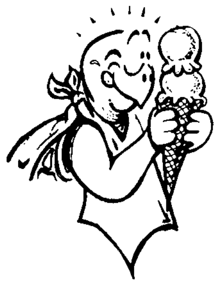
There are some other neat things you can do with words and lists. In the example above, you used the COUNT of the variable :NUMS2 to create a square. You can also select an item from a word or list and use that, too.

Here's an example. I bet you can guess what this is going to look like. It also tells you what ITEM does in a Logo procedure.

```
REPEAT ITEM 3 :NUMS2 [SQUARE RT 12]

TO SQUARE
REPEAT ( ( COUNT :NUMS2 ) - 1 ) [FD 100 RT 90]
END
```

What do you think ITEM 3 :NUMS2 is?

You know that :NUMS2 is a list — [22 11 30 567 982]. So what is ITEM 3 :NUMS2?



Another double-dip ice cream cone if you said 30.

ITEM outputs the third element of the variable :NUMS2. It doesn’t matter whether the variable is a word or a list.

```
SHOW ITEM 2 “CAT
  A

SHOW ITEM 2 7861236
  8
```

Get the idea?

In the :NUMS2 example, you knew what NUMBER you were looking for — the third element, or 30. But what if you didn’t know?

Logo lets you ask. Take a look.

```
TO CHECK :X
IFELSE MEMBERP :X :NUMS2~
  [REPEAT ITEM 3 :NUMS2~
  [SQUARE RT 12]] [SQUARE]
END

TO SQUARE
REPEAT ( ( COUNT :NUMS2 ) - 1 ) [FD 100 RT 90]
```

```
END

MAKE "NUMS2 [22 11 30 567 982]
```

In the CHECK procedure, Logo asks if :X is a member of the variable :NUMS2. If it is, it runs the REPEAT command

```
[REPEAT ITEM 3 :NUMS2 [SQUARE RT 12]]
```

If not, it just runs the SQUARE procedure.

Logo picks up these instructions from the IFELSE command. It's like saying if a condition is true, then do this, or else do this.

Ask Logo

You just saw MEMBERP. It asks if THING1 is a member of THING2. THING is the same as the dots in a variable, remember?

```
SHOW MEMBERP 1 [1 2 3 4]
TRUE

SHOW MEMBERP "CAT [DOG CAT RABBIT]
TRUE

SHOW MEMBERP "A "CAT
TRUE
```

There's also a MEMBER command. If "thing2" is a word or list, it shows the portion of THING2 from the first instance of THING2 to the end. If THING1 is not part of THING2, MEMBER outputs an empty word or list, whatever THING 1 is.

Does this have you nice and confused? Try it out! You're getting good at this by now.

Well, there are a number of other questions you can ask Logo. Your version of Logo may use a ? rather than a P at the end of the commands; MEMBER?, for example.

EQUALP

Are two words or lists equal or identical? For example:

```
IF EQUALP :X (ITEM 3 :NUMS2) [REPEAT …
```

EMPTYP

Is a word or list empty (") or ([ ])? For example:

```
IF EMPTYP :NUMS2 [STOP] [REPEAT …
```

If :NUMS2 is an empty list STOP, else run the REPEAT command.

NUMBERP

Is the object a number? For example:

```
IF NUMBERP (ITEM 3 :NUMS2) [REPEAT…
```

If ITEM 3 :NUMS2 is a number, then continue with the REPEAT command. Otherwise skip it and go on to the next line.

WORDP

Is something a word.

LISTP

Is something a list?

Logical Operations

There are three other primitives you need to look at before you leave Logo arithmetic: AND, OR, NOT.

AND

AND tests to see if all the conditions following the command are true.

```
MAKE "X 210
MAKE "Y 724
MAKE "Z 910
IF AND :X > 200 :Y < 800 [FD 100]
```

The conditions are true so the turtle moves forward 100.

