Kam Lai, Jiao Li, Xinyuan Wang, Bohong Zhao, Yi-Yin Chang

Language Tutorial for Educators

The aim of this tutorial is to introduce the core commands and functions to the educators who use Logo++ as an education tool. As a language designed for K-12 students, Logo++ can be used with minimal knowledge in programming. However, we want the educators to take advantage of Logo++ as much as possible. Therefore, this tutorial is not only a quick start, but also an initiative of lesson planning.

In this tutorial, input stream would be preceded with a > character for identifying input. `The font of the stream on the computer screen would be the same as this sentence.`

For a detailed language reference and grammar description, please refer to the language reference manual.

1. Getting started

Logo++ is an interpreted language. It means that the input stream is read and analyzed line by line, and no compilation is required to produce output. In other words, Logo++ is interactive. Users only need to open the program and enter the commands to produce output. Java Runtime Environment (JRE) is required to run Logo++.

1.1. Hello, world!

Although Logo++ is a GUI based language, we follow the convention and introduce the first program as “Hello world” program:

```
>Print “Hello, world!”
```

In this example, the input is a command `Print` with an argument “Hello, world”, which is a string constant denoted by a pair of quotes. The console will print:

```
Hello, world!
```

on the next line.

1.2. Into the Logo++ world

Most part of Logo++ is to navigate the turtle drawing different geometric figures along the trail of the turtle. The basic commands for navigation are `Forward`, `Back`, `Left`, and `Right`. `Forward` and `Back` each takes a numeric argument describing the distance that the turtle will travel forward and backward. `Left` and `Right` each takes an argument describing the angle (in degree, 1/360 of a circle) the turtle turns left and right.

1.2.1. First step

Now we are ready for our first step in the Logo++ world:

```
>Forward 100
```

It means that the turtle will walk 100 steps forward:

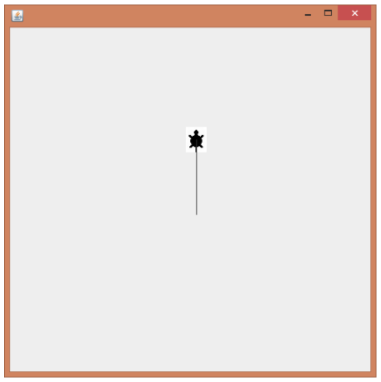

Figure 1 Your first step in Logo++.

Note that, in fact, 1 step means 1 pixel on the screen. The argument following the 4 commands mentioned above could be float points, but may not be reflected on the screen if the step size is quite small.

1.2.2. My first figure

We have just walked our first step. It would be very intuitive to complete a square:

```
>FD 100
>RT 90
>FD 100
>RT 90
>FD 100
>RT 90
>FD 100
>RT 90
```

Here, FD is the shorthand of and has the same meaning as `Forward`. The shorthand of `Back`, `Left`, `Right` are BK, LT, and RT respectively. We turn 90 degrees to the right every time we walk 100 steps forward. The resulting figure would be a square with side length 100:

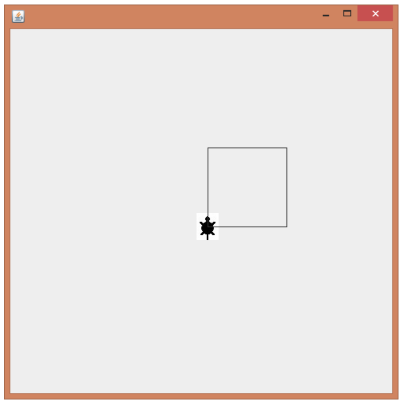

Figure 2 A square with side length 100.

Congratulations! You have just finished you first figure. With these four basic commands, you already can draw many different figures in the Logo++ world.

1.3. Not only a canvas, but also a calculator

Logo++ also serves as a calculator with basic functions. Entering arithmetic expressions will produce results:

```
>1+2*3^2
```

will yield an result

```
19
```

Operators +, -, *, /, and ^ (power operator) are supported in the current version.

2. Variables

2.1. Input

Variables needed to be initialized on the first time of use. Any unreserved identifier can be the name of a variable. Use the `Set` function for initialization as the following:

```
>Set i 2
```

It assigns value 2 to variable named `i`.

There are 2 types of variables in LOGO++: numerical and string. Numerical indeed includes integers and float points, but both have double type accuracy and you don't need to distinguish them. You don't need to and shouldn't declare types of variables explicitly during initialization. LOGO++ could automatically recognize (and even convert) different types once needed.

