CHAPTER 1

A First Look at Logo

This chapter introduces the basic mechanics of using Logo. It describes how to evaluate simple commands and how to define and edit procedures. The examples are given in terms of using turtle graphics to draw pictures on the screen. Even though we do not, at this point, introduce more than a few commands or attempt a full explanation of the rules for writing programs, the material in this chapter and the next is sufficient to allow you to use Logo for a wide variety of interesting projects such as the ones described in Chapter 3. Try to work through this chapter at the computer keyboard, experimenting with the different features as they are introduced.

1.1. Getting Started

You start Object Logo like any other Macintosh application by double-clicking on the program icon. When Object Logo starts, it places a window on the screen called the *Listener window*. The Listener window is where you normally type Logo commands and where Logo prints its responses. Object Logo then loads the Startup file and the compiled files in the Startup Folder if they exist.

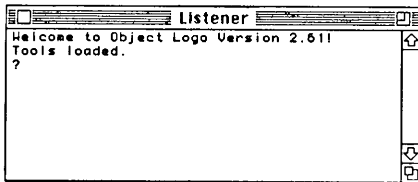

Figure 1.1: The Listener window as it appears when Object Logo is first started

Figure 1.1 shows the Listener window as it appears when Object Logo is first started. Logo prints a welcome message followed by a line beginning with a question mark. The question mark, called a *prompt*, indicates that Logo is waiting for you to give it a command.

1.2. Using Logo Commands

To give Logo a command, you type the command and press the RETURN key. For instance, to tell Logo to print the product of 37 and 67, you type the command line

```
PRINT 37 * 67
```

that is, you type the keys **P**, **R**, **I**, **N**, **T**, space, **3**, **7**, space, *, space, **6**, **7**, RETURN. The computer then prints **2479**, followed by a new line with a question mark prompt, indicating readiness to accept a new command. Bear in mind that when you type a command line, it is not evaluated until you press the RETURN key. To tell Logo to print the message "Logo is a language," you type the command line

```
PRINT [LOGO IS A LANGUAGE]
```

followed by RETURN. This example illustrates how square brackets are used in Logo to group words into *lists*. You can use lists in this way to print messages on the screen, but there are many other uses for lists in Logo, and we will study these in detail in Chapters 5 and 10.

The spaces in these command lines are important, because they indicate to Logo how the line is to be broken into its component parts.[1] If you type the first command line omitting the space between the **T** and the **3** as follows:

PRINT37 * 67

then Logo will think you are telling it to evaluate a command named **PRINT37** and complain that it does not know how to do this, by responding with the error message:

You haven't told me how to PRINT37.

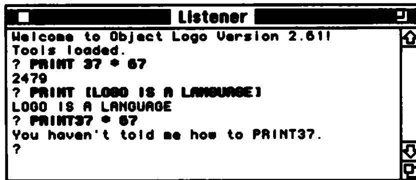

Figure 1.2: Three commands typed to Object Logo and the system's responses

Figure 1.2 shows what appears in the Listener window after you give the three command lines described above, along with the computer's response to each line. The question mark shown at the beginning of each command line is the prompt typed by Logo, and the rest of the line is the command typed by the user. In this book, when we want to emphasize the difference between the characters that you type and the characters that Logo types, we print the characters that you type in boldface. For example, the first command interaction in figure 1.2 would be printed in this book as

? PRINT 37 * 67
2479

In later chapters, we will see how to write Logo programs that manipulate numbers and text. But we begin our study of Logo by investigating how to use the computer to produce drawings by issuing commands to a "object" known as a *turtle*. To set up the screen for drawing, type **CLEARSCREEN** and press RETURN. A new window, called the *Graphics window*, will appear on the screen. As shown in figure 1.3, the Graphics window will be blank, except for a small triangle in the center.

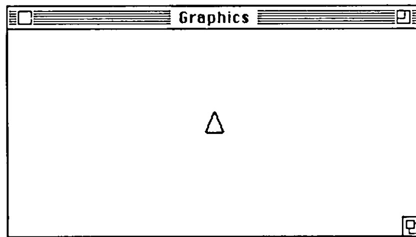

Figure 1.3: Initial appearance of the Graphics window

1.2.1. Basic Turtle Commands

The turtle is the triangular pointer that appears at the center of the Graphics window. You make drawings by telling the turtle to move and to leave a trace of its trail. There are four basic commands for moving the turtle. The commands **FORWARD** and **BACK** make the turtle move along the direction it is pointing. Each time you give a **FORWARD** or **BACK** command, you must also specify a number that tells how far the turtle should move. The commands **RIGHT** and

---

[1] Logo has some knowledge about where it is reasonable to divide lines into component parts, even when they are not separated by spaces. For example, it knows enough to interpret the string of 5 characters **37*67** as containing three elements: the number 37, the symbol *, and the number 67. However, it is a good habit to always use spaces to separate the elements of command lines, even when this is not strictly necessary.

