

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

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

Phone 847 951 0127
Fax 847 733 8812

Math and Computers

Frequently Asked Questions

TJ Leone
June 2005



Introduction

This document contains questions I've received in the past about readings or problems in the course text. Questions are arranged by chapter. This document will be updated about once a week until August 2005.

You should attempt all problems before looking at the corresponding FAQs. I tried to indicate whether my answer gives a hint, a partial solution, or a complete solution to a problem.

Questions are written in italics.



Appendix A

Application A.8, page 495

I don't understand what they mean by "use Logo notation" in A.8. Do they mean to put it in the way Logo would understand it, or does Logo have a certain notation?

The easiest way to explain is to give examples. Here's a function written in algebraic notation:

$$k(x) = 7x + 3$$

Here's the same function in Logo notation:

```
to k :x
  op 7 * :x + 3
end
```

So "use Logo notation" just means "write the function as a Logo procedure".



An Elementary Linear Equation Solver, page 496

I really don't understand the "elementary linear equation solver" The exploration says that you need to write the program but I don't understand it to begin with.(p.496).

The problem statement for the elementary problem solver is confusing. The way to think about it is this:

In the A.7 problems, we found inverses for functions like $y = 2x + 1$ by switching x and y and then solving for y . For the current example, we would switch x and y to get $x = 2y + 1$ and then solve for y to get $y = (x - 1)/2$.

For the linear equation solver, the author wants us to start with $c = ax + b$, then switch x and c and solve for c . So we switch to $x = ac + b$, then, solving for c , we get $c = (x - b) / a$.

This follows nicely the pattern from A.7, but it makes things confusing when you're trying to figure out how to write the elementary linear equation solver. The inputs to the equation solver are the original a , b , and c from the equation $ax + b = c$.

An easier way to think about this problem is that we want to find x , so you really want to solve the equation for x . We can do this as follows:

$$\begin{aligned} ax + b &= c \\ ax &= c - b \quad (\text{subtracted } b \text{ from both sides}) \\ x &= (c - b) / a \quad (\text{divided both sides by } a) \end{aligned}$$

Once you have found x , you have your output for the SOLUTION procedure.

Let me know if this helps or if you still need more explanation for this.



Exploration A.1, page 496—partial solution provided

How do I apply the solution program from exploration a.1 to the problems from a.8?

Here's how I did it:

:: A.8.1 original problem: $3x + 5 = 12$

show solution 3 5 12

2.3333333333333333

show a 2.3333333333333333

12

:: A.8.2 original problem: $2(5x - 7) = -8$

:: which is equivalent to $10x - 14 = -8$

show solution 10 -14 -8

0.6

show c 0.6

-8

:: A.8.3 original problem: $(2x + 8)/5 = -3$

:: which is equivalent to $2x + 8 = -15$

show solution 2 8 -15

-11.5

show e -11.5

-3

:: A.8.4 original problem: $(7 - 5x)/2 = 40$

:: which is equivalent to $-5x + 7 = 80$

show solution -5 7 80

-14.6

show g -14.6

40



Creating a List, page 498

In the "Creating a List" section before Application A.10 on page 498, they talk about the procedure `inchestofeet`. When I type it in the way they have it in the book, it tells me that I have too much inside ()'s. I know how to fix this, but I don't understand why it doesn't work.

There are different versions of Logo. The author wrote the final versions of his programs in Object Logo for Macintosh. In Object Logo, `INTEGER` is valid Logo primitive.

In `MSWLogo`, the `INT` primitive is used instead. So, to get the program to run in `MSWLogo`, you need to write the `inchestofeet` program like this:

```
to inchestofeet :len
output list (int :len /12) (remainder :len 12)
end
```

It's very rare when a primitive used in the book does not exist in `MSWLogo`. If you suspect that you've found another one, the best way to find out is to test it in a simple command in the input box, like this:

```
show integer 18 / 2
I don't know how to integer
```

If `MSWLogo` tells you "I don't know how to..." it means you have found a word that isn't a primitive and hasn't been defined in a procedure written by you. In most cases, when you get an error like this, it means that your spelling of the word in the input box doesn't match the spelling of the primitive or the spelling of the procedure you defined.

