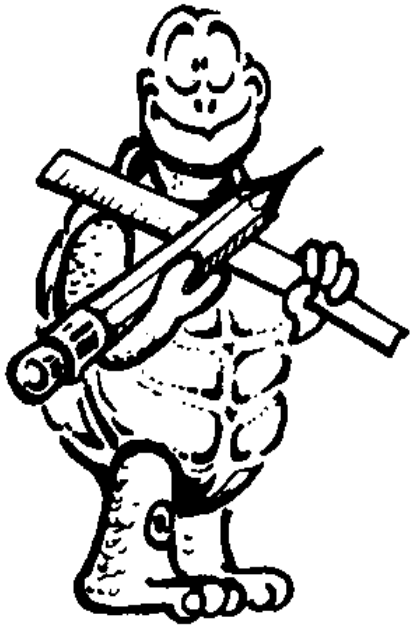
Chapter 9. The Great Math Adventure

"For some reason, when I'm doing my arithmetic homework, it seems more like a pain in the neck than a math adventure."



True enough!

But think about this for a moment… What part of your life does not involve mathematics…some form of 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 are counted as zeros and ones.

Everything we 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.

Come on…let's take a look.

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

This one's a bit more interesting.

FD 150 / 50 * 10

That's FD 150 divided by 50 = 3. 3 * 10 = 30 or FD 30.

FD :C / (:A * 3)

Does this command 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, it says FD 150. This is divided by 50 * 3 or 150. So we 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 de<u></u>fine the limits 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.

FD _____ _____ _____ _____ _____ _____

FD _____ _____ _____ _____ _____ _____

FD _____ _____ _____ _____ _____ _____

Mathematical Operations

There's lots of other ways you can use arithmetic with Logo. Here are the operations included with MSW Logo.

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

Some of these are a bit advanced. But some are very useful for all users.

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?

FD INT (121.8 - 21.1)

Forward the integer of 121.8 - 21.1. That equals 100.7. But, since the command is FD INTeger, or whole number, the decimal is dropped.

FD ROUND (121.8 - 21.5)

Forward 121.8 - 21.5 rounded off. The answer is 100.3, which rounds 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.

The Tangram Procedures

First let's take a look at the procedures to draw the Tangram Puzzle pieces you saw earlier.

```
TO TRIANGLE.RT :SIDE
FD :SIDE RT 135
FD :SIDE / SQRT 2 RT 90
FD :SIDE / SQRT 2 RT 135
END

TO TRIANGLE.LT  :SIDE
FD :SIDE LT 135
FD :SIDE / SQRT 2 LT 90
FD :SIDE / SQRT 2 LT 135
END

TO SQUARE.LT  :SIDE
MAKE "SIDE1 :SIDE / ( 2 * SQRT 2 )
REPEAT 4 [ FD :SIDE1 LT 90 ]
END

TO SQUARE.RT  :SIDE
MAKE "SIDE1 :SIDE / ( 2 * SQRT 2 )
REPEAT 4 [ FD :SIDE1 RT 90 ]
END

TO MED.TRI.RT  :SIDE
FD 2 * ( :SIDE / ( 2 * SQRT 2 ) ) RT 135
FD :SIDE / 2 RT 90
FD :SIDE / 2 RT 135
END
```

```
TO MED.TRI.LT  :SIDE
FD 2 * ( :SIDE / ( 2 * SQRT 2 ) ) LT 135
FD :SIDE / 2 LT 90
FD :SIDE / 2 LT 135
END

TO SMALL.TRI.RT  :SIDE
FD :SIDE / 2 RT 135
FD ( :SIDE / SQRT 2 ) / 2 RT 90
FD ( :SIDE / SQRT 2 ) / 2 RT 135
END

TO SMALL.TRI.LT  :SIDE
FD :SIDE / 2 LT 135
FD ( :SIDE / SQRT 2 ) / 2 LT 90
FD ( :SIDE / SQRT 2 ) / 2 LT 135
END

TO PARGRAM.LT  :SIDE
REPEAT 2 [ FD :SIDE / ( 2 * SQRT 2 ) LT 45 ~
        FD :SIDE / 2 LT 135 ]
END

TO PARGRAM.RT  :SIDE
REPEAT 2 [ FD :SIDE / ( 2 * SQRT 2 ) RT 45 ~
        FD :SIDE / 2 RT 135]
END
```

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?