Where you have more than two conditions, the commands and the command AND must be enclosed in parentheses.

```
MAKE "Z 555
IF (AND :X > 200 :Y < 800 :Z >500) [FD 100]
```

The conditions are true so the turtle moves forward 100.

OR

Where AND tests if all conditions are true, OR tests to see if any of the conditions is true. If you have more than two conditions to test, use parentheses as shown below.

```
IF OR :X > 200 :Z >1000 [FD 100]
IF (OR :X > 200 :Y < 800 :Z >1000) [FD 100]
```

Because at least one of the conditions is true, the turtle moves forward 100.

You could rewrite the Serpinski curve procedure from the last chapter using OR by changing

```
IF :LEVEL = 0 [FD :SIZE STOP]
IF :SIZE < 2 [FD :SIZE STOP]
```

to

```
IF OR (:LEVEL = 0)(:SIZE < 2) [FD :SIZE STOP]
```

NOT

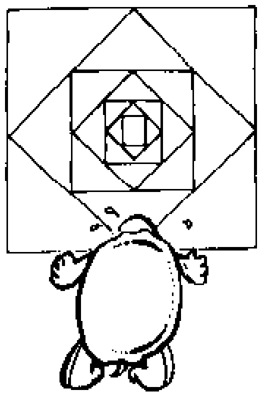
If the condition is false, NOT outputs True. In other words:

```
IF NOT :Z > 1000 [FD 100]
```

Since Z is *not* greater than 1000, the turtle moves forward 100.

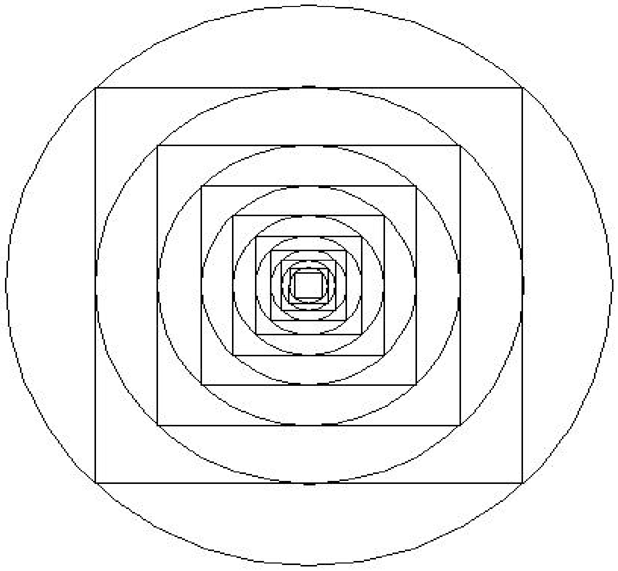
Math Challenges

Math Challenges may sound a bit like homework, but these problems are fun!



Here's a fairly easy challenge for you. Draw a series of squares within squares like the ones that are causing Logy to shake her head. It's tricky, sure. But it's really not that tough.

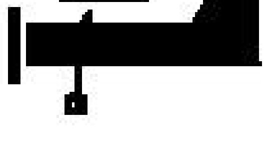
Here's another bit of a challenge to see what you've learned so far.



This drawing is a Mandala. People in India believe this is a symbol of the endless universe. Take some time to figure out how this procedure. It’s an interesting exercise in turtle geometry. (MANDALA.LGO in the \procs\chpt10 subdirectory.)

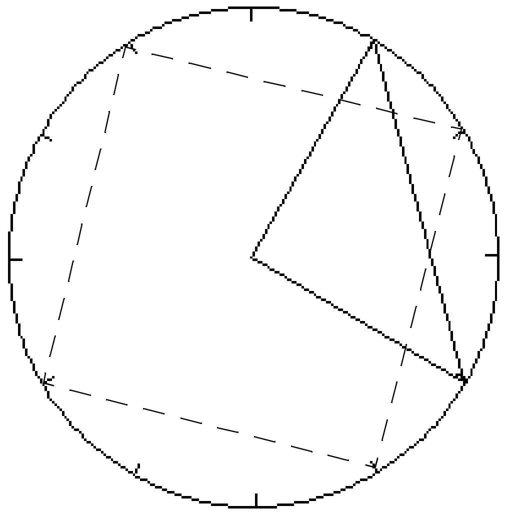
OK! After you get the Mandala procedure all figured out, try doing the same thing using triangles instead of squares. If you get bored with that, there are more math challenges in the \projects\chpt10 directory on the CD.

