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

Functions

TJ Leone

October 2004

Introduction

This unit introduces the algebraic idea of functions and shows how functions are written in Logo.

You may remember that in Logo an *operation* is a procedure that takes zero or more inputs and produces an output. You can think of functions as operations that have at least one input and behave in a predictable way.

In this guide, we will look at functions that convert units of distance, volume, and temperature, and formulas to calculate area and find the distance traveled from the rate traveled and time traveled. We will also consider composite functions.

Variables

In the document http://www.leonelearningsystems.com/IntroToMSWLogo.pdf, we saw how to write Logo procedures. If you don't remember how to do this, you should review the section of this document called "Writing Procedures".

The first procedure we wrote was triangle:

```
to triangle
     cs
     fd 100
     rt 120
     fd 100
     rt 120
     fd 100
     rt 120
end
```

This procedure doesn't need any inputs. It draws an equilateral triangle whose sides are 100 turtle steps long. What if we wanted to draw a triangle with sides of 50 steps? Or 200 steps? Or 80, or 150, or 75? We could write five more procedures, changing the input to fd in each one, or we could change the triangle procedure to take an input:

```
to triangle :side
     cs
     fd :side
     rt 120
     fd :side
     rt 120
     fd :side
     rt 120
end
```

Now, each time we invoke triangle, we can give it an input. See what happens when you try these:

```
triangle 100
triangle 20
triangle 50
triangle 200
```

Variables

The first line of the procedure

```
to triangle :side
```

says “I am writing a procedure named triangle. Triangle needs one input. In this procedure, whenever I say “`:side`” I mean the input.”

So, if you invoke the procedure like this:

```
triangle 70
```

then Logo evaluates the line

```
fd :side
```

as

```
fd 70
```

Inches, Feet, and Yards

I'm sure you know that there are 12 inches in a foot. So if we know that a box is three feet wide, we can find the number of inches in the width by multiplying three by 12:

3 feet times 12 inches per foot = 36 inches

Write this procedure:

```
to feet.to.inches :feet
     op product 12 :feet
end
```

Does this procedure define a command or an operation? What does it do? If you're not sure, the document http://www.leonelearningsystems.com/IntroToLogo_MSW.pdf (for MSWLogo) or http://www.leonelearningsystems.com/IntroToLogo_UCB.pdf (for Berkeley Logo) contains all the Logo you need to know to understand the procedure. If you're not sure how to create or save the procedure, see http://www.leonelearningsystems.com/IntroToMSWLogo.pdf or http://www.leonelearningsystems.com/IntroToBerkeleyLogo.pdf.

It should be clear to you that the `feet.to.inches` procedure defines an operation (If it isn't clear, then look again at the documents mentioned above).

Once you've saved your procedure, try executing it. In MSWLogo, this means typing the procedure name in the Input Box along with any input and pressing the Enter button. In Berkeley Logo, you type at the Logo prompt instead of the Input Box.

```
show feet.to.inches 3
```

What is the output of the `feet.to.inches` operation when the input is 3? Try running `feet.to.inches` with other inputs. See if you can predict what the output will be before you run the procedure.

In mathematics texts, the `feet.to.inches` function might be expressed like this:

$f(x) = 12x$

The function is *f*, the input is *x* and the output is 12*x* (which means 12 times *x*). In algebra, single letters are usually used to name functions and inputs, so you do less writing. In Logo, we try to give functions and inputs more meaningful names.

Now, before you read any further, try to write a procedure to convert yards to feet. Just in case you forgot, a yard has three feet. Test it to see if it works.

Inches, Feet, and Yards (continued)

You should have written something like this:

```
to yards.to.feet :yards
     op product 3 :yards
end
```

In an algebra book, this function might be written like this:

$g(x) = 3x$

Let's look more closely at the difference between a function definition written in Logo and the same function definition written in math textbook style.

First of all, a function definition in Logo is a procedure. You actually use it to run instructions like

```
show yards.to.feet 1
```

Function definitions in Logo and standard algebra

A function definition in a math textbook is a general description. It tells you how to calculate an output using the function's input, but it doesn't do the calculation for you.

As we said earlier, people who write Logo programs usually try to use meaningful names for functions and inputs. The Logo `yards.to.feet` function above could have been written like this:

```
to g :x
     op product 3 :x
end
```

The operations `g` above will always give the same output as `yards.to.feet` whenever they are both given the same input. Why?

Another difference between Logo function definitions and textbook ones is the way they separate the function names and input from the recipe that tells you how to carry out the function.

In a textbook, we first write the function name (for example, *g*). Next to the function name, we put the input to the function in parentheses. We use the equal sign (=) to separate the function and the input from the recipe that tells how to carry out the function (for example, 3*x*, which tells us "Multiply the input by 3").