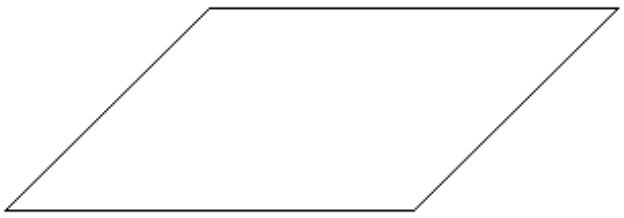
You'll have to run the procedure to see.

Remember, when you run the TANGRAM procedure, you have to type a value for :SIDE…

```
TANGRAM 200
```

What's a Parallelogram

"Logy, there's something strange here? You've got a rectangle that looks like it's falling over."



"You're right, you know. I never thought of a parallelogram like that," said Logy.

"Para-who?"

"That's another shape, a parallelogram. You might call that the granddaddy of a square," Logy answered.

"I don't get it? What do you mean, granddaddy?"

"Take a look at this procedure. It's called PARGRAM for short."

```
TO PARGRAM  :SIDE
REPEAT 2 [ FD :SIDE RT 45 FD :SIDE / 2 RT 135]
END
```

"What would happen if I changed the angles from RT 45 and RT 135 to RT 90?"

"Hey, that would be a rectangle," said Morf, jumping up and down excitedly.

"So, you can say that a rectangle is like the 'child' of a parallelogram.

"Now look at the sides. You've got two that are the length of :SIDE and two that are the length of :SIDE divided by 2. What would happen if I took away the 'divided by 2?"

"Well, let's see. You'd have two sets of sides that all use the same variable. That means they'd all be the same."

"And if all the angles are RT 90, what's that?"

“Hey, that’s a square!”

“OK, then. Is it fair to say that a square is the child of a rectangle?"

“Seems that way.”

“Right! So now we can add a new rule.”

A square has four equal sides and four equal angles.

A rectangle has two sets of equal sides and four equal angles.

A parallelogram has two sets of equal sides and two sets of equal angles.

Fun With Tangrams

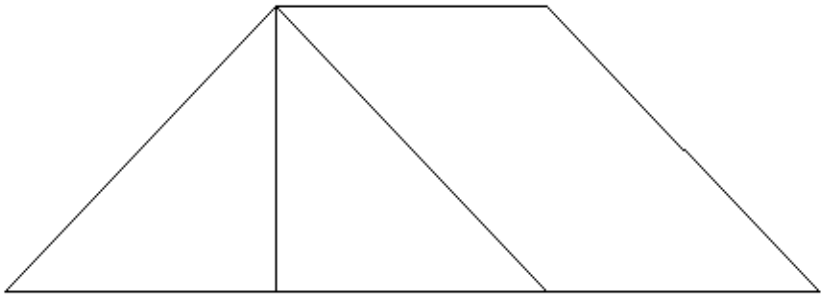
Enough of this stuff. Let’s have some more fun with tangrams?

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?

You'll find lots of puzzle books that have tangrams in them. But you don't really need those books, do you? I’ll bet you can think up all sorts of shapes on your own.

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. CRAZY.SHAPES is included in the TANGRAM.LGO procedure.

```
TO CRAZY.SHAPES  :SIDE
SHAPES :SIDE
MOVE :SIDE
CRAZY.SHAPES :SIDE
END
TO SHAPES :SIDE
MAKE “SHAPE INT 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 INT 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
```

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 we know that we have two equal sides connected by an angle of 90 turtle turns, or 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's lots of rules like this for triangles and other shapes. We'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 root around. Square roots are like saying, "Here's the answer. Tell me what the question is…here's 16, tell me how I got that?"

So let's look at the question…4 multiplied by itself equals what? In the list of Logo arithmetic operations, there is 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 and the 3 are called "exponents."

"What about that thing called RANDOM?"