The reason you got the error

```
too much inside ()'s in inchestofeet
```

has to do with the way that `MSWLogo` evaluates errors. For example, if I haven't defined any procedures `A` or `B`, I get:

```
show list (a b)
too much inside ()'s
show list (a)
I don't know how to a
show sum (a b)
too much inside ()'s
show sum (a)
I don't know how to a
```

⋮

Creating a List, page 498 (continued)

MSWLogo gets its errors from Berkeley Logo. Apparently, if you put more than one thing inside parentheses that Berkeley Logo can't evaluate, that's too much. If it only finds one that it can't evaluate, Berkeley Logo tells you that it doesn't know how to do it.

The moral is: if you're getting an error message that isn't very helpful, try finding the smallest piece of code that gives you an error and see if the message is more helpful.

⋮

Application A.11, #5, page 501—solution

How do you solve A.11, #5?

The way to think about changemaker is to think how you would process 92 cents:

$92 / 25$ is 3, so there are 3 quarters with REMAINDER $92 - 25 \cdot 3 = 17$ left over

$17 / 10$ is 1, so there is 1 dime with REMAINDER $17 - 10 = 7$ left over

$7 / 5$ is 1, so there is 1 nickel with REMAINDER $7 - 5 = 2$ left over

the last element in the output list is the leftover 2.

The way I solved the problem was to create PTQD

```
to ptqd :pennies
output se (first ptq :pennies) (ptd last ptq :pennies)
end
```

Then I wrote CHANGEMAKER like this:

```
to changemaker :pennies
output (se (butlast ptqd :pennies) (ptn last ptqd :pennies))
end
```

This was an especially tricky problem because it's hard to keep track of what you're supposed to be passing on and how to get everything into a list.

If my solution doesn't make sense to you, let me know.



Application A.12, #5, page 505—hint #1

For Application A.12-I don't understand what they want for the star program.

For the STAR procedure, you input a size and the procedure draws a five pointed star. I'm attaching a picture of a star I drew with a STAR procedure that I wrote.

The STAR procedure is going to look like the SQUARE and TRIANGLE procedures. I'll start it off:

```
to star :size
repeat ?? [fd :size rt ??]
end
```

You have to figure out what should replace the ??'s in procedure above.

To solve this problem, one thing to consider is, "How many sides does the star have?" The lines cross each other, so it might look like there are more lines than there really are. When most people draw this kind of star, they do it with five lines. Can you see how to do that? If not, let me know.

The next question is "How much of a turn should I make each time?" You can experiment with different numbers to see.

Here's something to notice about angles of turns: for the triangle, each turn is 120 degrees. When the turtle has completed his path around the triangle, he has turned $120+120+120=360$ degrees. This makes sense because by the time the turtle completes his path around the circle, he has turned all the way around and is facing in the same direction he was facing before he started.

Further, when the turtle draws the square, he turns $90+90+90+90=360$ degrees.

From this, you might guess that you can draw a pentagon with five turns of $360 / 5 = 72$ degrees, and you'd be right.

In fact, we can write a general POLYGON procedure like this:

```
to polygon :size :sides
repeat :sides [fd :size right 360 / :sides]
end
```

But how does that help with the star? A star is not a polygon.



Application A.12, #5, page 505—hint #1 (continued)

Well, we have the idea of a turtle turning all the way around to complete a path. But there is more than one way for a turtle to wind up at its original heading when it finishes drawing. For example, it could turn 360 degrees to the right or the left. It could also turn around more than once. For example, if a turtle made 2 complete turns, it would turn 720 degrees. If it made 3 complete turns, it would turn 1080 degrees, and so on.

Try this: Draw a five pointed star on a piece of paper. Use a small toy or cut out a small triangle to use for a turtle. Trace the star with the turtle, and see if you can tell how many times the turtle turns around before it completes tracing the path.

You can also try experimenting with different values in your STAR procedure to help guide your work.

I think I've given you enough hints to make a good stab at it. Let me know if you need more help.

⋮

Application A.12, #5, page 505—more hints

I still don't get how to write the STAR procedure.

Here are more hints for the STAR procedure.

