Chapter 8. Recursion

Recursion is when a procedure uses itself as part of the solution.

Say what?

Yes, strange as that may seem, a recursive procedure is one that calls itself as part of the total solution. You have seen a few examples of this in other chapters. It is like the two turtles in the picture. Each turtle is using the other to draw itself.

Here's a fun procedure to help you make some sense out of recursion, even though it really isn't Logo.

```
TO GET.THROUGH.LIFE
GET.THROUGH.TODAY
GET.THROUGH.LIFE
END
```

Think about it for a moment. This says that to get through life, you have to get through today. Once you are through today, you have to move on, right?

But where?

You can't go backward. You can't stop time. You have to get through life. But to get through life, you have to get through today. But each day is different. So while this looks like a simple loop, it really isn’t.

Let’s add another twist to this. Let’s suppose that when you’re standing in front of the Pearly Gates, you want to take a look at your Book of Life.

Embedded recursion can actually help you with this. All you have to do is add a line to your life procedure.

```
TO GET.THROUGH.LIFE
GET.THROUGH.TODAY
GET.THROUGH.LIFE
PRINT BOOK.OF.LIFE
END
```

It's like every time you GET.THROUGH.TODAY, you write a page in a book. Only the pages are not printed until you halt the procedure. When you stop the procedure, recursion reads the entire book starting with the last page it saved.

Awfully simple…or simply awful.

Tail-end Recursion

Yes, recursion is confusing. So let's try it with something we know something about.

```
TO MAZE :N
FORWARD :N
RIGHT 90
MAZE :N + 5
END
```

This is an example of what we call "tail-end recursion." The recursive call is at the tail-end of the procedure.

To see just how this works, type…

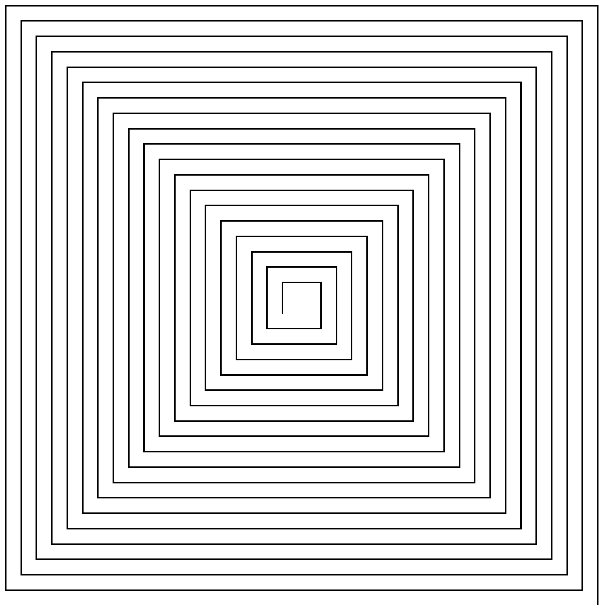
```
MAZE 20
```

Now watch what happens. The turtle goes forward 20 steps and then turns right 90 turns. Then the procedure

tells :N (that's 20 to us) to become :N + 5 (that's 25 to us now). Then the procedure says…

```
MAZE 25
```

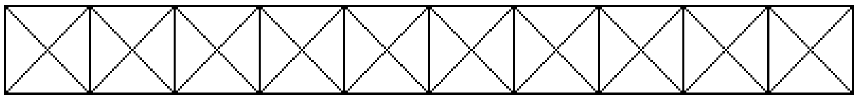
It starts all over again. MAZE 25 becomes MAZE 30. MAZE 30 become MAZE 35, and on and on and on. The screen soon looks something like this, and it just keeps on going, gradually filling up the screen.



Of course, we can put a STOP in there if we want.

```
TO MAZE :N
IF :N = 300 [STOP]
FORWARD :N
RIGHT 90
MAZE :N + 5
END
```

The first line sets up a conditional test. Each time the procedure runs, it tests :N to see if it equals 300. When :N does equal 300, the procedure stops.

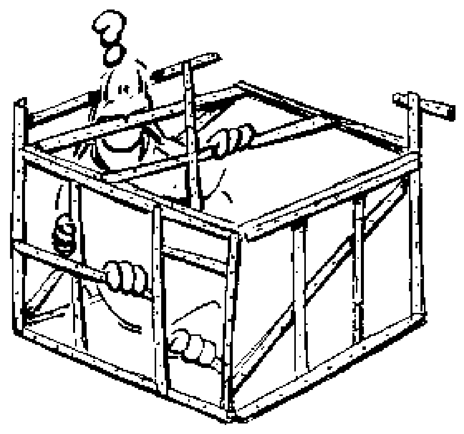


Let’s take a look at some other examples.

Did you ever play with an Erector Set? Did you ever build bridges or highways? Well, Logo can help you draw your plans.