"Wait a bit. There's a really great example of that coming up."

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

SHOW COUNT [LOGY AND MORF]
3
```

Here's a line from a procedure we talk about in chapter 14. There's a lot in this example. But you're only interested in that first line…the one with COUNT in it. Let's see if we could use that line to help make some sense out of COUNT.

```
REPEAT ( ( COUNT :NUMS2 ) - 1 ) ~
    [MAKE "NUMS1 BF :NUMS1~
    IF (FIRST :NUMS1) = FIRST :NUMS~
    [MAKE "CARRY FIRST BF :NUMS]]
```

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

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

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

SHOW COUNT :NUMS2
      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 2.

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 Gold Star 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, 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 (LIST 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 line…

```
[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.

There are a number of other questions you can ask Logo.

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…

IFELSE EMPTYP :NUMS2 [STOP] [REPEAT …

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

NUMBERP…Is an object a number?

For example…

IF NUMBERP (ITEM 3 :NUMS2) [REPEAT…

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

WORDP

LISTP

These are like NUMBERP only these commands ask if the object is a word or a list.

Logical Operations

There are three other primitives we need to look at before we 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
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, they 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 are 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.

NOT

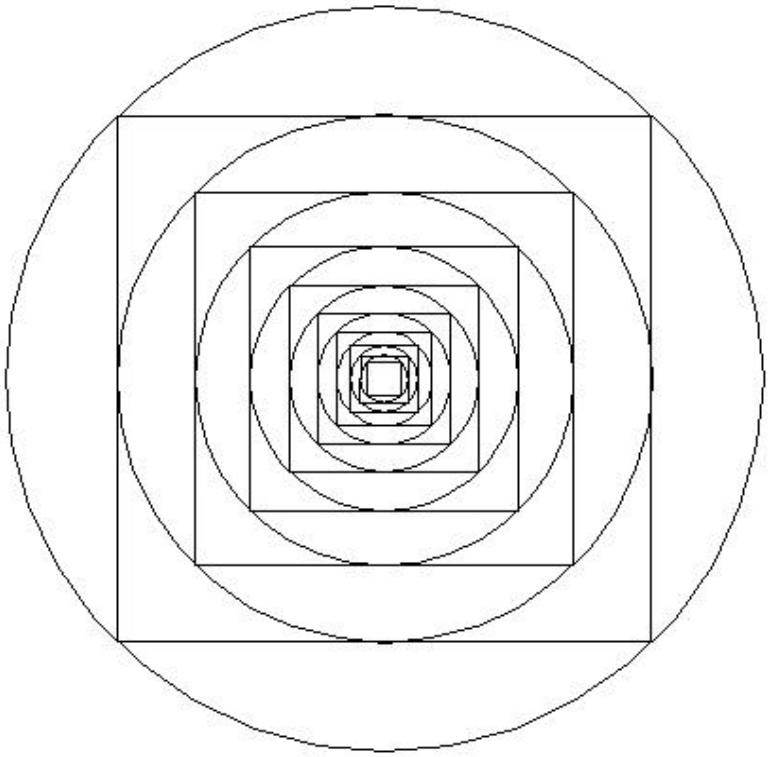
If the conditions are false, NOT outputs True. In other words…

```
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…and a bit of a challenge to see what you've learned so far.



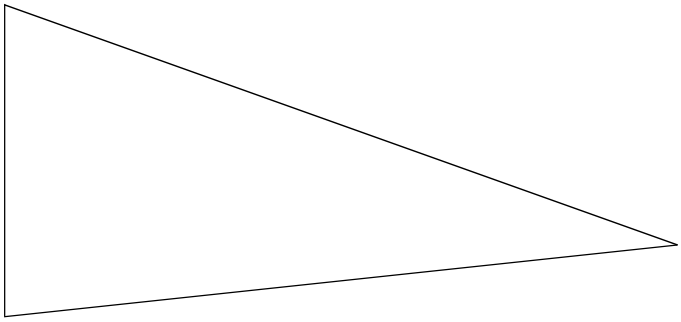
This is a Mandala. People in India believe this is a symbol of the endless universe.

The procedures to create this drawing are on the next page. Take some time to figure out how they work. They're an interesting exercise in turtle geometry.