To solve the STAR problem, you need to replace the ?? below with the right number:

```
to star
repeat 5 [fd 100 rt ??]
end
```

As discussed in earlier e-mail, the total amount of turning needs to be some multiple of 360 so the turtle can get back to its original heading.

In the case of the STAR, the total amount of turning is 5 times ??. If the total turning were 360, then the ?? would be $360 / 5 = 72$. But if we use 72, we get a regular pentagon:

```
to pentagon
repeat 5 [fd 100 rt 72]
end
```

What if the total turning is 720? 720 is a multiple of 360 because $2 * 360 = 720$. Then the ?? would be $720 / 5$. Try that. Does it work? Do you understand why the turtle has to end up at its original heading after turning through 720 degrees?



Application A.13, #9, page 507

Can you explain #9 in a.13 to me, I'm not sure I understand what the book wants me to do.

The object is to first find the number of turtle steps it takes to get from the center of the screen to the right edge. This is your X dimension. Then, you want to find the number of turtle steps from the center of the screen to the top of the screen. This is your Y dimension.

SETPOS has a list as its input. The list needs to have two numbers. The first number gives you an X coordinate, and the second number gives you a Y coordinate. To find the number of turtle steps from the center of the screen to the right edge, you could try SETPOS with different inputs like this:

```
SETPOS [100 0]
```

If this doesn't put the turtle at the right edge of the screen, you can try SETPOS [200 0] and so on until you get the turtle to the right edge of the screen. Suppose you get the turtle to the right edge of the screen with SETPOS [345 0]. Then your X dimension is 345 (and the whole screen is 690 turtle steps wide).

To get the Y dimension, you test with the second number in SETPOS's input list with things like SETPOS [0 100] and so on.

Chapter 1

Application 1.3, #2, page 11—debugging DISTANCE

I'm not sure what is wrong with my code. It tells me that it doesn't know what to do with the answer and the answer isn't right. I don't know how to fix it. I already fixed some things in it- by doing that thing with separating the parentheses like you told me- but it still doesn't work.

Can you tell me what I am doing wrong?

```
to distance :x1 :y1 :x2 :y2
output sqrt(sq(:x1 - :x2) + sq(:y1 - :y2))
End
```

```
to sq :num
output product :num :num
end
```

Like most of the procedures you've written so far, DISTANCE is an operation, which means it delivers information through the OUTPUT command. Any time you execute it, you need to tell Logo what to do with the output information. For example, you can do any of these without producing an error:

```
show distance 0 0 3 4
fd distance 0 0 3 4
setpos list 80 distance 60 140 10 20
```

Here's how I figured out why your DISTANCE procedure was giving the wrong answer. First, I started with a solution I know. I know that DISTANCE 0 0 3 4 should output a 5 (3 squared + 4 squared = 5 squared). You could have started with some other solution you know.

Next, I tried breaking down the formula in the DISTANCE procedure and executing each piece separately:

```
show sq (0 - 3)
9
show sq (0 - 4)
16
```

⋮

Application 1.3, #2, page 11—debugging DISTANCE (continued)

So far, so good. Now to try adding these pieces together:

```
show sq (0 - 3) + sq (0 - 4)
169
```

What happened? $9 + 16 = 25$, not 169. I tried something simpler:

```
show sq 3 + sq 4
361
```

More weirdness. Hmm... sometimes operations are done in the wrong order when you use infix operations like +, -, *, and / instead of SUM, DIFFERENCE, PRODUCT, and QUOTIENT. What if I use SUM instead of +?

```
show sum sq 3 sq 4
25
```

I works! So operations are happening in the wrong order. I should also be able to fix this with parentheses:

```
show (sq 3) + (sq 4)
25
show (sq (0 - 3)) + (sq (0 - 4))
25
```

Yup.

You should be able to figure things out from here. Good luck!



Application 1.4, #2, page 11

I have no idea how to do #2 in 1.4.

The first paragraph in the Application 1.4 section starting on page 15 says that we are extending the problem of the house and the barn.

For the house and barn problem in application 1.3, we found one point on one road (point P on the x-axis "road"). For the house and barn problem in 1.4, we need to find two points on two roads (point P on the y-axis "road" and point Q on the x-axis "road").