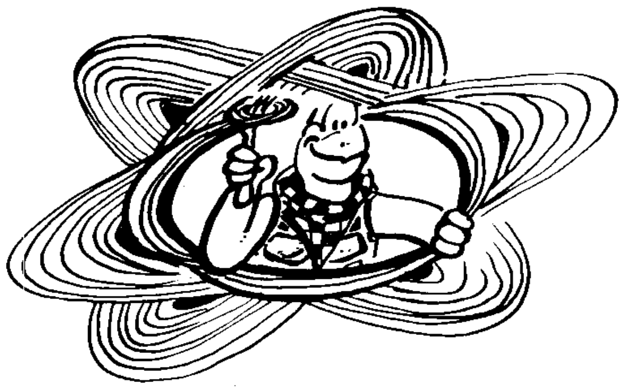
```
TO ERECTORSET :N :X
IF :X = 0 [STOP]
SECTION :N
MOVE :N
ERECTORSET :N :X - 1
END

TO MOVE :N
RT 90 FD :N LT 90
END

TO SECTION :N
REPEAT 4 [TRI :N FD :N RT 90]
END

TO TRI :N
FD :N RT 135
FD :N / SQRT 2 RT 90
FD :N / SQRT 2 RT 135
```

```
END
```



Do you like Italian food? How about spaghetti?

```
TO SPAGHETTI
CIRCLE 5
CIRCLE 4
CIRCLE 3
CIRCLE 2
RT 45
SPAGHETTI
END
```

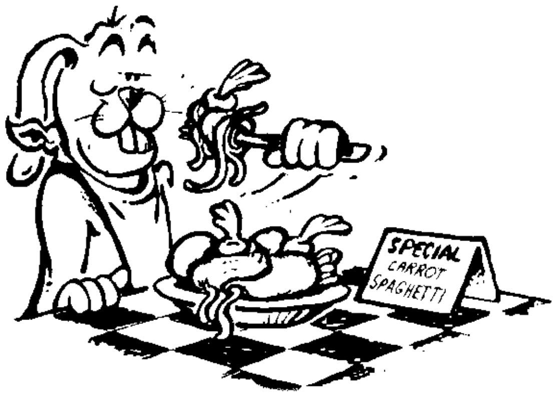


```
TO CIRCLE :N
REPEAT 36 [FD :N RT 10]
END
```

You can make the SPAGHETTI procedure even more variable with a few changes:

```
TO SPAGHETTI :N
CIRCLE :N + 5
CIRCLE :N + 4
CIRCLE :N + 3
CIRCLE :N + 2
```

```
RT 45
SPAGHETTI :N
END
```



How about this one?

```
TO SPAGHETTI :N
CIRCLE :N
IF :N = 0 [STOP]
RT 45
SPAGHETTI :N - 1
END
```

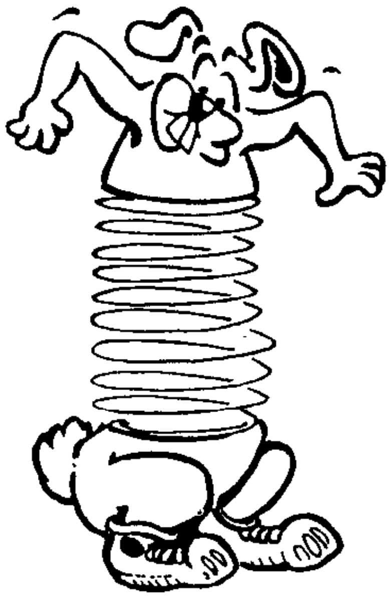
This is a recursive procedure like some others you have used before. Will it draw spaghetti like the others? Why?

In addition to spaghetti, what else can you do with this circle procedure.

```
TO CIRCLE :N
REPEAT 36 [FD :N RT 10]
END
```

How about a Slinky?

```
TO SLINKY
LT 90
CIRCLES
END

TO CIRCLES
CIRCLE 5
FD 20
CIRCLES
END
```



You might think that recursion is just like a loop…that it just goes around in circles. Not quite! Sometimes things aren't what they seem to be.

Embedded Recursion

To check this out, let's look at a procedure that uses “embedded recursion.”

```
TO TEST.RECURSION :N
PRINT [IS THIS RECURSION?]
IF READWORD = “YES [TEST.RECURSION :N + 1]
PRINT :N
END
```

In this procedure, the variable “N” is used as a counter. It is used to keep track of your answers to the question, IS THIS RECURSION?"

Watch and see. Type…

```
TEST.RECURSION 1
```

When you start the procedure, the first thing you see on the screen is the question…

```
IS THIS RECURSION?
```

READWORD tells Logo to stop and wait for you to type an answer. Type YES. The counter knows that this is your first answer.

Then we come to a test. If the word you typed was YES, then the procedure calls itself. What happens?

You guessed it…there's that question again…

```
IS THIS RECURSION?
```

Type YES a few times when you see the question. Then type NO. What happens this time?

When you type NO, the procedure comes to the test. This time the word you typed doesn't match YES and so the computer reads the next line…

```
PRINT :N
```

WOW…what happened then? Why were so many numbers printed? That's what makes recursion different from just a simple loop.

