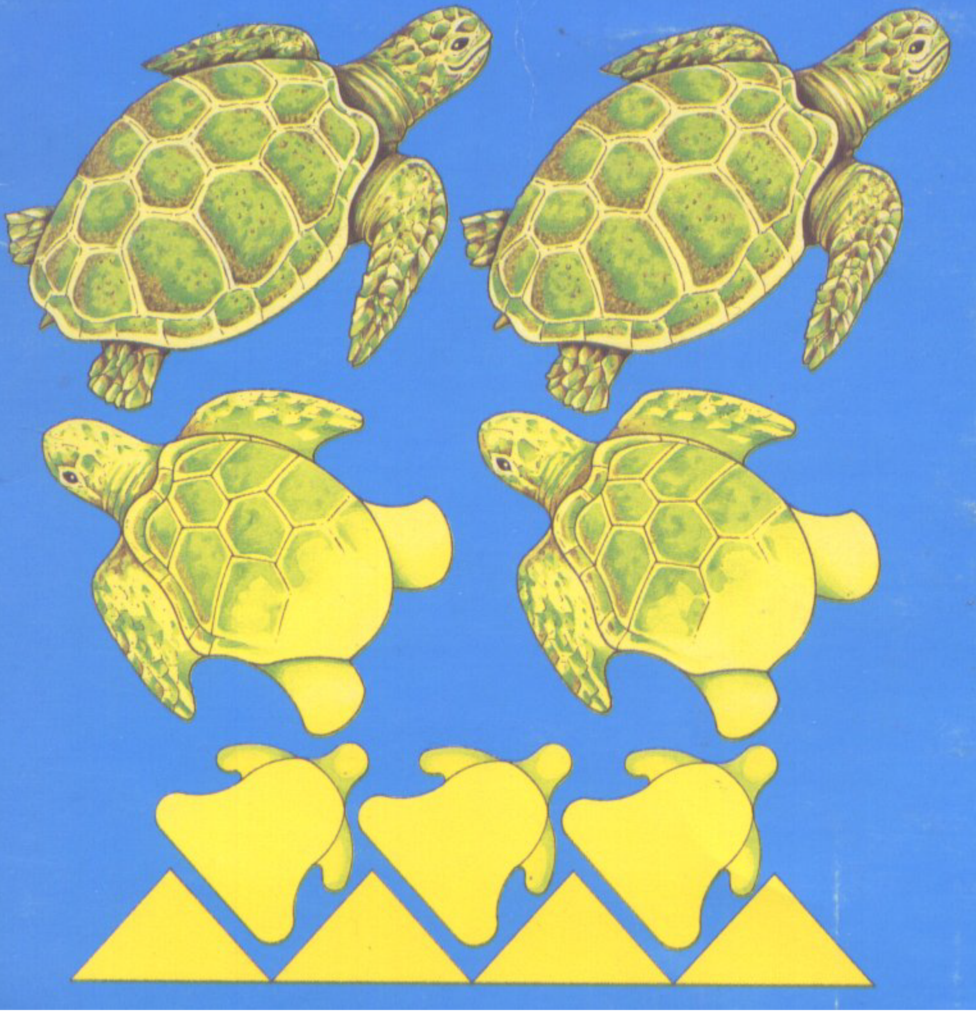
sinclair

ZX Spectrum

Logo 1



Sinclair Logo 1
Turtle Graphics

Sinclair Logo 1
Turtle Graphics

by Ellen Sparer
and the editorial staff of SOLI/LCSI

This edition first published in 1984 by

Sinclair Research Ltd
25 Willis Road Cambridge CB1 2AQ England
ISBN 185016 018 X

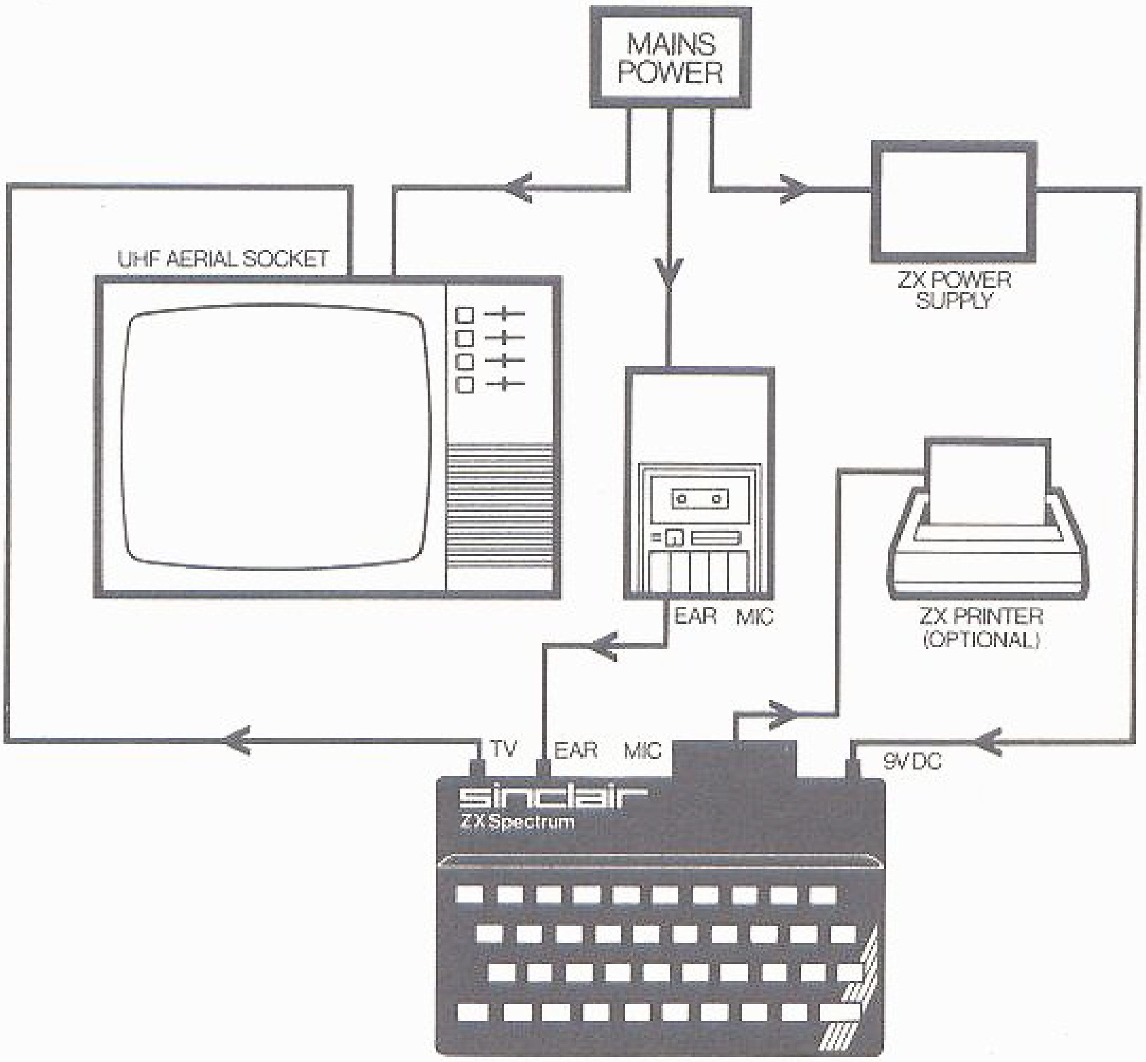
Production supervised by Business Literature Services Ltd;
Typeset by Goodfellow & Egan Ltd;
French's Mill, French's Road, Cambridge CB4 3NP

Printed in England by Staples Printers St Albans Limited at The Priory Press.

Contents

Les Systems d'Ordinateur
Logo International
SO LI I ntl
33 de Poissy
75005 Paris
FRANCE

Logo Computer Systems Inc
LCSI Inc
9960 Cote De Lisse
Lachine Quebec
CANADA H8T 1 A1



How to connect your ZX Spectrum

Chapter 1

Logo on the ZX Spectrum

INTRODUCTION

Welcome to Logo, a computer language which enables you to use your computer to:

Draw
Write
Play games
Calculate.

This manual will teach you to do all these things.

Unlike languages such as English or French, Logo does not have many words or grammatical rules. However, there are a number of words - called primitive procedures (primitives for short) - which Logo understands. These primitives allow you to program your Spectrum in a number of ways. You can write programs which draw, or which manipulate words and lists.