Rotating Planes in Space



"No, Morf. This is not about rotating or rolling your favorite Biplane. This is about moving shapes through space.

Do you remember the picture below? You saw it in Chapter 2 and also in Chapter 9, only it didn't have the dotted-line square in the picture then.



The original idea was to use string and make a triangle; first from 12:00 o'clock to 3:00 o'clock, the center, and back to 12:00 'oclock. Next you moved to 1:00 o'clock and made the triangle shown in the picture. Then you moved to 2:00 o'clock and so one, all the way around the clock.

The triangles moved around an axis that was at the center of the string clock. When you did the same thing with squares, they moved around on an axis at the center of the square.

Can you write Logo procedures that shows the shapes rotating around an axis?

Sure you can! Take a look at ROTATE.LGO in the \procs\chpt10 subdirectory of the CD. Type START to begin the show.

This procedure shows squares and cubes rotating around an axis. It also gives you another example of trigonometry at work.

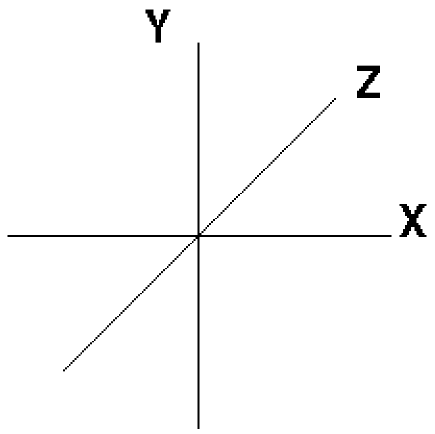
How about a challenge or two?

1. These procedures show the "side view" of the shapes rotating around a vertical axis. Write procedures to show a top view of the shapes rotating around the axis — as if you were looking down on the shapes.
2. Write procedures for other shapes, such as a rotating triangle. Use different types of triangles.
3. What about moving a pyramid through space?

Well, maybe you had better leave that job to the next project.

How About Turtle CAD

When some junior high students saw the work that the third grade students had done creating and folding soccer ball patterns, they wondered if it would be possible to work in three dimensions on the Logo screen. They were thoroughly familiar with the two dimensions of the x - y coordinate system.

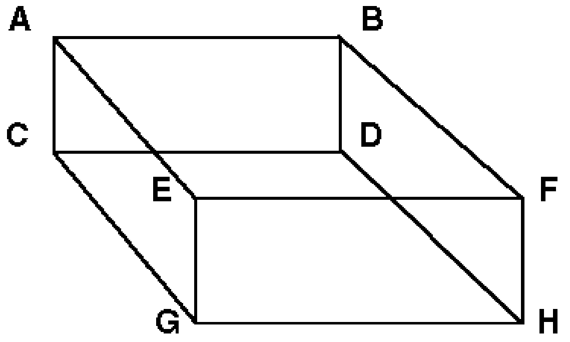


Could this be expanded to serve three dimensions: X, Y, and Z?

Yes, it can. In addition, the resulting procedure offers a good look at property lists, an often confusing feature of Logo.

In the procedure listed on the following pages, (ECONOBOX.LGO on the CD) the basic unit is the coordinate point as defined by the POINT procedure. Points have letter names and x, y, and z coordinates to position them in three-dimensional space.

Take a look at the DIAMOND and CAR procedures. To display a 3-D object, you must define each point in space and each line. Once you have defined all the required points by name and position, you can construct shapes like this one.