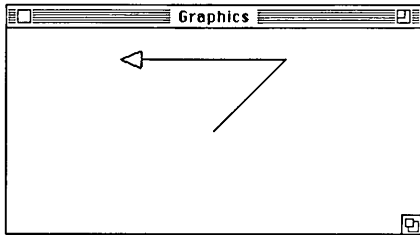

Figure 1.4: Result in the Graphics window of a simple sequence of turtle commands

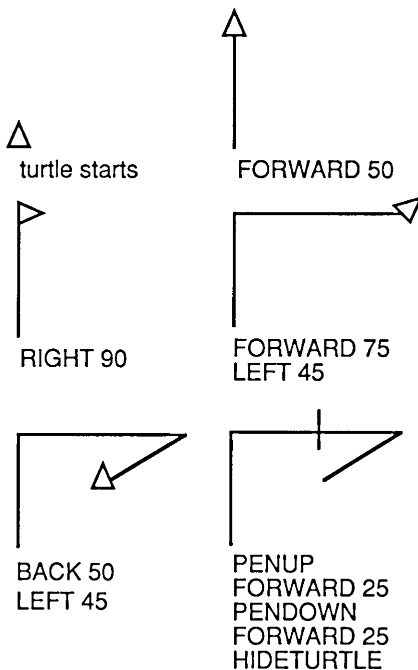

Figure1.5: Drawing with the turtle

LEFT cause the turtle to rotate. **RIGHT** and **LEFT** each require you to specify the amount of rotation in degrees. Try typing the following sequence of Logo commands:

RIGHT 45
FORWARD 100
LEFT 135
FORWARD 200

This should produce the wedge-shaped drawing shown in figure 1.4. Remember to terminate each command line with RETURN and to include a space between the command word and the number. If you mistype a character, you can delete the character by pressing the DELETE key.

The number following the command is called an *input.* **FORWARD, BACK, LEFT,** and **RIGHT** each need one input. Logo commands may or may not require inputs, depending on the command. **CLEARSCREEN** is an example of a command that takes no inputs. Later we will see examples of commands that require more than one input.

If you want to move the turtle without drawing a line, give the **PENUP** command. Subsequent **FORWARD** and **BACK** commands will now make the turtle move without leaving a trail. To resume drawing, give the **PENDOWN** command. Neither **PENUP** nor **PENDOWN** takes an input. The **HIDETURTLE** command causes the turtle pointer to disappear, although the turtle is still "there" and will draw lines if the pen is down. **SHOWTURTLE** makes the pointer reappear. Figure 1.5 illustrates the use of these commands to draw a simple picture.

If you want to start over and draw a new picture, you can use the **CLEARSCREEN** command. This erases the screen and restores the turtle to its initial location at the center of the screen, pointing straight up.

1.2.2. Error Messages

If Logo cannot evaluate the input line, it replies with an error message. Logo's error messages attempt to be helpful in describing what went wrong. For example, if you try to evaluate the command line

PRINT 3 –
Logo will reply

Not enough inputs to –.

because it expects to find something after the – to be subtracted from 3. Another common error message is the result of attempting to use a command that has not been defined. For instance, if you try to evaluate

TURN 100

Logo will respond

You haven't told me how to TURN.

unless you have first defined a procedure named **TURN**.[2] The **YOU HAVEN'T TOLD ME HOW TO** error message often occurs as a result of a typing error. For example, if you type an input line like

FORWARD100

omitting the space between the **D** and the **1**, Logo responds

You haven't told me how to FORWARD100 .

because Logo reads the entire line as a single word, which it assumes is supposed to be the name of a procedure.

When Logo responds to your command with an error message, you should try to determine the reason for the error. Sometimes it is a simple typing error. If so, you can retype the line, or, if you prefer, you can edit the line: Use the up arrow key to move the cursor up to the mistyped line and press the DELETE key. This will recopy the line to the bottom of Listener window where you can edit it. Sometimes however, the reason for a Logo error may be hidden deep in the design of one of your programs. The activity of rooting out and repairing errors in programs is called *debugging*, and Logo provides debugging aids to make this task easier. These are described in section 4.3.