However, you can also extend Logo's vocabulary. You can take the primitive procedures which exist and use them to build new procedures. You can then use your new procedures to build even more complex programs.

This manual concentrates on programs which produce computer graphics, i.e., pictures on the computer screen. Computer graphics allow you to see clearly what you are doing as you are doing it, and are therefore a good introduction to programming.

This manual is not a complete user's guide, but it does enable you to start programming and to edit your programs, and to save and retrieve your work. For more advanced Logo you should refer to Sinclair Logo2 - Programming Reference Manual.

You may run into problems as you work through this book, so we have included some sections called Snags, which suggest how to solve them.

WHAT YOU NEED TO START

To use Sinclair Logo you'll need four things:

1 A 48K Sinclair Spectrum computer and power supply;
2 A television or monitor;
3 A cassette player;
4 A Logo language cassette.

Connect the equipment as shown in the diagram.

Switch on the television set and turn the volume right down. Adjust the tuning until the message:

```
© 1982 Sinclair Research Ltd
```

appears on the screen.

Put the Logo cassette into the tape recorder and set the volume control to Just over half way.

Press the J key on the Spectrum. The word LOAD will appear on the television screen. Hold down the **SYMBOL SHIFT** key and the **P** key twice. The message on the screen will now read LOAD **" ".**

If you have made a mistake typing the loading instructions in you can delete an incorrect letter or word by holding down the **CAPS SHIFT** key and pressing the **0** key.

Press **ENTER** key. The message will disappear, and your Spectrum is Now ready to be loaded with Logo.

Press the \`Play\` on your tape recorder. After a few seconds you will See a pattern of rapidly moving horizontal lines around the edge of the Screen. This means that the program is loading.

If this doesn't happen, rewind the tape and adjust the tape recorder volume a little. Unplug the 9V DC plug from the back of the Spectrum to Clear its memory; then plug it back in. The Sinclair message will reappear, and You can start the loading procedure again. When loading has finished, the following message will appear on your screen:

```
WELCOME TO SINCLAIR LOGO
© LCSI-SOLI 1984
```

This means that Logo is loaded, and you can switch off your tape recorder.

If the \`Welcome\` message does not appear on your screen, rewind the tape and start the loading procedure again making sure that you have carried out each step correctly. The *Spectrum Introduction Manual* gives detailed instructions.

The copyright notice on the screen:

```
LCSI-SOLI 1984
```

means that the program cannot legally be copied.

Logo uses the ? sign as a prompt. The small flashing rectangle is called the *cursor*. It moves along the line as you type. Think of it as the point of a pencil; it tells you where Logo is going to write the next character. The cursor shows that Logo is ready and waiting to receive your instructions, a state known as *top level* in Logo's language

THE KEYBOARD

The keyboard is fully described in the manuals accompanying your ZX Spectrum. However, the following notes will help you with your introduction to Logo.

BREAK/SPACE	The **BREAK/SPACE** key leaves a blank space.
ENTER	The **ENTER** key tells Logo to execute, or carry out, any instructions you have written and returns the cursor to the beginning of the next line. The **ENTER** key must be pressed each time you finish typing a Logo instruction. A Logo instruction begins with the prompt ? and may contain up to 250 characters - thus occupying many screen lines.
SYS	**SYS** is our abbreviation for the SYMBOL SHIFT key.
for example:	Pressing the SYS and P keys at the same time will produce ".
CAPS	**CAPS** is our abbreviation for the **CAPS SHIFT** key. Pressing **CAPS** and the letter keys will produce upper case (capital) letters.
C MODE	Pressing **CAPS** and **2** will lock the keyboard in upper case mode - **C MODE**.
L MODE	Pressing **CAPS** a second time will return you to lower case - **L MODE**.
Delete	If you press **CAPS** and **0** at the same time, Logo will delete one character to the left.
⇦	If you press **CAPS** and **5** at the same time, Logo will move the cursor one character to the left- but will not delete anything.
⇩	If you press **CAPS** and **6** at the same time in **EDIT** mode (Chapter 5), Logo will move the cursor one line down.