The FIGURE procedure takes the shape name and a list of two-point lists; for example, [[A B][A C][A E][B F][B D][C D][C G]…]. The two-point lists represent the line segments of the shape with each letter representing an endpoint.

Here's a simple procedure to develop a pyramid. Each point is defined along with the lines linking those points.

```
TO PYRAMID
POINT "A [0 0 0]
POINT "B [0 0 50]
POINT "C [50 0 50]
POINT "D [50 0 0]
POINT "E [25 50 25]
FIGURE "PYRAMID [[A B][B C][C D][D A][A E]
  [B E][C E][D E]]
END
```

The procedure allows you to create as many shapes as you want. Each can be as complex as you want. But only one can be manipulated at a time.

Once you have defined your shape, you can expand it or contract it, rotate it, magnify it, shrink it, and then restore it to its original shape.

To expand a shape, use EXPAND. Tell the procedure which shape to expand, which axis the expansion will operate on, and how much to expand it.

MAGNIFY is very similar to EXPAND. However, you don't specify an axis since the figure is magnified in all directions.

ROTATE operates on a plane: xy, xz, or yz. Specify the shape, the plane, and the degrees of rotation you want to see. As you move your shape through space, the turtle remembers the position of your shape and moves it from its last position. When you want to start over with a new shape, or start from your shape's original position, use

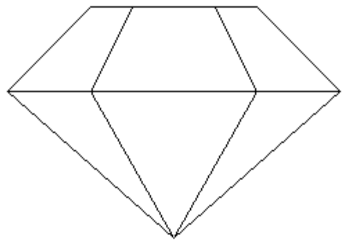
RESTORE "<*figure name*>.

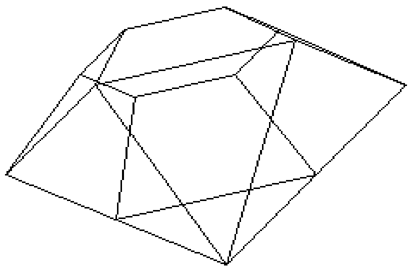
To get you started, there are three examples provided in the procedure below: a diamond, a pyramid, and an econobox-like car.

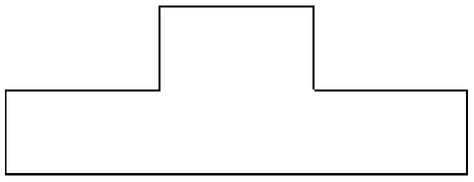
Type DIAMOND or CAR to see a front view of the figures. Then rotate the figures using commands such as:

ROTATE "<*figure name*> “XY 45
ROTATE "<*figure name*> "YZ 30

Now you're ready to start off on your own.

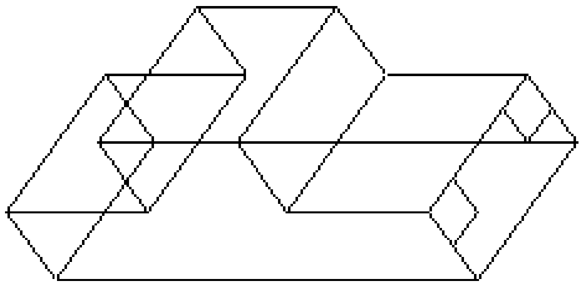






ROTATE “AUTO “XZ 90

]



ROTATE "AUTO "XY 45
ROTATE "AUTO "XZ 45

Understanding Property Lists

For the longest time, Property Lists were a major source of confusion — until Logy and Morf discovered a way to read them.

Put the PROPerty of :PROPERTY, which has the value of :VALUE, with the name :NAME. It helped when they saw PPROP written as a procedure.

```
TO PPROP :NAME :PROPERTY :VALUE
MAKE (WORD :NAME CHAR 32 :PROPERTY) :VALUE
END
```

Another way to look at the PPROP procedure is to use some more familiar terms.

```
PPROP "TEXAS "CAPITAL "AUSTIN
PPROP "TEXAS "ABBREVIATION "TX
PPROP "TEXAS "CITIES [HOUSTON DALLAS
   AMARILLO EL PASO]
PPROP "TEXAS "REGION "HILL.COUNTRY
```

Put the property of CAPITAL with the value of AUSTIN with the name, TEXAS. Put the ABBREVIATION TX with TEXAS. Put the cities of Houston, Dallas, Amarillo, and El Paso with Texas.