When you first look at this procedure, it seems like it is going to go around in a loop. Every time it passes the TEST.RECURSION :N + 1 line, the counter is going to add 1. Then, when you type NO instead of YES, you'd think the procedure would simply print the current value of N.

Well…that isn't the way recursion works. Morf has one of his trails to show you what happens.

Rabbit Trail 19. Recursive Pages

Let's look at this procedure again. You'll need some blank paper, a pencil, and scissors for this one.

Fold the paper in half. Then fold it in half again…and again…then one last time. Crease the edges good and then open up your piece of paper.

Cut the paper along the folds. You should end up with 16 small pieces of paper. Now number these “pages” from 1 to 16 by writing a small number at the bottom of the page.

Get your pencil ready and type…

```
TEST.RECURSION 1
```

First, you see the question, IS THIS RECURSION?, on the screen. So write a big 1 on your first piece of paper and put that piece off by itself.

Type YES and what happens? The question appears on the screen and :N becomes :N + 1 or 2. Write 2 on your second piece of paper and put that on the pile with your first piece…the one with the 1 on it.

Type YES again. What does :N become now? Write 3 on the next piece of paper and put that piece on the number pile. Do this again three more times, writing the new number for :N each time. Put each piece of paper on the top of your growing pile of papers.

Now, when you type NO, what happens on the screen? You see a list of numbers counting backward, right? From 7 back to 1.

Why? Look at the screen. There are seven questions shown there. You typed “yes” six times and “no” once. You typed 7 answers.

You should have two stacks of paper now.

You have some blank pages left over in one stack, pages 8 to 16. The other stack has the pages you numbered from 1 to 7. Each page has a big number written on it.

Now put the pages back in order from 1 to 16.

So what do you do? You put page #7 with the big number 7 on it, on top of page 8. You put page #6 on top of page #7, page #5 on top of page #6, and so on until you have all the pages back in a single stack again.

OK! Picture the memory in your computer like that stack of pages. Each time the procedure is run, another page is written to memory. When the procedure is stopped, Logo prints the pages.

Amazing Mazes

Remember the MAZE procedure, the example of tail-end recursion you read about earlier in this chapter?

```
TO MAZE :N
IF :N > 300 [STOP]
FD :N
RT 90
MAZE :N +10
END
```

Now look at this procedure.

```
TO AMAZE :N
IF :N > 120 [STOP]
AMAZE :N + 10
FD :N RT 90
END
```

Here's another example of “embedded” recursion. Will this procedure produce the same picture as the MAZE

procedure or will it be different? Try to picture what it will look like before you run it.

Think about how recursion works; about how it reads and acts on procedures. Let's start with :N as 50. This is how Logo reads the procedure the first time.

TO AMAZE 50	
IF 50 > 300 [STOP]	**Since 50 is smaller than 300, Logo goes to the next line.**
AMAZE 50 + 10	**AMAZE 50 becomes AMAZE 60 and starts again.**
FD 50 RT 90	**This line is held in memory.**
END	

Next, you have…

TO AMAZE 60	
IF 60 > 300 [STOP]	**Since 60 is smaller than 300, Logo goes to the next line.**
AMAZE 60 + 10	**AMAZE 60 becomes AMAZE 70 and is called.**
FD 60 RT 90	**This line is held in memory.**
END	

Next, you have…

TO AMAZE 70	
IF 70 > 300 [STOP]	Since 70 is smaller than 300, Logo goes to the next line.
AMAZE 80 + 10	AMAZE 70 becomes AMAZE 80 and is called.
FD 70 RT 90	This line is held in memory.
END	

Next, you have…

TO AMAZE 80	
IF 80 > 300 [STOP]	Since 80 is smaller than 300, Logo goes to the next line.
AMAZE 80 + 10	AMAZE 80 becomes AMAZE 90 and is called.
FD 80 RT 90	This line is held in memory.
END	

Each time the Logo runs the procedure, it never gets to the last line. So it writes that line on a "page" of the memory stack. It will keep going, writing pages for each line it did not run from AMAZE 90, 100, 110, and finally 300. Then it stops.

As Logo reads "puts the pages back in order," it sends the turtle FD 300, RT 90…then FD 290, RT 90…FD 280, rt 90…back to where she started…FD 50, RT 90.

So-o-o…are the pictures produced by MAZE and AMAZE the same? The end results look the same. The difference is that MAZE starts small and gets larger. AMAZE starts big and gets smaller.

Here's another example of embedded recursion to explore. This procedure produces a crazy drawing. Why? Can you tell without running it?

```
TO TOWER :SIZE
IF :SIZE < 0 [STOP]
SQUARE :SIZE
TOWER :SIZE - 10
SQUARE :SIZE
FD :SIZE
END

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

How would you change this procedure to make a better looking drawing?

Factorials

Another common use of embedded recursion is in calculating factorials. Factorial 5 is just another way of saying…

5 * 4 * 3 * 2 * 1 = 120

To write that in Logo…

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

About the only way you can make sense of this is to use the Recursive Pages approach like we did with AMAZE.

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

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

This is saved on the first page as the procedure is called again. This time, it reads…

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

This continues until the second line of the procedure reads…

```
IF 1 = 1 [OUPUT 1]
```

Then Logo reads back through the pages…

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

Spirals and Fractals

Do you know what they call the drawings that MAZE and AMAZE produce?

They draw what some people think are the prettiest drawings you can make with Logo. They're spirals.

MAZE and AMAZE produce square spirals…

```
FD 50 RT 90
FD 60 RT 90
FD 70 RT 90
FD 80 RT 90
FD 90 RT 90
```

…and on and on and on.

But what about other types of spirals?

Remember that procedure you wrote to draw any kind of shape?