In a Logo procedure, we write the word "`to`" in front of the function name. There is always a space between the word "`to`" and the function name. After the function name there is another space before any input. In Logo, the recipe for "multiply 3 times *x*" is `product 3 :x`. This product is input to op, which tells Logo to output the result of 3 times *x*. Logo procedures always end with the word `end`.

In Logo we put a colon in front of variable names. In standard algebra, we put variable names in italics.

Solved Problems

From now on, each unit for this course will have a section like this one, named Solved Problems. The Solved Problems section gives problems together with their solutions to give you an idea of how these problems are done. The Solved Problem section will also give some explanation of how the problem was solved.

You should try to solve these problems yourself before you look at the solutions. This will help you make sense of the solutions, and prepare you for exercises where the answers are not given.

1. *Problem:* Write a command called `square` that needs one input. The procedure draws a square. The input tells the procedure how long to make each side of the square.

   Solution: We begin a procedure definition with the keyword "`to`" followed by the procedure name, followed by any input to the procedure. We are told that our procedure name should be `square` and that it needs one input, which is the length of each side of the square.

   Whatever we name the input, we need to put a colon (:) in front of the name. The colon tells us that the name is the name of an input. Let's name the input `:side`.

   This gives us the first line of our procedure:

   ```
   to square :side
   ```

   Now, we need to figure out how to draw the square. The way to think about it is to imagine the turtle (the little triangle on the screen) walking around the edge of a square. It walks the length of one side, then turns, walks the length of the other side, turns again, walks again, and keeps going until it is back where it started.

   How do we get the turtle to move forward for a length of `:side`? We use the command

   ```
   fd :side
   ```

   Once the turtle has walked the length of one side, what does he have to do next? He has to turn to face the next corner. He can do this with a turn of 90 degrees, so we give the command:

   ```
   rt 90
   ```

Solved Problems (continued)

Now we just need to keep walking and turning until we get back to where we started. How many times to we walk and turn? How many sides and corners does a square have? The complete procedure is shown below.

```
to square :side
     fd :side
     rt 90
     fd :side
     rt 90
     fd :side
     rt 90
     fd :side
     rt 90
end
```

You can type this procedure into your Logo editor and try executing it.

If you get this error message:

```
I don't know how to square
```

Then maybe you forgot to save the procedure into your file. Use File/Save and Exit from the editor window if you are using MSWLogo or use Ctrl-X Ctrl-W to write the buffer if you are using Berkeley Logo.

It could also be that you spelled square wrong, either in your file when you defined your procedure, or when you typed the instruction into the Input Box (MSWLogo) or at the Logo prompt (Berkeley Logo).

Another error message you might see is:

```
not enough inputs to square
```

You would see this error message if you defined square as above but invoked it without any inputs like this:

```
square
```

As written, the square procedure needs one input, which gives the length of each side of the square:

```
square 50
```

Solved Problems (continued)

2. *Problem:* Write an operation called `cups.to.ounces` that needs one input. The operation converts cups into ounces. In case you don't remember, there are eight ounces in a cup. When you invoke the procedure with inputs shown below, you should see the outputs shown below:

```
show cups.to.ounces 3
24
show cups.to.ounces 1
8
show cups.to.ounces 7
56
```

Solution: Like the `square` procedure above, this procedure needs one input, so the first line of the procedure starts with "`to`", then the procedure name, then the name of the input (which starts with a colon). So we can write it like this:

```
to cups.to.ounces :cups
```

However, there is an important difference between this procedure (`cups.to.ounces`) and the `square` procedure. The `square` procedure is a *command*. It *gives Logo an order* ("Draw a square!"). The `cups.to.ounces` procedure is an *operation*. It must *answer a question* ("How many ounces is it?").

When we write an operation, we use the `op` command to tell Logo, "take the information that follows and make it the output of the operation." So we know there has to be an `op` command in this procedure.

But what should we output? Well, a cup is eight ounces, so whatever input we get for the number of cups, we should multiply that number by eight. The Logo operation for multiplication is `product`. Here's the whole operation:

```
to cups.to.ounces :cups
     op product 8 :cups
end
```

Try creating this procedure in Logo. If you get the error

```
I don't know how to cups.to.ounces
```

make sure you saved the file and spelled the procedure name correctly when you defined it and when you invoked it.

Solved Problems (continued)

Did you get an error message like the one below?

```
You don't say what to do with 32
```

I got this error message when I tried to invoke the procedure like this:

```
cups.to.ounces 4
```

The problem here is that `cups.to.ounces` is an operation, not a command. Operations answer questions. They provide information. They don't give orders. They don't tell Logo what to do.