```
TO MANDALA :RADIUS :CENTER
CIRC
SQUARE
IF :RADIUS < 10 [STOP]
MANDALA :RADIUS :CENTER
END

TO SQUARE
SETH 225 FD :RADIUS SETH 0
MAKE "SIDE SQRT (2 * (:RADIUS * :RADIUS))
PD REPEAT 4 [FD :SIDE RT 90]
MAKE "RADIUS :SIDE / 2
END

TO CIRC
PU SETPOS :CENTER
SETX XCOR - :RADIUS
PD CIRCLER :RADIUS
PU SETPOS :CENTER
END

TO CIRCLER :RADIUS
LOCAL "STEP
MAKE "STEP 2 * :RADIUS * 3.1416 / 36
REPEAT 36 [RT 5 FD :STEP RT 5]
END
```

OK…now that you've got the Mandala procedure all figured out, try the same thing using triangles instead of squares.

Here's another one…



Draw a triangle on the computer…any type of triangle will do.

Now draw a circle around that triangle so that the edge of the circle touches the three points of the triangle.

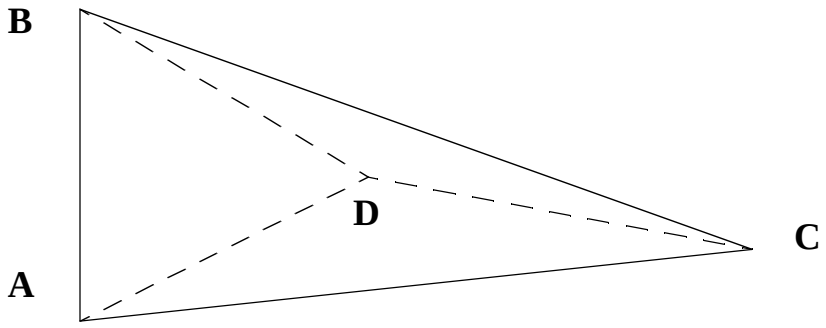
This is going to get a bit complicated. Just remember...the whole idea behind Logo is to break a problem down into its simplest parts. Start with what you know. Determine what you don't know. Then go find it.

What do you know?

You know that the three points of the triangle are going to be on the edge…the circumference…of the circle. If you can find a point that is the same distance from each of those points, then you have the center of the circle, right?

To make things easier to understand, let's label the points on the circle. We'll call them A, B, and C.

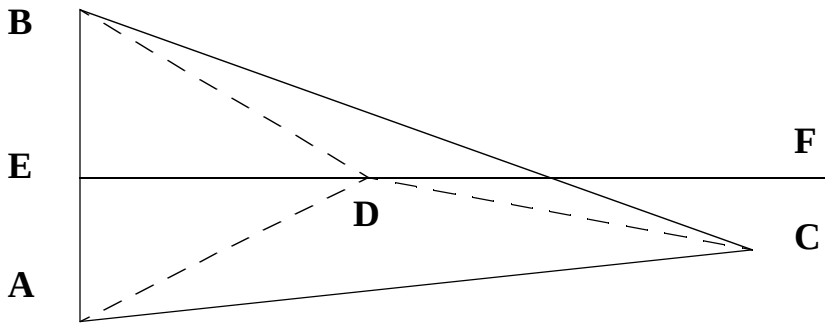
We have to find point D…a point inside the triangle that is the same distance from A as it is from B and C.



If point D is the same distance from A, B, and C, then point D must be the center of the circle and the three lines, AD, BD, and CD are each a radius of the circle we are supposed to draw.

Now…how can we prove that.

Draw the line EF so that it is perpendicular to the middle of line AB.



Perpendicular means that the line EF is at right angles to line AB.

What can you learn from this drawing now?

You have two triangles…ADE and BDE…that share one side and have two short sides that are equal. Therefore, the sides AD and BD must be equal.

OK…if we can find the point on line EF that makes these two lines equal to line CD, we have found the middle of the circle we want to draw.

Let's do it.

The Random Triangle

The first step is to create a random triangle… something like we have already drawn.