```
TO POLYGON :SIDE :REPEATS
REPEAT :REPEATS [FD :SIDE RT 360 / :REPEATS]
END
```

OK…here's a challenge for you. Change this procedure into a recursive procedure that will draw the same picture. How about this?

```
TO POLYGON :SIDE :REPEATS
FD :SIDE
RT 360 / :REPEATS
POLYGON :SIDE :REPEATS
END
```

You can make this easier by changing the :REPEATS variable to an :ANGLE variable.

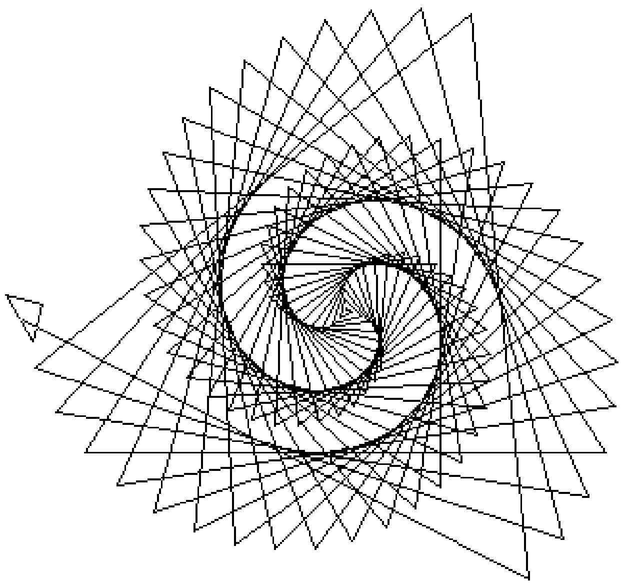
```
TO POLYGON :SIDE :ANGLE
FD :SIDE
RT :ANGLE
POLYGON :SIDE :ANGLE
END
```

OK…if you set :SIDE to 100 and :ANGLE to 120, you will send the turtle on a continuous trip around a triangle.

But that's no fun! Here's one way to do it.

```
TO POLYGON :SIDE :ANGLE :AMT
IF :SIDE > 300 [STOP]
FD :SIDE
RT :ANGLE
POLYGON (:SIDE + :AMT) :ANGLE :AMT
END
```

Now…what do you think the AMT variable does? Well, here's a drawing produced by this procedure. Does it help?



Play around with different numbers for the three variables of the POLYGON procedure. You'll be surprised at the things you can do.

What happens when you change 120 to 123?

```
POLYGON 1 120 3

POLYGON 1 90 5
```

How would you put more than one spiral on the screen at the same time?

Here is a procedure a young student developed. The goal was to create two spirals within the same procedure. What do you think of it…without running it, that is?

```
TO SPIRAL :N
IF :N > 100 [STOP]
FD :N RT 90
SPIRAL :N + 5
FD 200
IF :N > 100 [STOP]
FD :N RT 90
SPIRAL :N + 5
END
```

It seemed perfectly logical to this student that the turtle would draw the first spiral, move 100, and then draw the second one. What that student overlooked is that the recursive call sends the turtle back to the beginning. The result of this procedure is a mess.

But how could you straighten it out? This student overlooked a very valuable lesson about Logo.

You need to think in “chunks.”

Logo has to process one "chunk” of information at a time. In the procedures below, SPIRAL is one “chunk” of information. When you want to process more than one “chunk” of information, you need to add a procedure that will process your “chunks,” one at a time.

This is what SPIRALS does for you.

```
TO SPIRAL :N
IF :N > 100 [STOP]
FD :N RT 90
SPIRAL :N + 5
END
```

```
TO SPIRALS :N
SPIRAL :N
PU FD 200 PD
SPIRAL :N
END
```

This SPIRALS procedure draws two spirals. What would you have to do to make it draw four? Six? A variable number?

Rabbit Trail 20. String and Wire Art

Have you ever seen string or wire art?

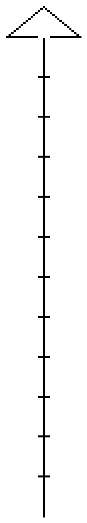
These are beautiful patterns created by wrapping colored string or wire around pins or small nails hammered into a felt-covered board. You can find some very colorful string or wire art kits at a local hobby store.

What's even more fun is to transfer the art patterns to the screen. There you can begin to see the relationships that work together to create the pattern.

First…let's start with a shoe box. Paint the inside of the top using flat black paint. This creates a dull background to show off your string patterns.

The next job is to create an even pattern that you will use to punch tiny holes evenly around the edge of the box top. You can do this very easily on the computer. Here's a recursive procedure that should be pretty easy for you by now.