⇧	If you press **CAPS** and **7** at the same time in **EDIT** mode (Chapter 5), Logo will move the cursor one line up.
⇨	If you press **CAPS** and **8** at the same time, Logo will move the cursor one character to the right.
E MODE	When you press **CAPS** and **SYS** at the same time, you will see an **E** in the lower left hand corner of your screen. **E MODE** allows you to use the characters printed in red underneath each key. You must hold the **SYS** key down when pressing the third key indicated.

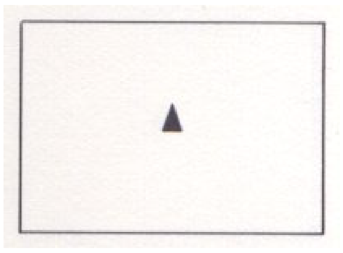
Chapter 2
Top of the Document
Let's draw

INTRODUCTION

The best way to learn Logo is to experiment!

Let's begin programming by learning to draw designs. We will learn to draw by driving a *turtle*, a small animal which 'lives' on the screen. Some Logo turtles are robots which move about the floor on wheels, attached to the computer by a cable. Our graphics turtle appears as a small triangle on the screen. There are many instructions, or *commands*, you can give the turtle. In this chapter we will introduce you to some of the most important. Type:

```
?SHOWTURTLE
```
(press ENTER)

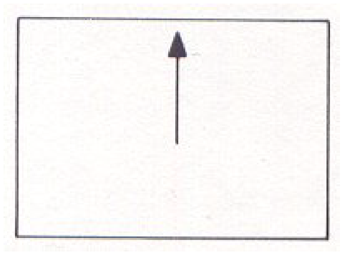


Every time you give a command related to the turtle, your screen will split into two parts. There will be two lines at the bottom where you write your commands; the rest of the screen is the *field* over which the turtle can move.
Notice that the shape of the turtle tells you both its position and its *heading* (in which direction it is pointing). The position and heading are known as the turtle's *state.*

CHANGING THE TURTLE'S STATE

We will now look at some of the instructions or commands for changing the turtle's state. Many of them have abbreviations to make it simpler and quicker to type them in. We will show these abbreviations as we introduce new commands.
We give specific examples, such as FORWARD 50, but you should experiment with different figures as you work though the text so that you become familiar with the effects of the commands. You may type in upper or lowercase characters (large or small letters), although Logo will interpret them in upper case.
Remember that you have to press the ENTER key to make Logo execute your commands.
]]>

```
?FORWARD 50        FD   50
```

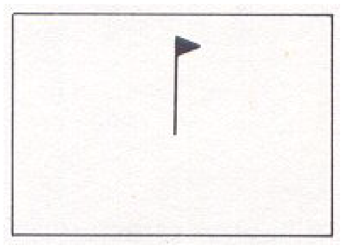


FORWARD is a command, which needs an input: information, which tells Logo how to carry out the command. In these examples, the input is a number. You may, of course, choose almost any input you like for Logo commands - Logo will tell you if the input is not acceptable

The space between the command and the input is very important Because Logo distinguishes between the command FORWARD 50 and the word FORWARD50. On the other hand, if you leave extra spaces between commands and their inputs, Logo will ignore them

Notice that the turtle has changed its position, but not its heading the direction in which it is pointing.

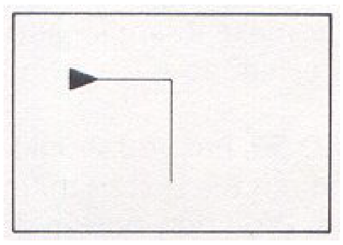
```
?RIGHT 90    TR 90
```



To change the turtle's heading, ask it to turn RIGHT (RT), or LEFT (LT), followed by the number of angular degrees through which you want it to

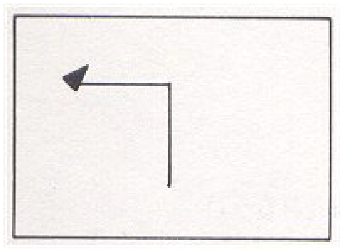
Notice that, in this example, the turtle changes its heading but not its position.

```
?BACK 50      BK 50
```

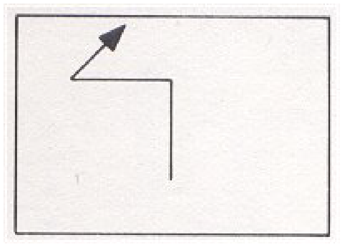


BACK, like FORWARD, tells the turtle to change its position but not its heading. BACK tells the turtle to back away from its current position.