1.2.3. Practice with Commands

If this is your first exposure to Logo, it would be a good idea to review the material covered so far by drawing some figures using the turtle commands. Try to understand any error messages that occur. Following are some things to note in your exploring.

Abbreviations

Some of the commonly used Logo commands have abbreviations to make them easier to type. Abbreviations for some of the commands we have seen so far are

PRINT	**PR**
FORWARD	**FD**
BACK	**BK**
RIGHT	**RT**
LEFT	**LT**
PENUP	**PU**
PENDOWN	**PD**
HIDETURTLE	**HT**
SHOWTURTLE	**ST**

[2] Section 1.3 explains how to define procedures.

Multiple commands on a line

There is no restriction that each line be only a single Logo command. If you like, you can evaluate lines like

```
FORWARD 10 PENUP FORWARD PENDOWN
```

Logo will evaluate the separate commands in order, from left to right. If some command in the line causes an error, Logo will evaluate the commands up until the point of the error before typing an error message. However, single lines that contain many separate commands can be confusing, and it is generally better to use only one command per line.

The REPEAT command

One useful addition to your repertoire of Logo commands is **REPEAT**. **REPEAT** takes two inputs—a number and a list of commands—and repeats the commands in the list the designated number of times. For example,

```
REPEAT 4 [FORWARD 50 RIGHT 90]
```

makes the turtle draw a square. Notice that the list of commands is enclosed in square brackets. Be sure to use square brackets **[ ]**, not parentheses **( )**, for lists. This is a very simple example of how lists are used in Logo to group things. Lists are introduced in section 5.4. **REPEAT**s can be nested. For a pretty effect, try

```
REPEAT 10 [REPEAT 4 [FORWARD 50 RIGHT 90] RIGHT 36]
```

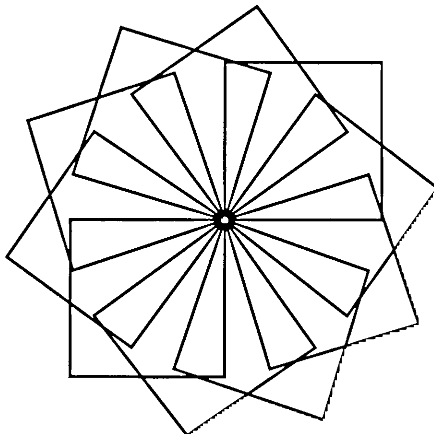

Figure 1.6: Using nested **REPEAT**s to produce a complex drawing

which produces the drawing shown in figure 1.6. Playing with nested **REPEAT**s can be fun, but in terms of program clarity and power, it is much better to combine commands by defining procedures, as we describe in section 1.3.

Stopping evaluation with Command-Period

When Logo is evaluating a command, typing **Command-Period** causes it to stop whatever it is doing and wait for a new command. You type **Command-Period** by holding down the key with an apple and pressing the period key. Logo types

```
Stopped!
```

and prompts for a new input line. For example, if you should start Logo evaluating some long process like

```
REPEAT 10000 [PRINT 1]
```

and then think better of it, you can halt it by pressing **Command-Period.** Another way to stop Logo is to select **Stop** from the **Logo** menu. This will have the same effect as pressing **Command-Period.**

1.3. Introduction to Procedures

You can regard Logo commands like **FORWARD, PRINT, CLEARSCREEN,** and so on, as words that the computer understands when the Logo system is started. These "built-in" words are called *primitives*. One of the most important things about the Logo language is that it makes it easy for you to teach the computer *new* words. Once you define a new word, it becomes part of the computer's working vocabulary and can be used just as if it were a primitive. You teach Logo new words by defining them in terms of words that are already known. These definitions are called *procedures*, and this section describes the simple mechanics of how to define and edit procedures. As in the previous section, the examples are drawn from turtle graphics programs.

1.3.1. Simple Procedures

The following sequence of commands makes the turtle draw a rectangular box as shown in figure 1.7.

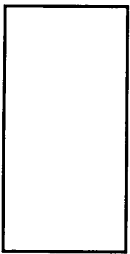

Figure 1.7: Shape drawn by the **BOX** procedure

```
FORWARD 40
RIGHT 90
FORWARD 20
RIGHT 90
FORWARD 40
RIGHT 90
FORWARD 20
```

You can teach the computer to evaluate this sequence of commands whenever you give the command **BOX** by defining **BOX** as a procedure:

```
TO BOX
FORWARD 40
RIGHT 90
FORWARD 20
RIGHT 90
FORWARD 40
RIGHT 90
FORWARD 20
END
```

Notice first that the format of the procedure definition is

• A *title line*, which consists of the word **TO** followed by the name you choose for the procedure.