```
TO PATTERN :DIST :MARKS
IF :MARKS = 0 [STOP]
FD :DIST MARK :DIST
MAKE "MARKS :MARKS - 1
PATTERN :DIS :MARKS
END

TO MARK :DIST
RT 90 FD :DIST / 10
BK :DIST / 5 FD :DIST / 10 LT 90
END
```



This procedure divides the task of drawing a pattern into easily understood chunks. The big chunk is drawing the pattern. The little chunk that is included but separate draws the actual marks.

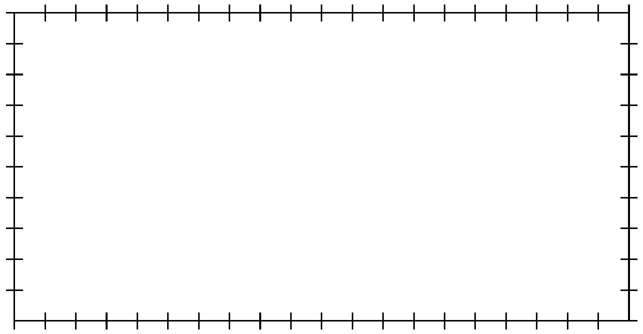
The variables let you set the number of marks (:MARKS) and the distance (:DIST) between them. For example, if you want to print 20 horizontal marks that are 25 turtle steps apart, type PATTERN 25 20 and press RETURN.

Print the patterns and cut them into narrow strips. Then tape them to the edge of your painted box top.

Curves From Straight Lines

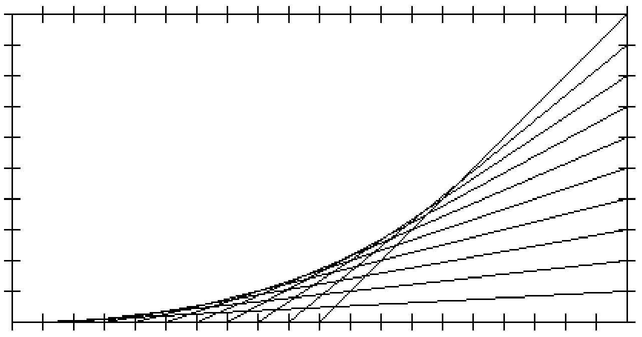
Now there are lots of things you can do. For one thing, you can use colored yarn and a needle to make curves from straight lines.

Here's a boxtop pattern.

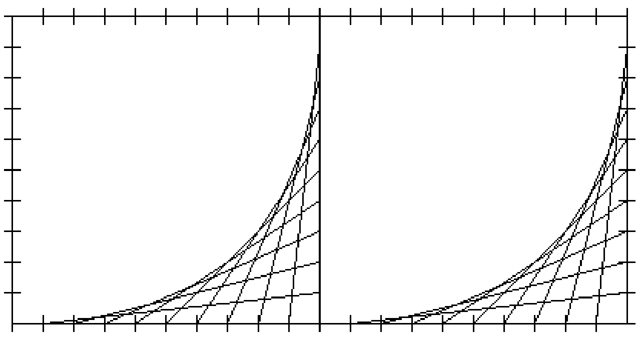


1. Start at the lower left hand corner.
2. Push the needle through the corner mark into the boxtop and then out through the mark at the lower right corner.
3. Move up to the first mark up the right side and push the needle from the outside into the boxtop.
4. Go to the first mark in from the left corner and push the needle from the inside to the outside of the boxtop.

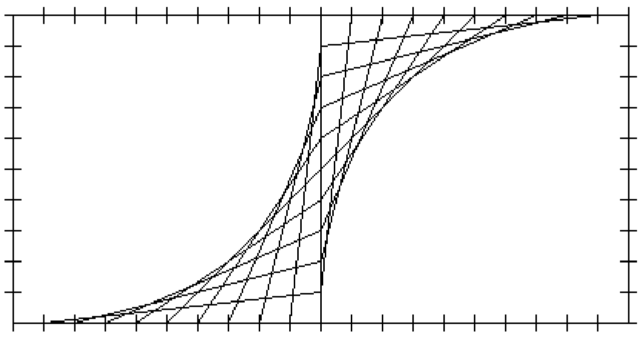
Soon you will have a pattern that looks like this, a curve made from straight lines.



There are lots of other patterns you can make. Why not try these?



Turn one of those upside down and look what you'll get.



There are all sorts of patterns you can make. If you want to dress them up a bit, try different colors of yarn for different parts of the design.

When you've used up all your old shoe boxes, you can try other designs on the Logo screen.

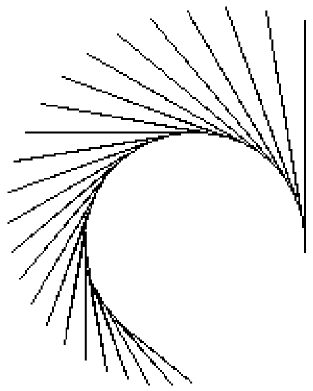
But, wait a minute! How are you going to do that?

Straightedge Curves

Well, let's start with a pencil, a piece of paper, and a straightedge. A ruler makes a good tool for this project.

1. Put the ruler on the paper in a vertical position…so that it's going straight up and down.
2. Draw a line from the bottom of the ruler up to about six inches and back to one-half inch from the bottom.
3. Hold your pencil in place and turn the ruler about 10 degrees.
4. Repeat steps 2 and 3 several times.

Does your drawing look something like this?



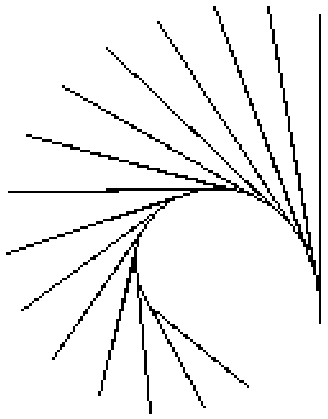
Not bad! Here's how you can do that on the computer.

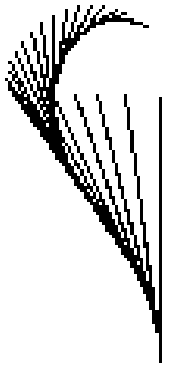
```
TO FANLEFT :DIST :ANGLE
IF :DIST < 0 [STOP]
FD :DIST BK :DIST - 10 LT :ANGLE
FANLEFT :DIST - 5 :ANGLE
END
```

What do you think would happen if you changed the angle *and* the distance each time a line was drawn?