```
TO RANDOM.TRI
MAKE "POINTA POS
FD 100 RT 120 - RANDOM 30
MAKE "POINTB POS
MAKE "DIST 250 - RANDOM 100 FD :DIST
MAKE "POINTC POS  HOME
END
```

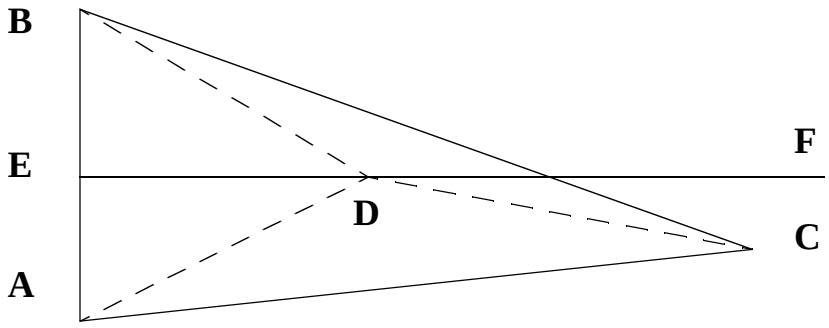
This procedure starts from HOME…POINT A with coordinates 0,0. The turtle goes FD 100 and turns right a random angle…somewhere between 120 and 90.

This is POINTB…coordinates 100,0.

The turtle then goes FD between 150 and 250 and sets POINTC. Then the turtle goes Home.

The next step is to draw the perpendicular line.

```
TO RT.ANGLE
SETPOS :POINTA
FD 100 / 2 RT 90
MAKE "POINTE POS
FD 200 PU HOME PD
END
```



Now we have a drawing something like this, but without the dotted lines.

What do we need to know now to complete our circle? We need to find the point D on line EF that is the same distance from B as it is from C. We already know that AD and BD are going to be equal…and that each is going to be a radius of our circle. So if we can make one equal to line DC, the other is automatically equal to DC.

The first thing we need for that is a distance procedure.

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

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

The DIST procedure measures the distance between two sets of coordinates. MSW Logo measures that difference very precisely. So to keep things simple and easy to compare, the output is an integer…a whole number. (It's a lot easier to compare whole numbers than it is to compare long decimals.)

Now let's put the DISTance procedure to work. We'll use it to calculate two distances…the distance between B and D…and the distance between C and D. When these are the same, we'll draw our circle.

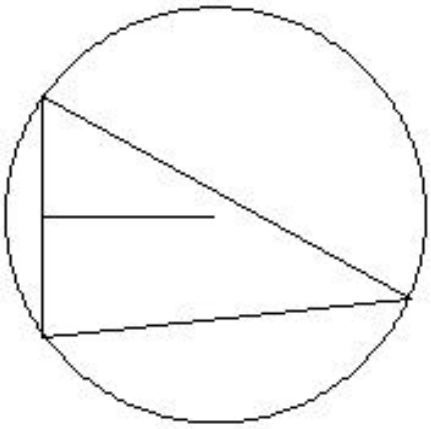
```
TO CHECK.DIST
MAKE "BD DIST FIRST :POINTB LAST :POINTB
        FIRST :POINTD LAST :POINTD
MAKE "CD DIST INT FIRST :POINTC
        INT LAST :POINTC FIRST :POINTD
        LAST :POINTD
SHOW :BD SHOW :CD
TEST :BD = :CD
IFTRUE [HT CIRCLE :POINTD :BD]
IFFALSE [FD 1 MAKE "POINTD POS
        CHECK.DIST]
END
```

Here's more new commands…FIRST and LAST. You already know that :POINTB is a list of two coordinates. So FIRST :POINTB must be the first coordinate. And if that's true, then LAST :POINTB must be the last element in the list or the y-coordinate. You'll learn more about characters, numbers, words, lists, FIRST, LAST, and other good stuff in chapter 11.

Now let's run through the CHECK.DIST procedure. The first two lines calculate the distances BD and CD. So that you can see how these distances change, the distances are printed in the Command Box.

Then Logo tests the two numbers. If :BD = :CD is true…they are equal…Logo draws a circle with :POINTD as the center and a radius of :BD. If the two distances are not equal, the turtle moves FD 1 and checks the distances again.