Variable can be modified through any arithmetic operation.

```
>Set i 200
>Forward i
>Left 90
>Set i i/2
>Forward i               ## notice that i was assigned to i/2 = 200/2 = 100
```

will yield an output:

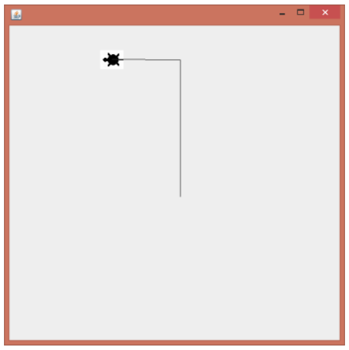

Figure 3 An inverted 'L'.

The turtle has walked forward for 200 steps and left for 100 steps.

The double hash characters (##) signal a comment such that any characters after this sign on the same line will be ignored by the interpreter.

2.2. Output

Once need to know the value of a variable, we could simply output it as below:

```
>i
```

The console will print the value of i as follows:

```
2
```

Now we have two ways to output variables (as well as constant). One is using the command `print`, which is demonstrated in Hello World example, to output all kinds of variables and expressions. The other way is directly typing the name of variables or expressions and press enter. This could be used for numerical variables and expressions. That's how Logo++ could serve as a calculator.

3. Iteration statements

It is painful when we need to enter the repeated commands for multiple times. Fortunately, iteration statements simplify this a lot. With Repeat, for, and while keywords in Logo++, it is convenient and intuitive to loop commands.

3.1. The `Repeat` Statement

Repeat is a reserved keyword in Logo++ for loops. Let's try to use loop to create the square:

```
>Repeat 4 [Forward 100 Right 90]
```

This produces the same square as in Figure 1. This one-line statement has the same meaning as repeating the commands [`Forward 100 Right 90`] for four times, where the bracket is the scope of the `Repeat` statement.

We can combine Repeat statement with variables to create many interesting figures, like this one:

```
>Set i 0
>Repeat 10 [Set i i+25 FD i RT 90]
```

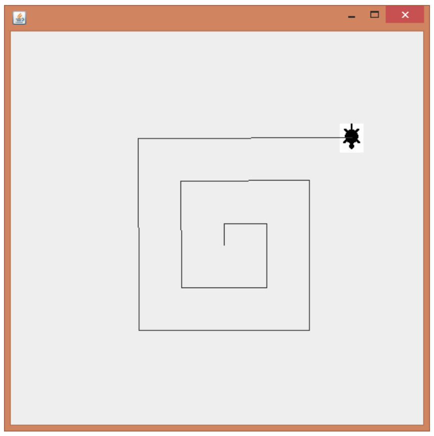

Figure 4 A square maelstrom.

3.2. The `for` statement

`for` statement is similar to repeat statement, except that it always loops with a variable.

```
>for i = 1:4 [FD 100 RT 90]
```

generates again the same square as in Figure 1. In this example, `i` is the index of the iteration; `1:4` is read as “1 to 4”, in which the index is counting from 1 to 4, incrementing by 1 after each iteration. Let’s consider another example:

```
>for i = 0:25:225 [FD i+25 RT 90]
```

This iteration generates the same pattern as in figure 3. In this example, there are three numbers in the `for` expression: `i = 0:25:225`, read as “i from 0 to 225 by 25.” It means to initialize `i` to 0, increment by 25 after each iteration, and repeat until `i` reaches 225. The increment value is optional; it is default to be 1 if not specified. Notice that the value of `i` can be accessed and modified within the loop statement.

3.3. The `while` statement

While statement is another iteration statement. The equivalent statements that generate figure 1 and 3 are:

```
>Set i 1
>while (i <= 4) [FD 100 RT 90 Set i i+1]
```

and

```
>Set i 0
>while (i <= 225) [FD i+25 RT 90 Set i i+25]
```

The statements within the bracket repeat until the expression inside the parenthesis becomes false after certain iteration. With that said, it is very easy to create an “infinite loop”, a loop that never stops, if users have not pay enough attention defining the iteration.

4. Non-navigation commands

Aside from the commands that navigate the turtle, there are several non-navigation commands that control the environment of the canvas. These commands usually take no argument.

`Clearscreen`, or CS in short, erases all drawings on the canvas and returns the turtle to home, which is the center of the canvas.

`Wrap/Fence` toggles on/off the fence on the border of the canvas. In wrap mode, the turtle can travels through the border to the opposite side, regarding the canvas like the map of the earth. In fence mode, the turtle is blocked on the boundary. It means once the turtle reaches the boundary, it will stop immediately. The default mode of the interpreter is wrap mode.

PenDown/PenUp enables/disables the turtle to actually draw on the canvas. In PenDown mode, once the turtle moves from one point to another, it will draw a line between these two points. In PenUp mode, the turtle only moves without leaving trace on the canvas. See this example:

```
>Repeat 20[PenDown FD10 PenUp FD10]
```

draws a dotted line:

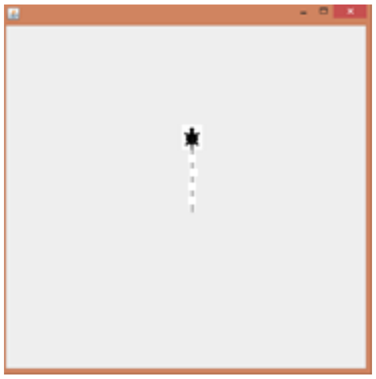

Figure 5 A dotted line.