```
TO LETSFINDOUT :DIST :ANGLE
IF :DIST < 0 [STOP]
FD :DIST BK :DIST - 10 LT :ANGLE
LETSFINDOUT :D - 5 :ANGLE + 2
END
```

If you can't see the difference here, try changing the number added to the ANGLE.





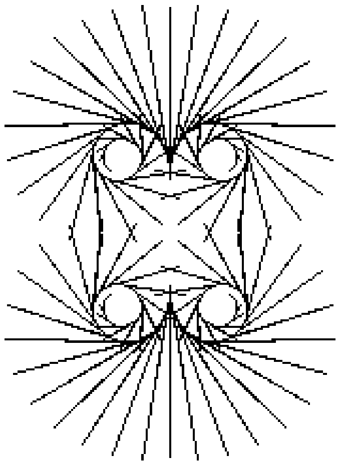
Here's a challenge for you.

How would you create this drawing?

Here's a hint. Take a look at the angles between the lines.

Here's a few more ideas to play with. How about a FANRIGHT procedure? What would happen if you combined them?

```
TO SWIRL :DIST :ANGLE
START1
FANLEFT :DIST :ANGLE
START1
FANRIGHT :DIST :ANGLE
START2
FANLEFT :DIST :ANGLE
START2
FANRIGHT :DIST :ANGLE
END
```



Note that the FANLEFT and FANRIGHT procedures were changed slightly to produce this drawing. Check the recursive statement in each procedure.

```
TO FANLEFT :DIST :ANGLE
IF :DIST < 0 [STOP]
FD :DIST BK :DIST - 5 LT :ANGLE
FANLEFT :DIST - 3 :ANGLE + 1
END

TO FANRIGHT :DIST :ANGLE
IF :DIST < 0 [STOP]
FD :DIST BK :DIST - 5 RT :ANGLE
FANRIGHT :DIST - 3 :ANGLE + 1
END

TO START1
PU HOME PD
END
```

```
TO START2
PU HOME RT 180 FD 50 PD
END
```

Another thing you might want to try is to add a START3 and START4 so that you have figures drawn at 90-degrees and 270-degrees.

There are lots of other things you can do with string and wire art. You'll find out more about it in the next chapter.

But before we leave recursion, you can't overlook the fun you can have with fractals.

Fractals

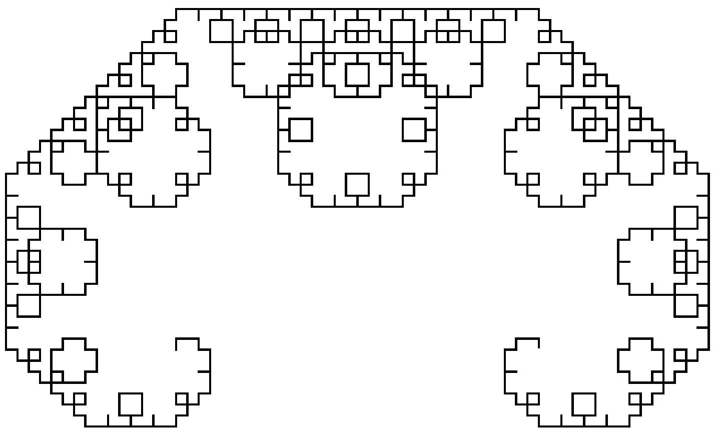
Fractals were once thought to be math monsters. No one could figure out what to do with them. But thanks to computers, we now know that these recursive monsters help make beautiful computer graphics.

Rather than try to explain fractals, let's look at how they work. Here's a procedure from the MSW Logo files.