When you are tackling a tough problem, it's generally a good idea to try a simpler version first, and then refer to the simpler version when you're working on the tough version.

We solved a simpler version of 1.4 when we solved 1.3. In 1.3 (picture on page 14), we reflected the point B around the x axis to form the straight line AB', then found the intersection of AB' with the x axis to give us the point P. We know that the shortest distance between two points is a straight line and that the distance APB is equal to the distance APB'.

In the 1.4 version, we can reflect the point B about the x-axis and also reflect the point A about the y-axis to give us a straight line A'B' that intersects the y-axis at P and the x-axis at Q.

Do you see why that would help solve the problem? Does that get you started?

For more help, see <http://www.leonelearningsystems.com/Application1.4.pdf>.



Chapter 2

Application 2.1, #2, pages 24-25

In 2.1, my SPOT procedure doesn't give me a spot, I don't know what is wrong with it.

The author used a different version of Logo than the one we're using. In MSWLogo, no spot shows up if you give the commands `pd fd 0`. I wrote a different version of SPOT that draws a tiny circle. Notice that this version of SPOT needs "helper" procedures:

```
to spot
circlepoint pos 10
end

to circlepoint :point :radius
do.circlepoint :point :radius pos heading
end

to do.circlepoint :point :radius :startpos :startheading
pu
setpos :point
fd :radius
rt 90
pd
circler :radius
pu
setpos :startpos
seth :startheading
end

to circler :radius
repeat 360 [fd (pi * :radius) / 180 rt 1]
end
```

⋮

Application 2.1, #2, pages 24-25

In 2.1, does the SCALES procedure need to have :XSCALE and :YSCALE as inputs or does it automatically understand them to be what they were in axes?

You make the :XSCALE and :YSCALE variables and give them values when you execute the AXES procedure as written in Appendix B. Notice the first lines of the AXES procedure on page 522:

```
TO AXES :XS :YS  
MAKE "XSCALE :XS  
MAKE "YSCALE :YS  
...
```

so, if you execute

```
AXES 20 30
```

then type `show :xscale` in the Input Box and press the Execute button, you will see

```
show :xscale  
20
```

in the List Recall Box.

Try `show :yscale` and see what happens. These are the values of :XSCALE and :YSCALE that will be used in your SCALES procedure.



Chapter 3

Application 3.3, #1, page 50

In 3.3 I don't understand what the SCALEDIST procedure is supposed to do. Can you explain it to me?

The idea is that each mark on the X axis might be twenty turtle steps, but it might, for example, stand for 30 miles. Assuming that each mark on the X and Y axes were supposed to stand for 30 miles, you would draw the (no-wind speed and wind) vectors in the original airplane problem (figure on page 47) with these commands:

```
seth 60 pd fd scaledist 250 30
seth 120 pd fd scaledist 50 30
```

So SCALEDIST takes the mileage to be drawn (e.g., 250 for 250 mph) and the mileage scale (30 miles per mark) and outputs the number of turtle steps.

If the axes had a mark at each turtle step, then SCALEDIST could work like this:

```
to scaledist :miles :milescale
  op quotient :miles :milescale
end
```

But the number of turtle steps per mark is usually bigger than 1. For example, when you create your axes with AXES 20 20, the number of turtle steps per mark (:XSCALE) on both axes is 20.

So how do you make SCALEDIST work for any :XSCALE? What do you think?

Chapter 3 (continued)

Application 3.12, page 67

I'm confused by 3.12. What am I supposed to do?

For 3.12, the important thing to understand first is how TOWARDS works and then how DISTANCE is supposed to work. Here are some sample instructions using TOWARDS:

```
cs
show towards [300 400]
36.869897645844
fd 400
show towards [300 400]
90
```

When the turtle is in the center of the screen, it needs to SETH 36.869897645844 degrees to face towards the point [300 400]. When the turtle moves fd 400, its position is then [0 400]. From that position, the turtle needs to SETH 90 to face towards the point [300 400].

Now let's consider how DIRECTION is supposed to work. Here's an example using direction:

```
show direction [2 3.46]
60
```

We should get this result whenever the x and y scales are the same, no matter where the turtle is on the screen. When the x and y scales are different, we should get a different answer that turns the turtle towards the scaled point.

