THE WORLD OF LOGO

Joseph Rice
University of Arizona
jjrice@email.arizona.edu

Chris Rehling
University of Arizona
crehling16@email.arizona.edu

April 21, 2020

ABSTRACT

1 Introduction

2 History

Logo was created in 1967 by Wallace Feurzeig, Seymour Papert, and teams from Bolt, Beranek, and Newman as a tool for learning. Logo is still around today and "is widely known for its use of turtle graphics" (WIKI) which is a feature that allows users to control a turtle with movement and drawing commands to draw shapes, designs, and pictures.

3 Control Structures

Though it was designed to be easy for programming novices, Logo does not skimp on its features. Among these features are many ways to modify and control program flow. The basics are, of course, included: for-loops are present in both the simple **repeat** command and the more elaborate **for**, as is **while** and its variants **do-while** and **do-until**. An equivalent to C's **switch** is included as **case**, and **goto** and **wait** are present for those who want to fine-tune performance. It's good to have the basics, but Logo's uniqueness shines through in some of its more obscure control-flow operators.

One of these areas where Logo puts more emphasis is in its conditionals. While the basic **if** and **ifelse** are standard, the **test** command offers a more robust form of conditionals based on logical operations. It stores the result of the given expression as 0 (false) or 1 (true). From there, the user can reference the result of the test with the commands **iftrue** and **iffalse**, allowing certain commands to execute based on the test result. This trio of commands offers better performance and more flexibility by automatically storing and reusing the result.

Being created as a dialect of Lisp, Logo naturally includes some list-processing control-flow functions. It groups these operations together under the concept of 'template iteration', where a template is some expression or procedure that uses the elements of the provided list(s). The **map** and **filter** commands will be familiar to functional programmers– respectively applying a procedure to all elements, and reducing a list to fewer elements based on given criteria. [1]

4 Data Types

Data types vary between different versions of Logo. The main version of Logo used for the study of this paper is the UCBLogo. UCBLogo has three data types: word, list, and array.

4.1 Words

WORD is the data type used for variable names. The UCBLogo manual gives a brief explanation about the data type, "In traditional Logo syntax, a non-numeric word typed without punctuation represents a request to invoke the procedure

named by that word. A word typed with a preceding quotation mark represents the word itself. For example, in the instruction

PRINT FIRST "WORD

the procedures named FIRST and PRINT are invoked, but the procedure named WORD is not invoked; the word W-O-R-D is the input to FIRST." [1]. The procedure PRINT is the command used in Logo to output values passed into the command, while the procedure FIRST is a variable that holds the string "WORD". In Logo, to access the contents of a WORD, a colon, :, is used in front of the variable. For example, to get the string "WORD" from the variable FIRST the colon and variable are used together like, :FIRST, and the value "WORD" is passed back. That is, for example, if we took the procedure from the example above and replaced, FIRST "WORD, with :FIRST, the string "WORD" is output to the console in which the command was run.

4.2 Lists

Logo, like many other languages, uses the data type LIST to hold elements of varying types. Unlike other languages, Logo lists are able to hold variables of differing types such as WORDS, ARRAYS, and other LISTS. For example, the list

[[MAKE "FIRST 1 PRINT SUM FIRST FIRST] "hello "world "!]

contains an instructionlist with the procedures MAKE, PRINT, and SUM and the words "hello, "world, and "!". An *instructionlist* is a type of list used in many of Logo's commands, that is used to feed procedures into the command one after the other.

There are many commands in Logo to insert and extract data from a list. For example the command FIRST, which outputs the first element of a list, LAST, which outputs the last member of a list, and BUTFIRST, which outputs a list all all the elements except for the first element. These commands are what give Logo LISTS and advantage over Logo ARRAYS.

4.3 Arrays

Arrays in Logo are created using the following command

ARRAY size

which "outputs an array of size members (must be a positive integer), each of which initially is an empty list" [1]. Members in an array are numbered, "the first member of the array is member number 1 unless an origin input (must be an integer) is given, in which case the first member of the array has that number as its index. (Typically 0 is used as the origin if anything.)" [1]. Like lists there are commands to access members of an array. To access a member from an array command ITEM (shown below) is used.

ITEM index thing

The command outputs the *indexth* member in the array. And the command SETITEM (shown below) is used to change the value of the *indexth* item in the array with a given value.

SETITEM index array value

The commands shown above are the only commands that can access and change individual items in the an array. Compared to LISTS, where LISTS can be accessed and manipulated with multiple commands, ARRAYS are limited in their usefulness.

5 Subprograms

Compared to other more feature-heavy languages, Logo offers little in the way of subprogramming features. Java-esque packages are missing, and the language has no notion of a class. Logo does, however, offer some modularity in the form of imports and functions. Functions in Logo are easy to declare, notably lacking any requirement to set a return type due to the language's very loose typing. The basic syntax for a function is as follows:

```
TO function1 print [Hello World] END
```

TO and END act as brackets around a block of code would in other languages, but the use of English keywords make their purpose more obvious–which can be helpful to novice programmers. One can also include variables for use in the function by including words after the function name; to expand on the previous example, one could print anything they wanted by coding a function.

```
TO function1 :var print :var END
```

and then calling that function with "function1 [word]". This example is of course useless in practicality, but real functions can include much more code as well as multiple variables. One thing of note about Logo functions is that it draws a distinction between functions with side-effects, such as moving the turtle or printing, and those without. Functions with side-effects but no output are called *commands,* while functions that compute an output with no side-effects are called *operations* [2, 17]. This is a small distinction, but important given how turtle graphics is primarily done through commands.

The other subprogramming functionality provided by Logo is a basic form of modules in the SAVE and LOAD commands. The two do exactly what they say; one can save all created functions to a file with SAVE, and later LOAD that file into the current workspace–allowing that file's functions to be used without needing to write them again [1, 62-63]. While not as fully-developed as classes or packages, since the code does have to be re-interpreted each time, the two allow Logo programmers to break down their codebases into manageable portions.

6 Summary

References

[1] Brian Harvey. *Berkeley Logo User Manual*. University of California Berkeley, 1997.

[2] Brian Harvey. *Computer Science Logo Style: Symbolic Computing*, volume 1. Massachusetts Institute of Technology, 2nd edition, 1997.