```
TO CIRCLE :CENTER :RADIUS
LOCAL "AMT
MAKE "AMT :RADIUS * PI / 180
PU SETPOS :CENTER
SETX XCOR - :RADIUS SETH 0 PD
REPEAT 360 [FD :AMT RT 1]
PU SETPOS :CENTER PD
END

TO PI
OP 3.14159
END
```



To put the whole thing together, here's a place to start.

```
TO START
RANDOM.TRI
SETPOS :POINTA
FD 50 RT 90
MAKE "POINTD POS
CHECK.DIST
END
```

Take your time with this procedure. Come back to it when you're ready. This is a good stepping stone to some of the other procedures you'll see in the rest of this book.

Numbers and Number Systems

I know what numbers are. But what's a number system?

There's a big, big difference between numbers and number systems. Numbers make up the arithmetic tables you memorize in school...2 + 2 = 4, 4 + 4 = 8, 8 + 8 = 16, and so on. Number systems, on the other hand, let you move far beyond the arithmetic tables to where mathematics becomes a language all its own.

Look at it this way.

There are 26 letters in the English language. These are used to make words and sentences. But words alone don't make a language. You have to have a way to string those words together so they make sense to the person you're trying to talk to.

Letters and words aren't worth much unless you know what they mean and how they fit together. More importantly, these letters and words have to mean the same thing to the person or people you're talking to. If they mean one thing to you and something else to other people, that gets pretty confusing.

When using numbers, we use the 10 digits from zero to 9…0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. Just like letters are combined to make words, digits are combined to make numbers. But the numbers don't mean anything unless you know how and why they fit together.

What does 1,487,653 mean? This is where our number system comes in -- the decimal number system. Decimal means that it is based on the number 10…or as the mathematicians say, it is base 10. In the decimal number system, we learn to use tens, hundreds, thousands, ten thousands, and so on. So we read that number as one million, four hundred eight-seven thousand, six hundred fifty-three.

Bits and Bytes

Ever hear of a byte? Sure you have. You've see the abbreviations KB and MB a lot, I'm sure. They stand for Kilo-Byte or one thousand bytes, and Mega-Byte or one Million Bytes. Computer memory is organized in bytes.

A byte is 8 bits. So what's a bit?

A bit is one piece of memory, either a zero or a one. Either the power to the memory unit is On for one, or Off for zero.

These 0's and 1's are called "machine language." It's the language computers use to communicate. Take a look...

1 + 1 = 10

To the computer, this addition problem is correct. Its number system only uses the two digits, 0 and 1.

When you add nine plus one, you have to go from the one's to the ten's column, right? The reason is that there is no single character in the decimal number system to represent ten. So you write a zero and add a one.

10 * 1 = 10
10 * 10 = 100
10 * 100 = 1,000
10 * 1,000 = 10,000

What does this tell you about
10?

This same idea applies to the binary system. You only have a zero and a one. There is no two.

So when you multiply the binary numbers…

10 * 1 = 10	**two * one = two.**
10 * 10 = 100	**two * two = four.**
10 * 100 = 1000	**two * four = sixteen.**
10 * 1,000 = 10000	**two * sixteen = thirty-two.**

Converting Numbers

Yes, this is all very confusing. So here's a procedure that converts numbers from one number system to another. It's called CONVERT.LGO. To run it, type something like…

```
SHOW CONVERT 12 10 2
```

In other words, convert 12 from base 10 to base 2. What do you get?