5. Coordinates

When we are drawing complex figures, the knowledge of the actual coordinates is essential for accuracy. Also, some movement of the turtle should be fine-tuned. Therefore, Logo++ has several useful commands dealing with these issues.

5.1. Getting the current position

GPS is a command to get current position, which takes in no argument, and output the pair of X and Y coordinates, as follows:

```
> GPS
X: 100, Y:-30
```

The center of the canvas is set to be the origin. The axes follow the conventional rectangular axes, where the positive X direction is on the right and the positive Y direction is upward. The unit step is 1 pixel.

5.2. Changing the position of the turtle

`Teleport [x,y]` is a command to directly set the position of the turtle, which takes in a pair of bracketed arguments x and y, which are the coordinates of X and Y respectively. This command teleports, or sets the turtle to the desired position. If you want to move the turtle along only the X or Y direction, i.e. only to change X/Y coordinate without changing Y/X coordinate, you could omit the corresponding argument. So the following 3 commands are all valid:

```
> Teleport [100,-30]    ## move to (100,-30)
> Teleport [, -30]      ## move to (currentX,-30)
> Teleport [100,]       ## move to (100,currentY)
```

Note that, the teleportation could be bounded by the boundary if fence mode is on. In warp mode, Logo++ regards this command as starting from home, and moving along X and Y direction by x and y steps respectively. So the final position is still in the canvas, but the turtle may have wrapped the canvas several times.

Home is a command that sends the turtle back to the origin, the center of the canvas, and also turns the turtle to head top.

6. Functions

Function is a special keyword that allows users to define a sequence of statements and reuse it in handy.

```
>Function factorial(i) [
>Set ans 1
>Repeat i[Set ans ans*i]
>ans]
```

This example defines a function of factorial. The defined function can be called by inputting the name of the function and the required argument properly. When an input 5 is given to the function factorial, the function would return the factorial of 5 by multiplying all integers from 1 to 5.

```
>factorial(5)
120
```

Functions would be very useful in terms of explaining new mathematical concepts that are not supported in default. By implementing a function, students can quickly grasp the computational theory behind the notations.

7. Modules

In Logo++, there are several predefined modules that users can add to the interpreter. Modules are packages containing extra functions and commands that are tailored for students of different grades. Not all modules are preloaded because enabling certain commands would increase the complexity of the system that students may fall short on the extra features. Modularizing also allows educators to pick the suitable contents progressively. Educators who are experienced in programming may also define their own module. However, explaining the details in modularization is out of the scope of this tutorial. So we focus on the introduction of the predefined modules.

To add a module to the interpreter, we just need the keyword Module and the name of it:

```
>Module Math
```

Successfully loaded module would print a message on the console:

```
Math module loaded.
```

7.1. Math

The Math module supports advanced mathematical functions such as trigonometry functions, absolute value, random number generator, the constants pi and e, permutations and combinations, and logarithmic functions. High school students would find these extra functions handy when creating their figure.

7.2. Logic

The Logic module supports the concept of true and false. As mentions in section 3, additional loop functions for and while are available with this module. Users can also leverage the logic operators = (equals), != (not equals), && (and), and || (or) for Boolean algebra expressions.

7.3. Compass

The Compass module intends to teach students the fundamental concepts on compass direction. In addition to allowing users to navigate the turtle using the eight compass directions, user can define where North is pointing at.

8. Network communication (Coming soon)

It would be a little lonely to have only one turtle on the canvas. Users can get connected using the Network module. After adding the module, users can become a host on one computers and invite their friend to draw on the same canvas. With this function, students can cooperate to complete a big picture.

9. Challenge (Coming soon)

Challenge is an interesting and unique function provide by Logo++. The idea is that, educator can draw a picture, and challenge student to draw as close as possible. Using the Challenge module allows educators to define and save the challenge for future use. For example:

```
>Challenge [FD 100 RT 90 FD 100]
```

When the challenge starts, there will be an L shaped line in green. Students, starting from the origin, are asked to draw the same figure which will show in red.

9.1. Hint

Some challenges are very trivial but difficult to be exact. For instance, if student does not know the length of the line segment, he will have a hard time guessing and trying. This is not the intention of this module. So we introduce the `Hint` keyword:

```
        >Hint [Print “Both line segment are of length 100. What's the degree
of a right angle?”]
```

This would be shown at student's console output, and he can use this to finish the challenge efficiently.

At this point we have covered the core functions of Logo++. However, it is just the start. As you start teaching with Logo++, you will find the infinite possibilities of this language from your students!