In the POINT procedure used on the previous pages, you have

```
PPROP :POINTNAME "POINT "TRUE
PPROP :POINTNAME "ORIG :COORDS
```

Put the property of POINT with the value of TRUE with the name :POINTNAME.

Put the property of ORIG with the value of COORDS with the name :POINTNAME. Take a look at the RESTORE procedure to see how this is used.

It’s not nearly as scary as you think!

Now that you have the properties defined, what can you do with them? For one thing, you can recall them using the GPROP procedure. That’s exactly what’s done in the RESTORE procedure.

```
TO GPROP :NAME :PROPERTY
OUTPUT THING (WORD :NAME CHAR 32
   :PROPERTY)
END
```

This outputs the THING (the value) defined in the PPROP procedure. For example:

GPROP "TEXAS "CAPITAL results in AUSTIN.

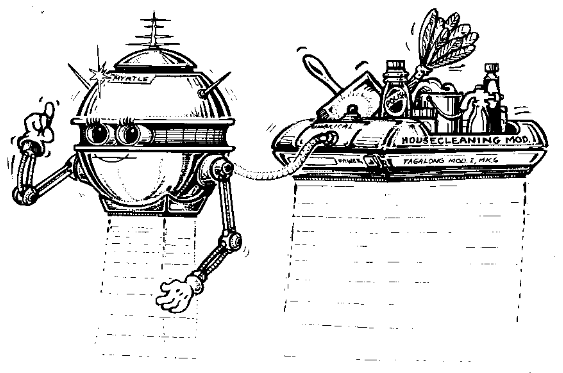
GPROP (FIRST :N9) "ORIG results in the original coordinates for :NAME.

The THREED procedures are one good examples of property lists. Play around with them on your own. You’re bound to find other uses.

Is Mathematics a Language

Are you convinced yet? If you don’t see how mathematics can be a universal language, there’s one more procedure to explore. It shows you how to convert numbers from one number system to another. For example, you can convert decimal numbers that you use everyday into binary numbers, which are the "language" of the computer.

Converting numbers from one number system to another is really something better left to Mrtle, our affable robot friend who never learned to speak computerese. She has her own



story to tell.

However, since number systems are part of The Great Math Adventure, it might be fun to explore a procedure that converts numbers from one number system to another. It's called CONVERT.LGO. To run it, type something like

SHOW (or PRINT) CONVERT 12 10 2

In other words, convert 12 from base 10 to base 2. Base 10 is the standard decimal number system people use in day-to-day living. Base 2 is the binary number system that computers use.

You've probably heard that computers speak the language of zeros and ones. It's like when a switch is OFF or ON. Explore the CONVERT procedure and you will see how easily everyday numbers convert to binary numbers the computer can understand.

Computers "understand" binary numbers thanks to an **instruction set**. This is a code designed into the computer that translates the OFF and ON signals into a code that people can understand and use. Those codes typically use octal (base 8) or hexadecimal (base 16) numbers.

Three sets of procedures were tacked on at the end. These convert numbers from base 10 to base 16 and back, from base 8 and back, and from base 10 to binary numbers (base 2) and back.

```
TO HEXTODEC :N
OP CONVERT :N 16 10
END

TO DECTOHEX :N
OP CONVERT :N 10 16
END

TO BINTODEC :N
OP CONVERT :N 2 10
END

TO DECTOBIN :N
OP CONVERT :N 10 2
END

TO OCTTODEC :N
OP CONVERT :N 8 10
END

TO DECTOOCT :N
OP CONVERT :N 10 8
END
```

Go ahead and explore.