```
TO CONVERT :N :FRBASE :TOBASE
OP DEC.TO.ANYBASE ~
      ANYBASE.TO.DEC :N :FRBASE 1 :TOBASE
END

TO ANYBASE.TO.DEC :N :BASE :POWER
IF EMPTYP :N [OP 0]
OP ( :POWER * C.TO.N LAST :N ) + ~
      ANYBASE.TO.DEC BL :N :BASE ~
      :POWER * :BASE
END

TO C.TO.N :N
IF NUMBERP :N [OP :N]
OP (ASCII :N) - 55
END
TO DEC.TO.ANYBASE :N :BASE
IF :N < :BASE [OP N.TO.C :N]
OP WORD DEC.TO.ANYBASE ~
      INT QUOTIENT :N :BASE :BASE N.TO.C ~
      REMAINDER :N :BASE
END
```

```
TO DIVISORP :A :B
OP 0 = REMAINDER :B :A
END

TO N.TO.C :N
IF :N < 10 [OP :N]
OP CHAR 55 + :N
END
```

Two sets of procedures were tacked on to the end…two convert numbers from base 10 to base 16 and two convert numbers 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
```

Two others that might be useful are…

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

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

Why add octal numbers? Because computers use binary, octal, and hexadecimal numbers.

Remember…a "bit" of memory is a tiny circuit that is OFF or ON, representing zero or one.

Early desktop computers used 8-bit circuitry. Eight bits equals one byte. Even though today's computer use 32-bit technology, we measure memory and storage Kilobytes and Megabytes.

Octal numbers are Base 8, using the digits 0 to 7. The digits 8 and 9 don't exist. So you have this situation…

7 + 1 = 10

Hexadecimal numbers are even more confusing. They are Base 16. Since we can only write ten digits, we add letters for ten to 15. So you have…

9 + 1 = A	**or ten.**
A + 1 = B	**or eleven.**
B + 4 = F	**or fifteen.**
9 + 7 = 10	**or sixteen.**
10 + B = 15	**or twenty-one.**

Computer Words

It would impossible to talk to a computer if its memory was just a jumble of on and off circuits. So we organize the circuits into 16-bit "words."

00000000 00000001
00000000 00000001 +
00000000 00000010

00000000 00001111 = 15
00000000 11111111 = 255
11111111 11111111 = 65,535

Imagine what it would be like if you had to program in machine language using only zeros and ones?

Think for a minute! The computer doesn't think. But if certain combinations of circuits are on, the computer does certain things.

SO-O-O…it seems that the computer must use some kind of code to translate the on and off circuits into binary numbers…then into instructions.

Each microprocessor…the integrated circuit that controls the computer…uses a special set of instructions to translate 0's and 1's into actions on the screen. Computer languages such as Logo translate your commands into instructions that the instruction set of the computer can understand.

Mrtle, the robot, has a whole book on how this works. So we'll keep this explanation short. Just remember that it all has to do with special codes.

I don't know about you…but this seems like a pretty good example of numbers being used as a language.

More Math Adventures

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 was installed when you installed MSW Logo. 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 thing in the Cost procedures…

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

What's that?

```
TO COST.OF.BOX
PR2
MAKE "BOXSIZE ((5 * (1 + RANDOM 5)) + 10)
MAKE "BOXCOST (:BOXSIZE * :COST)
(PR [THE COST OF A BOX OF] :BOXSIZE :COOKIE
[COOKIES IS $] :BOXCOST)
PR []
END

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
```

```
TO GAME
CLEARTEXT
(PR [In one hour you sell] :AMOUNT)
(PR :COOKIE [cookies.])
NEXT
(PR [The] :COOKIE [ cookies cost you $] (:AMOUNT *
:COST))
NEXT
(PR [Your gross sales were $] (:SALE.PRICE *
:AMOUNT))
PR []
PR [This is how much money you took in.]
NEXT
SETCURSOR [0 17]
(PR [Your net profit for this turn was $] ~
      ((:SALE.PRICE - :COST) * :AMOUNT))
PR2
MAKE "PROFIT (:PROFIT + ((:SALE.PRICE - ~
      :COST) * :AMOUNT))
(PR [Your profit for the game is $] :PROFIT)
NEXT
END
```

Engineering Notation

Engineering notation is strange. 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

SHOW 1.E - 5 * 9

SHOW 1.E + 14 * 128

What kind of answer is that…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?

You'll find it's shorthand for writing very big or very small numbers. And soon you'll be able to read them just like you read other numbers.

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!

There's lots more about math adventures coming up…Logo adventures, too!


MY JOURNAL
OR HOW I DID IT!