When Logo evaluated the procedure, it recognized that `cups.to.ounces` need one input. It saw the 4 and gave it to the `cups.to.ounces` procedure. The `cups.to.ounces` procedure multiplied 4 by 8 and output the result (32).

But when Logo got the output, the 32, Logo got confused. What is Logo supposed to do with 32? Should Logo move the turtle forward 32? Turn the turtle right 32? Show the number 32? I didn't say. So Logo sent back the message:

```
You don't say what to do with 32
```

How do we solve this problem? We give Logo something to do with the output of `cups.to.ounces`. For example, we could invoke the procedure like this:

```
show cups.to.ounces 4
```

Solved Problems (continued)

3. *Problem:* Write the following standard algebra function as a Logo operation:

$f(x) = 4x + 7$

Solution:

The expression $f(x)$ tells us that the function name is f and the input is x. In Logo, input names must start with a colon (:), so the first line of our Logo function definition is:

```
to f :x
```

After the equal sign, we see the expression $4x + 7$. This expression means "Multiply x by 4 and then add 7". In Logo, this is written as

```
sum product 4 :x 7
```

See "An explanation from Brian Harvey" in the document *Logo* (http://www.leonelearningsystems.com/IntroToLogo_MSW.pdf) if this is not clear.

Since we are writing an operation (not a command), we need the `op` command to tell Logo to output the result of the sum. Below is the entire procedure:

```
to f :x
    op sum product 4 :x 7
end
```

Try creating this procedure in Logo and invoking it. If you get any error messages when you invoke it, see the comments on error messages in the `cups.to.ounces` example above.

Solved Problems (continued)

4. *Problem:* Write the following Logo operation as a standard algebraic function definition:

```
to quarts.to.pints :quarts
    op product :quarts 2
end
```

The first line of the procedure definition tells us that the name of the procedure is `quarts.to.pints`. It also tells us that the input to `quarts.to.pints` is the variable `:quarts`.

In standard algebra, we use single letters for function names and variable names. We could use the letter *f* to stand for the function `quarts.to.pints`, and the letter *x* to stand for the variable `:quarts`. So *f(x)* defines our function name together with its input.

The output of the Logo operation is the product of the 2 and the input. In our algebra function, we would write this as $2x$. In standard algebra function definitions, an equal sign separates the function name and input from the directions that tell how the function is carried out. So our final algebraic function definition is:

$f(x) = 2x$

Supplementary Problems

In addition to Solved Problems, I will provide Supplementary Problems for each unit. The Supplementary Problems have more problems for extra practice, but no explanations are included as they are for the Solved Problems.

1. *Problem:* Write a procedure to draw a rectangle whose long sides are twice as long as its short sides. Tell whether the procedure is a command or an operation. Give three examples of how to invoke the procedure.

   Solution:

```
to rectangle :shortside
    fd :shortside
    rt 90
    fd product :shortside 2
    rt 90
    fd :shortside
    rt 90
    fd product :shortside 2
    rt 90
end
```

   The procedure is a command. Here are some ways the procedure could be invoked:

```
rectangle 50
rectangle sum 4 product 8 20
rectangle difference 90 7
```

Supplementary Problems (continued)

2. *Problem:* Write a procedure to convert gallons to quarts (there are four quarts in a gallon). Tell whether the procedure is a command or an operation. Give three examples of how the procedure could be invoked.

   Solution:

```
to gallons.to.quarts :gallons
    op product 4 :gallons
end
```

   The procedure is an operation. Here are some ways the procedure could be invoked:

```
show gallons.to.quarts 3
fd gallons.to.quarts 9
show gallons.to.quarts sum 7 product 10 8
```

3. *Problem:* Write the following standard algebra functions as Logo operations:

   (a) $f(x) = x + 1$
   (b) $g(x) = x \div 2$
   (c) $h(x) = 3x - 7$

   Solutions:

(a)
```
to f :x
    op sum :x 1
end
```

(b)
```
to g :x
    op quotient :x 2
end
```

(c)
```
to h :x
    op difference product 3 :x 7
end
```

Supplementary Problems (continued)

4. *Problem:* Write the following Logo operations as standard algebraic function definitions:

(a)
```
to miles.to.yards :miles
    op product 1760 :miles
end
```

(b)
```
to inches.to.feet :feet
    op quotient :feet 12
end
```

(c)
```
to fahrenheit.to.celsius :fahrenheit
    op product quotient 5 9
        difference :fahrenheit 32
end
```

Solutions:

(a) $f(x) = 1{,}760x$

(b) $g(x) = x \div 12$

(c) $h(x) = \left(\frac{5}{9}\right)(x - 32)$

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.