• A *body*, which is the sequence of command lines that make up the definition.

• The word **END** to indicate that this is the end of the definition.

Once **BOX** is defined, it can now be used in further definitions, such as

```
TO BOXES
BOX
PENUP
FORWARD 5
LEFT 90
FORWARD 15
RIGHT 90
PENDOWN
BOX
END
```

or

```
TO PINWHEEL
REPEAT 4 [BOX]
END
```

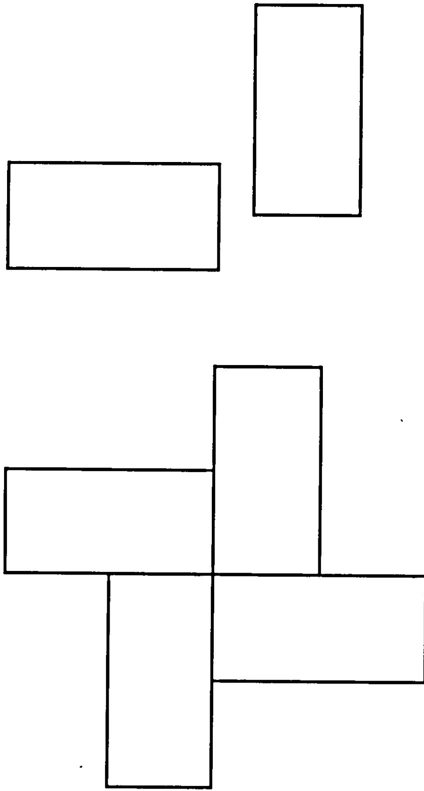

Figure 1.8: Shapes drawn by the **BOXES** and **PINWHEEL** procedures

which produce the drawings shown in figure 1.8. When a procedure is used as part of the definition of a new procedure, it is referred to as a *subprocedure* of the new procedure.

Remember that once a procedure is defined, you can consider it to be just another word that the computer "knows." You tell Logo to evaluate any of these procedures in the same way that you tell it to evaluate a primitive command—by typing the name of the command followed by RETURN.

1.3.2. Defining Procedures

The most convenient way to define procedures is to type the definitions into a separate window which you can later save as a file on a disk. To open the window select **NewFile** under the **File** menu at the top of the screen. A new *file window* will then open; the name of the window will be **Untitled**. Type the definition of **BOX** into the new window separating the lines by return. Don't forget to type **END** at the end of the definition. When you have completed the definition, go under the **Logo** menu and select **Run Selection**. At this point the message

```
BOX defined.
```

will appear in the Listener window, indicating that Logo has now "learned" your procedure definition. You must use **Run Selection** to process each procedure definition or the computer will not "learn" the definition. To try the procedure, select the Listener window and type **BOX** followed by RETURN, just like any other Logo command.

You can add new definitions to the same file window along with **BOX**. For each new procedure, choose **Run Selection.** This will run all the consecutive (non-blank) lines surrounding the current cursor location in the file window.

In general, when you define procedures, you will be moving back and forth between the Listener window and a file window. The definitions are typed in the file window. The Logo command lines that you want to run, together with Logo's responses, are typed in the Listener window. You can use the keyboard shortcuts **Command-1, Command-2** and **Command-3** to activate the topmost Listener, Graphics, and File window, respectively.

Changing Procedure Definitions

Suppose you want to change the definition of a procedure. For example, you may want to change the definition of **PINWHEEL** from

```
TO PINWHEEL
REPEAT 4 [BOX]
END
```

to

```
TO PINWHEEL
FORWARD 50
REPEAT 8 [RIGHT 45 BOX]
BACK 90
END
```

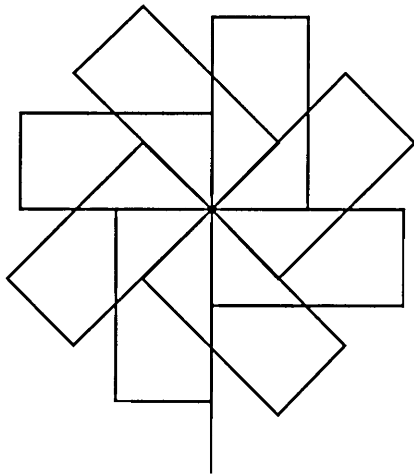

Figure 1.9: Shape drawn by the modified **PINWHEEL** procedure

so that it makes the drawing shown in figure 1.9. To do this, go back to the file window and edit the definition of **PINWHEEL.** When you have completed editing it, select the edited definition with the mouse and do **Run Selection.** Logo will reply

```
PINWHEEL redefined
```

in the Listener window.