```
TO C :SIZE :LEVEL
IF :LEVEL = 0 [FD :SIZE STOP]
C :SIZE :LEVEL - 1 RT 90
C :SIZE :LEVEL - 1 LT 90
END
```

If you look at the procedure, you see that :SIZE is the variable used by FD. :LEVEL is a bit confusing, so let's watch it work first. Type…

```
C 5 10
```



Wow…that's some pattern. Clear the screen and try…

```
C 20 3
```

OK…add this short procedure. Then you can see how the turtle builds such complicated pictures.

```
TO SEE
C :SIZE :LEVEL WAIT 50 CS
MAKE “LEVEL :LEVEL + 1
SEE
END
MAKE “SIZE 10
MAKE “LEVEL 0
```

Hey! Wait a minute…there's no procedure there at the end for those MAKE statements. How can that be?

Well…this is a new trick. You can tell the Logo what you want the variables to be without writing a procedure. It saves you the trouble of putting the variables in the procedure title.

Now…run the SEE procedure. You're watching fractals in action.

To help you figure out fractals, write the C and the SEE procedures on a piece of paper.

Change the WAIT variable to 100 or 150…long enough so that you can see the changes from one level to the next.

Another thing to do is change the LEVEL variable to 5 or 6…large enough so you can watch how the procedure really works. The higher the level, the more complex the picture.

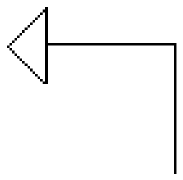
Hilbert, Snowflake, Dragons, and Things

You'll find some other fractal procedures on the Logo diskette…SNOWFLAKE, HILBERT, DRAGON, and others. Take a look at the DRAGON procedure.

```
TO DRAGON :SIZE :LEVEL
LDRAGON :SIZE :LEVEL
END

TO LDRAGON :SIZE :LEVEL
IF :LEVEL = 0 [FD :SIZE STOP]
LDRAGON :SIZE :LEVEL - 1 LT 90
RDRAGON :SIZE :LEVEL - 1
END
TO RDRAGON :SIZE :LEVEL
IF :LEVEL = 0 [FD :SIZE STOP]
LDRAGON :SIZE :LEVEL - 1 RT 90
RDRAGON :SIZE :LEVEL - 1
END
```

Can you see what the DRAGON procedure does…what a drawing would look like?

Here’s a picture for DRAGON 50 1.

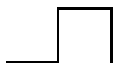


What would DRAGON 50 0 look like? Try it and see.

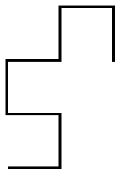
To see how DRAGON works, turn on TRACE. Type DRAGON 50 1 and press Enter. Then open up the Commander window (the Trace Editor in PC Logo), and look at the list of operations that Logo went through.

```
TO DRAGON 50 1
LDRAGON 50 1
END

TO LDRAGON
IF 1 = 0 [FD 50 STOP]
LDRAGON 50 1 - 1 LT 90
RDRAGON 50 1 - 1
END


TO RDRAGON
IF 1 = 0 [FD 50 STOP]
LDRAGON 50 1 - 1 RT 90
RDRAGON 50 1 - 1
END
```

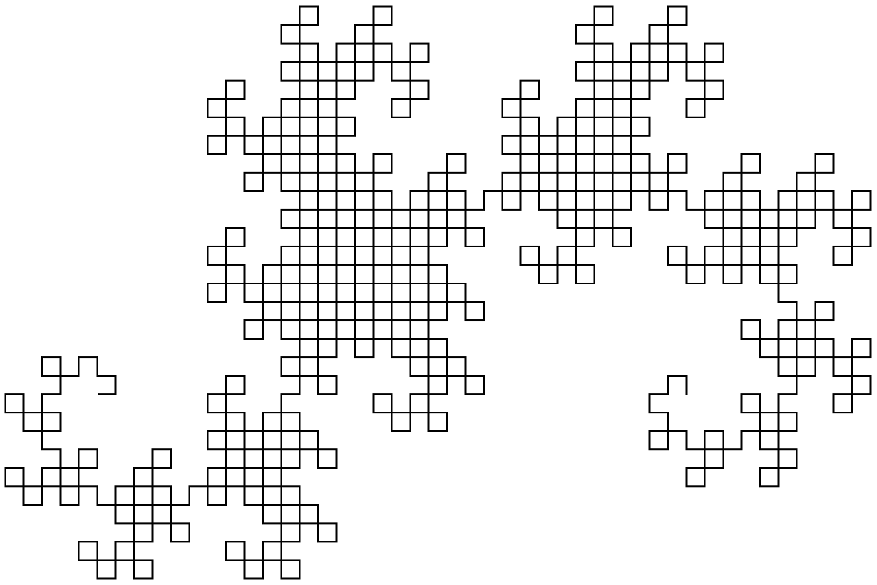
Now try Dragon 20 2…



DRAGON 20 3



DRAGON 10 10



If you have trouble understanding that list, use a pad of paper and start making piles for each recursive call…the same way you did before.

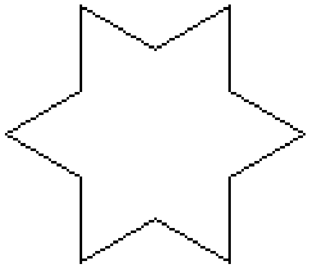
Now take a look at the SNOWFLAKE procedure.

