THE WORLD OF LOGO

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

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

April 30, 2020

ABSTRACT

Logo is a functional programming language developed in the 1960s and 1970s by MIT researchers to be a learning tool for novice programmers. As a dialect of Lisp, it performs excellently at list processing-oriented tasks, and its simple syntax helps beginner coders ease into learning to program. However, Logo's 'killer app' is not list-processing, but instead its 'turtle graphics' feature. In this style of graphics development, users code a sequence of commands for a cursor to draw on a canvas. The cursor, known in Logo parlance as a turtle, can move forward or backwards, turn a certain number of degrees, or jump to any point on the canvas—just to name a few commands. When used in combination with Logo's other features, programmers can create complex graphics out of these simple commands.

1 Introduction

This paper provides an overview of the programming language Logo. The first section of this paper provides a brief history of Logo. This section provides information on the purpose of the Logo language, as well as the original developers. The next section provides a brief description of the control structures available in Logo along with the syntax for properly formatting a control structure. The fourth section, Data Types, covers the three data types in Logo: Word, Array, List. The fifth section, Subprograms, provides an overview of procedures that can be created and used in the Logo command window.

2 History

Logo began in the mind of Seymour Papert, an MIT researcher who was called "among the first to recognize the revolutionary potential of computers in education" by the MIT Media Lab [5]. After founding the AI Research lab at MIT, he joined with researcher Wallace Feurzeig and his team to create Logo. The Logo team intended for the language to be a learning tool, with four features supplementing that goal: "modularity, extensibility, interactivity, and flexibility," according to the Logo Foundation. These early Logo programmers brought their creation to local schools to test its use as a learning language. In lieu of a digital cursor like modern Logo interpreters use, they instead built a robotic turtle to execute commands—hence the moniker of "turtle graphics" [1].

As Logo development continued through the late 20th century, its applications in learning grew. Apple and Texas Instruments both licensed Logo for use on their personal computer products. When those products came to schools, Logo came with them. Papert encouraged further adoption of Logo with his book Mindstorms, containing plenty of inspiration for teachers on how to integrate programming with Logo into their classrooms. Branches of Logo were created with new features, such as LogoWriter and its word-processing capability. One of the most successful branches was LEGO Logo, where coders could program LEGO-built robots to perform tasks. This version of Logo was later commercialized by LEGO under their Mindstorms brand, which has had multiple iterations over the last 15 years. Lego was not the only group to profit off of Logo, though; multiple companies were formed to sell Logo software and turtle hardware [1].

Even after more than 50 years, usage of Logo has persisted into the 2000s and 2010s. The popular Scratch programming language, also a learners' language, drew a large part of its functionality from a branch of Logo [1].

Professor Brian Harvey created UCBLogo at UC Berkeley, which is still being maintained and updated as of December 2019 [2]. While Python has supplanted it as the novice programmer's language in the public mind, Logo remains a relevant tool for modern young coders.

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**. These simple looping functions are essential for some shapes in turtle graphics; for example, circles are drawn as such:

```
repeat 360 [fd 1 rt 1]
```

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 implicitly storing and reusing the result. An example of these commands follows:

```
test [5=5]
iftrue [print [5 is equal to itself. What a surprise!]]
iffalse [print [5 is not equal to itself. Something has gone very wrong.]]
```

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. [3]

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." [3]. 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.

The data type Word works the same way as variables in other languages, like Java and C. Shown below is an example of how WORDS works like a variable.

```
MAKE "x 5
```

In the above code example the "x is the Logo Word that will hold the value of 5 in this case. The code above uses the Logo command MAKE to set x, the variable name, to the value 5. Logo is a weakly typed language and does not require a type to be specified before setting the value of a WORD. WORDS can also change type in the middle of a procedure. Logo WORD is very handy while iterating or storing desired values.

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 are 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 of all the elements except for the first element. These commands, known as data selectors, are what give Logo LISTS an 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" [3]. 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.)" [3]. 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* [4, 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 [3, 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

While Logo is known for its unique method of generating graphics, those graphics would not be possible if not for the powerful-yet-lightweight features the language boasts for more general programming. The various types of control structures assist in dictating control flow. The data types available for use enable functional programming with templates, as well as lists of instructions for the turtle. The subprogramming features give Logo the benefit of modularity and reusability of code. Without any of these features, Logo would be a much worse language to use. Yet with this robust set of features, Logo remains a strong contender among the learning languages, even 50 years after its creation.

References

[1] Logo Foundation. Logo history.

[2] Brian Harvey. Berkeley logo.

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

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

[5] MIT Media Lab. Professor emeritus seymour papert, pioneer of constructionist learning, dies at 88.