Changing a procedure's name (by editing the title line) is equivalent to defining a new procedure with the new title. For example, if you edit the **PINWHEEL** definition to read

```
TO FAN
REPEAT 8 [RIGHT 45 BOX]
END
```

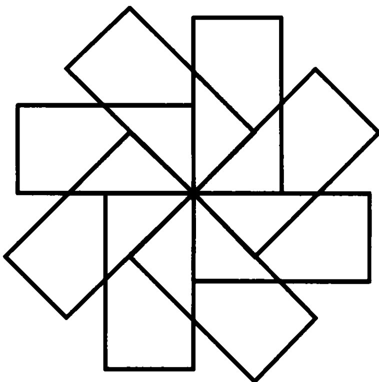

Figure 1.10: Shape drawn by the **FAN** procedure

(which draws the shape shown in figure 1.10), Logo will remember *both* **FAN** and **PINWHEEL.**

Long Lines in Procedures

Occasionally you may want to type a long procedure line, or format a single command line over more than one line so that it looks more readable. Object Logo treats any line beginning with an underscore character (_) as a continuation of the previous line. For example, you can type

```
REPEAT 10
_         [FORWARD 40 RIGHT 65
_          FORWARD 75 RIGHT 90
_          FORWARD 30]
```

instead of

```
REPEAT 10 [FORWARD 40 RIGHT 65 FORWARD 75 RIGHT 90 FORWARD 30]
```

When you reach the end of a screen line and wish to continue the command on the next line, press the tab key. The cursor will move to the next line and Logo will type an underscore and a space and you can continue typing.

1.3.3. Errors in Procedures

If Logo encounters an error while evaluating a procedure, it prints an error message. The error message gives a description of the error and the name of the procedure in which the error occurred. For example, suppose you define the procedure

```
TO BLOCK
ELL
RIGHT 90
ELL
END
```

and the definition of the subprocedure **ELL** contains a typing error (in the third line of the procedure):

```
TO ELL
FORWARD 50
RIGHT 90
FORWATD 25
END
```

Then if you give the command **BLOCK**, Logo will run until it tries to evaluate the third line in **ELL** at which point Logo will respond

```
You haven't told me how to FORWATD, in ELL
```

At this point you should edit **ELL** and correct the mistyped line.

Errors in Procedure Definitions

When Logo processes a procedure definition, it does not look for errors in the lines that make up the body of the definition. For example, if you make a typing error, as in the second line below:

```
TO PINWHEEL
REPEAT 8 [RIGXT 45 BOX]
END
```

the fact that **RIGHT** has been mistyped as **RIGXT** will cause an error when Logo attempts to *evaluate* **PINWHEEL**, not when you define the procedure.[3]

On the other hand, there are certain things that do cause errors when procedures are defined. For example, you may have mistakenly edited the procedure to remove the word **TO** from the title line or caused the definition to be badly formed in some other way. Logo will complain, for example, if you try to define a procedure with the same title as a Logo primitive. For instance, if you attempt to define a procedure named **FORWARD**, Logo will respond with

```
Can't redefine primitive FORWARD.
```

1.3.4. Defining Procedures in the Listener Window

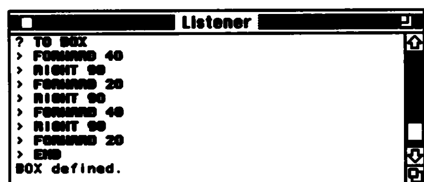

Figure 1.11: Defining a procedure in the Listener window

If you wish, you can type procedure definitions directly in the Listener window, rather than in a separate file window. When you type the title line followed by RETURN, Logo responds with a special prompt character **>**. The prompt **>** in place of the usual **?** indicates that you are defining a procedure—the lines you type are being remembered as part of the procedure definition, rather than being evaluated directly. When you type **END**, Logo responds with a message indicating that the procedure has been defined, and returns to the normal **?** prompt. Figure 1.11 shows an example of a procedure defined in this way.