```
TO SNOWFLAKE :SIZE :LEVEL
REPEAT 3 [RT 120 SIDE :SIZE :LEVEL]
END
```

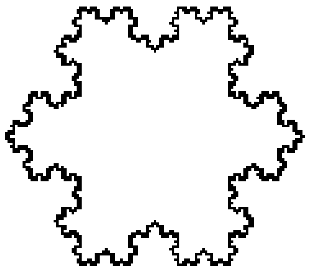
```
TO SIDE :SIZE :LEVEL
IF :LEVEL = 0 [FD :SIZE STOP]
SIDE :SIZE / 3 :LEVEL - 1 LT 60
SIDE :SIZE / 3 :LEVEL - 1 RT 120
SIDE :SIZE / 3 :LEVEL - 1 LT 60
SIDE :SIZE / 3 :LEVEL - 1
END
```

This procedure gets a bit more complex. What would SNOWFLAKE 50 0 look like…no fair trying it on the computer!

Here's a picture from SNOWFLAKE 100 1



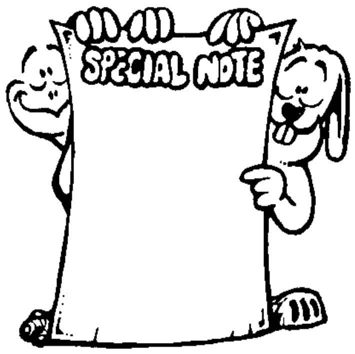
SNOWFLAKE 100 4



Now take a look at HILBERT.LGO. It's a bit more complex than SNOWFLAKE or DRAGON…a really good challenge.

For some more complex drawings, take a look at the Serpinski procedures shown on the next page.

```
TO GASKET :SIZE :LEVEL
; SERPINSKI GASKET: MANDELBROT P. 142
; TEST CASE: GASKET 100 2
CS HT PU BK 3 * :SIZE / 4 PD
LT 30 REPEAT 3 [GGEN :SIZE :LEVEL RT 120]
RT 30 PU FD :SIZE / 2 PD FILL
END
```



SPECIAL NOTE:
Hmmmm…something new. If you ever want to write notes in your procedures, type a semicolon. Logo ignores anything that is on the rest of that line. You can put the semicolon anywhere on the line…

```
FD 100 RT 90 ; that's a corner
```

```
TO GGEN :SIZE :LEVEL
IF OR (:LEVEL = 0) (:SIZE < 2) [FD :SIZE STOP]
(LOCAL "LEVEL1 "SIZE2)
MAKE "LEVEL1 :LEVEL - 1
MAKE "SIZE2 :SIZE / 2
GGEN :SIZE2 :LEVEL1
LT 120 REPEAT 3 [GGEN :SIZE2 :LEVEL1 RT 120]
PU RT 30 FD :SIZE2 / 2 PD SETFC [255 000 000] FILL
PU BK :SIZE2 / 2 LT 30 PD
RT 120 GGEN :SIZE2 :LEVEL1
END
```

```
TO CARPET :SIZE :LEVEL
; SERPINSKI CARPET: MANDELBROT P. 144
; TEST CASE: CARPET 100 2
CS HT PU HOME
SETH 45 BK :SIZE / 2 PD SETH 0
REPEAT 4 [CGEN :SIZE :LEVEL RT 90]
PU RT 45 FD :SIZE / 2 PD SETFC [000 000 255] FILL
END

TO CGEN :SIZE :LEVEL
IF OR (:LEVEL = 0) (:SIZE < 3) [FD :SIZE STOP]
(LOCAL "SIZE3 "LEV1)
MAKE "SIZE3 :SIZE / 3
MAKE "LEV1 :LEVEL - 1
REPEAT 3 [CGEN :SIZE3 :LEV1]
PU BK :SIZE BK :SIZE3 LT 90 FD :SIZE3 LT 90 PD
REPEAT 4 [CGEN :SIZE3 :LEV1 RT 90]
PU RT 45 FD :SIZE3 / 2 PD SETFC [255 000 000] FILL
PU BK :SIZE3 / 2 LT 45
BK :SIZE
PD REPEAT 4 [CGEN :SIZE3 :LEV1 RT 90]
PU RT 45 FD :SIZE3 / 2 PD SETFC [150 150 150] FILL
PU BK :SIZE3 / 2 LT 45
PU BK :SIZE3 RT 90 BK :SIZE3 RT 90 PD
END
```

There are many, many books on fractals…from the most basic level to the very complex. Take a look at some of these, especially those that deal with computer art and landscapes.