Leone Learning Systems, Inc.
Wonder. Create. Grow.

Leone Learning Systems, Inc.
237 Custer Ave
Evanston, IL 60202
Emal tj@leonelearningsystems.com

Phone 847 951 0127
Fax 847 733 8812

Axes

TJ Leone

November 2004

Introduction

This guide was written to provide further details on the AXES program in Philip G. Lewis's *Approaching Precalculus Mathematics Discreteley*.

Brian Harvey's web site (http://www.cs.berkeley.edu/~bh/) is a great source of additional information on Logo, especially the free downloadble PDFs for the three volumes of the second edition of *Computer Science Logo Style*.

XDIM and YDIM

If you've worked through Appendix B of *Approaching Precalculus Mathematics Discretely* and still don't understand how AXES works, I've written up another explanation. You should go through Appendix B before you read this.

To begin with, I'll go over the little procedures that need to be in place before AXES will work.

Here are XDIM and YDIM:

```
to xdim
op 480
end

to ydim
op 360
end
```

What do these procedures do? XDIM outputs the distance from the middle of the screen to the right edge of the screen. In my version of MSWLogo on my computer, I can execute `setpos [480 360]` and my turtle is still (just barely) in the top right quadrant of my screen. To put it another way, the four corners of my screen are approximately `[480 360]`, `[480 -360]`, `[-480 -360]` and `[-480 360]`.

MARK and CROSS

Next, we need mark:

```
to mark :scale
fd :scale cross :scale
end
```

What exactly is scale? It's the number of turtle steps between marks. Suppose I execute

```
axes 20 20
```

Then I'll have a mark every 20 turtle steps on both the x and y axes. The way this works is that MARK executes two commands. The first command is

```
fd :scale
```

which moves the turtle forward 20. The second command is

```
cross :scale
```

which actually makes the mark. This action is repeated as many times as needed to draw a line from the middle of the screen to one of the edges (top, right, bottom, or left).

How does CROSS work? Let's look at it:

```
to cross :scale
local "bar
make "bar :scale / 10
lt 90
fd :bar
bk 2 * :bar
fd :bar
rt 90
end
```

The first thing we do is make a local variable called :bar. There's some weird syntax here. We've seen variables before, like :scale in the CROSS procedure. The `make` command creates new variables from inside the procedure. The syntax of make is:

`make` *variable-name value*

MARK and CROSS (continued)

When we execute

```
cross 20
```

The value 20 replaces the name :scale everywhere in the procedure. Another way to think of it is that we make a variable named "scale and put the value 20 inside of it. Then, every time we write :scale, the value 20 is handed to Logo to be used as input to some command or operation.

When we execute

```
make "x 70
```

We make a variable named "x and put the value 70 inside of it. Then, every time we write :x, the value 70 is handed to Logo to be used as input to some command or operation.

Try it. Execute the make instruction above and then try

```
fd :x
show :x
```

Notice that you can't use a variable in the Input Box unless you `make` it first. For example, you can't use the :scale variable from the cross procedure *except inside the cross procedure*. The local command sets up variables so they can only be used locally, i.e., only inside the current procedure.

MARK and CROSS (continued)

OK, now let's take another look at CROSS:

```
to cross :scale
local "bar
make "bar :scale / 10
lt 90
fd :bar
bk 2 * :bar
fd :bar
rt 90
end
```

Because of the `local` instruction, the variable "bar can only be used inside of CROSS. If you try to use :bar in the Input Box after you execute CROSS, you will see something like this:

```
show :bar
bar has no value
```

If the `local` command hadn't been used, then "bar would have whatever value was last put into it, and you could use it in the Input Box.

The make command creates a variable named "bar and puts the value `:scale / 10` into it. So, if our scale is 20, then `:scale` will be 20 and `:bar` will be 2.

The rest of the procedure is pretty straightforward. The turtle turns left 90, goes forward 2, back 4, forward 2, then right 90. This draws a mark perpendicular to the current line.

XAX and YAX

Now we can look at XAX, which draws the x axis, and YAX, which draws the y axis:

```
to xax :xrep
repeat :xrep [mark :xscale]
bk product :xrep :xscale
end

to yax :yrep
repeat :yrep [mark :yscale]
bk product :yrep :yscale
end
```

In my example, we're starting with an x scale of 20 and a y scale of 20. In this case, for XAX, :xrep will be 24 (xdim / 20). For YAX, :yrep will be 18 (ydim / 20). For XAX, the repeat command will have us repeat 24 times the instruction `mark :xscale` where `:xscale` (for this example) is 20. This makes a line with 24 marks from the middle to the right or left side of the screen. After that, the `bk` instruction makes the turtle back up by 24 * 20 = 480 steps. YAX does similar work.

Notice that :xscale and :yscale are not inputs to XAX or YAX and neither are they defined with `make` inside these procedures. That's because they're defined (not locally) in AXES.

AXES

Last but not least, here's AXES:

```
to axes :xs :ys
make "xscale :xs
make "yscale :ys
local "xrep
local "yrep
make "xrep int (xdim / :xscale)
make "yrep int (ydim / :yscale)
repeat 2 [yax :yrep rt 90 xax :xrep rt 90]
end
```

Notice that there is no local command to set up the :xscale and :yscale variables. That's because we want to be able to use them in XAX and YAX. In fact, after you've run AXES, you'll find that you can use `:xscale` and `:yscale` in the Input Box as input to commands like `show` or `fd` or `rt`.

The number of marks to make in the x direction is determined found by xdim / :xscale. In our example, this is 480 / 20 = 24. The number of marks to make in the y direction for our example is 360 / 20 = 18. The `repeat` instruction tells us to perform the following set of instructions two times:

```
yax :yrep rt 90 xax :xrep rt 90
```

We need to do each of yax and xax twice because they are written to draw a line from the middle of the screen in the y or x direction to the edge of the screen.

We need to make sure that :xrep and :yrep are integers because they are used as the first input to the repeat command (you can't repeat something 4.7 times), so we used `int` to make sure that the result of xdim / :xscale and ydim / :yscale are integers. The text book uses the operation `integer`, which is not a legal operation in MSWLogo. Whenever you see `integer` in the text, use `int` for MSWLogo or Berkeley Logo.

If this still isn't clear, it might help to check out chapter 3 of Brian Harvey's Symbolic Computing book, which discusses variables. This book is available as a free PDF from http://www.cs.berkeley.edu/~bh/.

The Author

TJ Leone owns and operates Leone Learning Systems, Inc., a private corporation that offers tutoring and educational software. He has a BA in Math and an MS in Computer Science, both from the City College of New York. He spent two years in graduate studies in education and computer science at Northwestern University, and six years developing educational software there. He is a former Montessori teacher and currently teaches gifted children on a part time basis at the Center for Talent Development at Northwestern University in addition to his tutoring and software development work. His web site is http://www.leonelearningsystems.com.