Editing Procedures in Editor Windows

When you define a procedure in a file window, you can always return to that window to edit the procedure. To edit a procedure defined in the Listener window, you use the **EDIT** command. If you type, for example,

```
EDIT [BOX]
```

---

[3] One very good reason for this is that it is always possible that you *did* mean to type **RIGXT**, and you will be defining a procedure named **RIGXT** before using **PINWHEEL**. One facility that a computer language can provide to encourage sound programming practices is to make it possible to write definitions in terms of procedures that have not yet been defined.

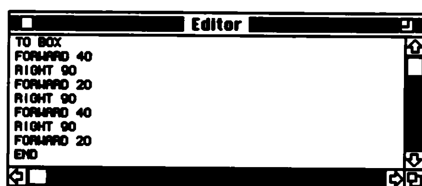

Figure 1.12: An Editor window created by the command **EDIT [BOX]**

Logo will create a new window, called an *Editor window*, and place the text of **BOX** in it so you can edit it as shown in figure 1.12. When you close the Editor window, **BOX** will be redefined according to the edited definition.

For working through this book, we suggest that you type your procedure definitions in a file window, rather than in the Listener window or in an editor window. This way you can organize the file window so that related procedure definitions appear together, which will make your programs easier to understand. You can also use a different file window for each of your Logo projects.

1.4. Other Graphics commands

Object Logo includes a wide variety of graphics commands. This selection describes some of the ones that we have not already seen.

In addition to the turtle commands **FORWARD, BACK, LEFT,** and **RIGHT**, Logo allows you to move the turtle by specifying *x, y* Cartesian coordinates. The **SETPOS** command takes a list of two numbers as input and moves the turtle to the corresponding *x, y* screen location. There are also commands **XCOR** and **YCOR** that output the turtle's position. The **SETHEADING** command rotates the turtle so that it faces in a specified direction, and the **TOWARDS** command outputs the direction in which the turtle should be pointing in order to face a specified point. Giving the command **HOME** moves the turtle back to its initial position in the center of the screen and facing straight up. **CLEAN** erases any drawings on the screen without changing the turtle's position.

1.4.1. Drawing in Color

If you have a color monitor, you can use the **SETPENCOLOR** command to change the color of the lines that the turtle draws. You can also use the **SETBACKCOLOR** command to make the turtle draw on various background colors. The input to these commands is a number that specifies the color. Color numbers are based on the encoding scheme used by Apple QuickDraw, in which the number encodes a particular combination of red, green, and blue. The glossary and the Object Logo Reference Manual provides details on how colors are encoded. Here are the numbers that correspond to various colors:

Black	33	Blue	409
White	30	Cyan	273
Red	205	Magenta	137
Green	341	Yellow	69

For example,

```
SETBACKCOLOR 137
SETPENCOLOR 273
```

will make the background magenta, and the turtle will draw in cyan.

To save you the trouble of remembering the color numbers you can create a file with the following procedure definitions to load in whenever you want to draw in color:

```
TO BLACK
OUTPUT 33
END

TO WHITE
OUTPUT 30
END

TO RED
OUTPUT 205
END
```

and so on.[4]

With these definitions, you can now type, for example

```
SETBACKCOLOR MAGENTA
SETPENCOLOR CYAN
```

1.4.2. Drawing with Patterns

The command **SETPENSIZE** changes the size of the turtle's pen, so that it draws lines of varying height and width.

```
SETPENSIZE 4 8
```

will draw lines that are 4 pixels wide and 8 pixels high. You can also draw lines that are filled in according to a pattern, which is encoded as a list of 8 numbers. For example

```
SETPENPATTERN [238 221 187 119 238 221 187 119]
```

will draw diagonal stripes. Figures 1.13 and 1.14 show examples of different pen settings. See the glossary and the Object Logo Reference Manual for information on how to encode patterns.

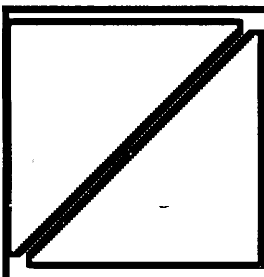

Figure 1.13: Turtle figure drawn with pen size 4 8

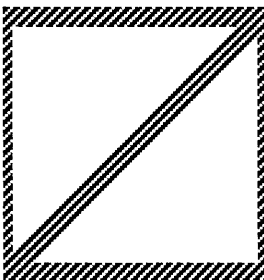

Figure 1.14: Turtle figure drawn with pen size 4 8 and pen pattern 238 221 187 119 238 221 187 119

[4] This is a very simple use of the Logo **OUTPUT** command which is discussed in section 5.2.