We saw in the section Vector Operations with Logo (pages 63-65 of *Approaching Precalculus Mathematics Discretely*) how LTR can be easier to write if you use LC and LC2. We can do something like that with DIRECTION.

The author suggests we use TOWARDS, so we can start by seeing how DIRECTION is different from TOWARDS (After all, if they're both the same, there's really no work to do). The differences are listed below.

⋮

Application 3.12, page 67 (continued)

1. TOWARDS depends on the location of the turtle. DIRECTION does not.
2. TOWARDS gives the clockwise angle from the y axis. DIRECTION gives the counterclockwise angle from the x axis.
3. TOWARDS doesn't know anything about the AXIS x and y scales. DIRECTION does.

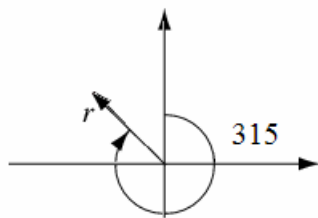
Let's start by making a version of DIRECTION called MAP.DIRECTION. This procedure always gives the direction TOWARDS would give if the turtle were at [0 0]. Here is an example of how it would work:

```
cs
show map.direction [300 400]
36.869897645844
fd 400
show map.direction [300 400]
36.869897645844
```

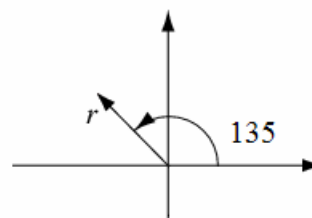
I'll leave it to you to figure out how to write this procedure.

This takes care of difference #1. What about difference #2? Our direction should measure its angle counter-clockwise from the x axis instead of clockwise from the y axis. Here's a picture of a vector r with its direction measured both ways:

Logo or map angle measurement



Math angle measurement



The same vector r has two different angle measurements, depending on the style of measurement. In Logo or map measurement, we measure angles clockwise from the y axis. In the measurement used in trigonometry and other areas of math, we measure angles counterclockwise from the x axis.

⋮

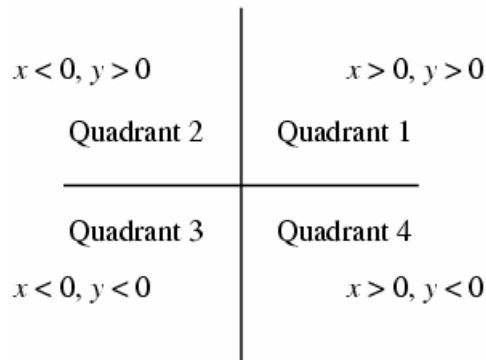
Application 3.12, page 67 (continued)

We could write a procedure called LOGO.TO.TRIG that converts map (or Logo) type headings into the “Math angle measurement” in the picture above.

Try to think for a bit about how you would write this procedure. Don’t spend more than five minutes or so if you’re feeling stumped. I’ll give you some hints to use as you get stuck.

Hint #1

This is one of those problems that can be solved more easily by breaking it into pieces. One way to break it into pieces is to use the way that axes divide up the Logo screen. When you draw the x and y axes on your Logo screen, you divide the screen into four different sections that mathematicians call quadrants. The first quadrant is between the positive x and positive y axes. The second quadrant is between the positive y and negative x axes. The third quadrant is between the negative x and negative y axes, and the fourth quadrant is between the negative y and positive x axes. Here’s a picture that shows the quadrants:



You can see that the quadrants are counted counterclockwise starting from the positive x axis.

Now try to solve the problem just for Quadrant 1. Notice that the math angle measurement is 90 when the Logo heading is 0, and the math angle measurement is 0 when the Logo heading is 90. How would you convert all Logo headings between 0 and 90 into math headings?

After you’ve solved the problem for Quadrant 1, try the other quadrants.

⋮

Application 3.12, page 67 (continued)

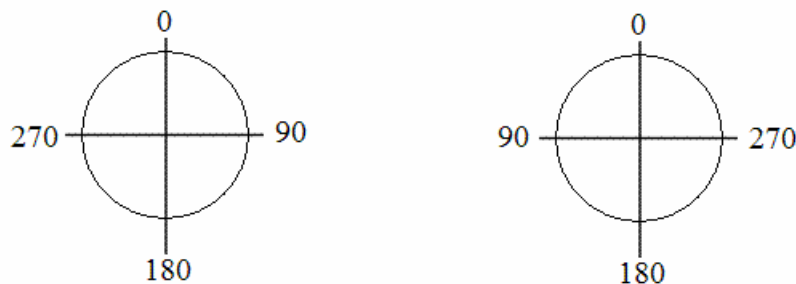
Hint #2

You don't necessarily need to break up the screen into quadrants in order to solve the problem. We could think of changing angle measurement by changing the screen orientation. Consider the figure below:



In this red and green circle, the green sector represents a right turn of 225 degrees, and the red sector represents a left turn of 135 degrees. We can see that either turn will take us to the same place. We can also see that if LEFT x and RIGHT y take us to the same place, then $x + y = 360$. In other words, every RIGHT turn of x degrees is equal to a LEFT turn of $360 - x$ degrees.

So we can change a clockwise measurement into a counterclockwise measurement by subtracting each measurement from 360:



In the figure above, $360 - 90 = 270$, $360 - 180 = 180$, and $360 - 270 = 90$, so $360 - x$ seems like a pretty good way to convert measurements. You will notice, of course, that $360 - 0 = 360$, not 0, but we can fix this in our procedure by using REMAINDER x 360, since REMAINDER x 360 = x whenever x is between 0 and 360 and REMAINDER x 360 = 0 when $x = 360$.

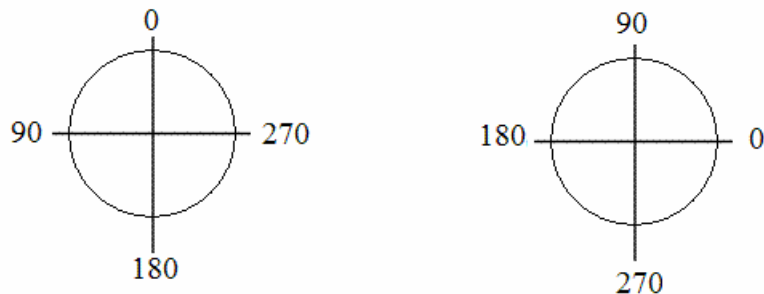
So we could write a procedure called CLOCKWISE.TO.COUNTERCLOCKWISE that accepts a heading and outputs the heading in counterclockwise degrees from the y axis. Try building that procedure before moving on. If you have problems with REMAINDER, see Hint #4.

⋮

Application 3.12, page 67 (continued)

Hint #3

After you've flipped your screen orientation around from clockwise to counterclockwise, you need to rotate it like so:



Notice that in this case we have $0 + 90 = 90$, $90 + 90 = 180$, $180 + 90 = 270$, $270 + 90 = 360$, and $\text{REMAINDER } 360 \ 360 = 0$. In other words,

```
REMAINDER SUM 0 90 360 = 90
REMAINDER SUM 90 90 360 = 180
REMAINDER SUM 180 90 360 = 270
REMAINDER SUM 270 90 360 = 0
```

So, in general, we can use `REMAINDER SUM :C 90 360` to rotate the angle measure `:C` by 90 degrees. Don't forget that `:C` must be the counterclockwise measurement that you generated in Hint #2, not the regular clockwise Logo heading.

Hint #4

You might have noticed a problem with `REMAINDER`. It only works on whole numbers. In other words, you will get an error message when you use `REMAINDER` with numbers that have decimal parts:

```
show remainder 537.4 360
remainder doesn't like 537.4 as input
```

This is problem, since `TOWARDS` usually outputs a heading with a decimal part. Here's a version of `REMAINDER` you can use with numbers that have a decimal part:

```
to rremainder :dividend :divisor
if less? :dividend 0 [op minus rremainder abs :dividend :divisor]
if less? :divisor 0 [op rremainder :dividend abs :divisor]
if less? :dividend :divisor [op :dividend]
op remainder difference :dividend :divisor :divisor
end
```



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>