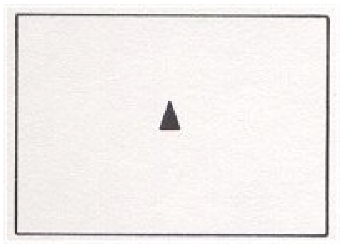
```
?LEFT 45      LT   45
```



The turtle turns 45° left of where it had been heading; it does not change its Position. You can perhaps see what has happened more clearly if you now ask the turtle to move FORWARD 25.



```
?CLEARSCREEN        CS
```



If you wish to clear the screen and start again, give the command CLEARSCREEN. It erases all the lines the turtle has drawn and returns the turtle to its original central position, facing towards the top of the screen.

SNAGS

You may run into snags when using your Sinclair Logo. The turtle may not do what you expect. Often, this will be caused by typing errors. In computer jargon, an error is known as a bug.

The most common bug for beginners is forgetting the space between the command and the input. For example, FORWARD 50 is a Logo instruction. FORWARD50 is a word you might define yourself but probably haven't at this point.

The difference between the two instructions is merely a space between words. The difference between FRWARD and FORWARD is merely 0, but for Logo, it is the difference between its being able to execute an action, and sending you a message.

If you type:

```
?FORWARD50
```

Logo will return a message:

```
I don't know how to FORWARD50
```

Such messages are Logo's way of telling you that it has run into a snag, but it will try to tell you what the snag is. If the message is more than one screen line long, Logo will stop printing at the end of the line and a flashing arrow will appear. Press **ENTER** to see the next line, and continue until you have read the whole message.

Chapter 3

A first procedure

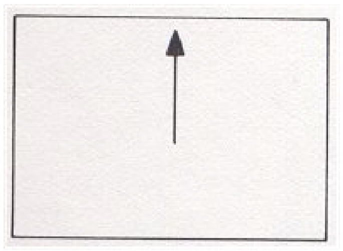
TEACH THE TURTLE TO DRAW A SQUARE

There are certain words Logo automatically understands. These words, such as **FORWARD, RIGHT** etc, are the primitive procedures. From the moment your Logo is loaded, it will understand **FD 50**, but the word **SQUARE**, for example, will mean nothing to it.

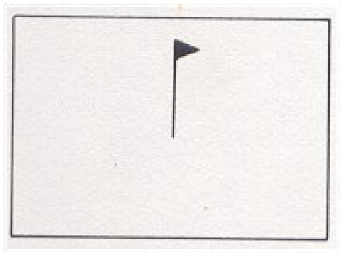
However, Logo can be taught to understand new procedures. For example, you can give **SQUARE** a meaning by combining instructions so that Logo knows how to **SQUARE**. You may call a procedure by any name, provided that it is not the name of a primitive procedure.

Using the commands **FORWARD** and **RIGHT**, we can make the turtle draw a square.

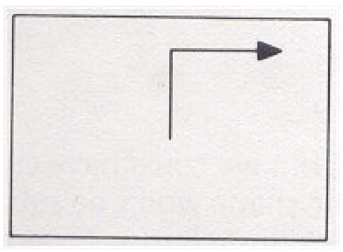
```
?FD 30
```



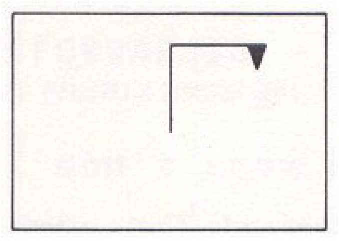
```
?RT 90
```



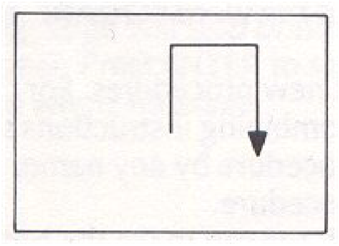
```
?FD 30
```



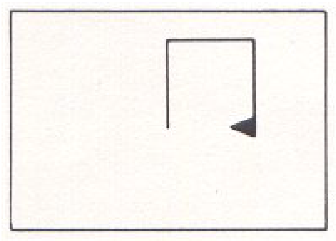
```
?RT 90
```



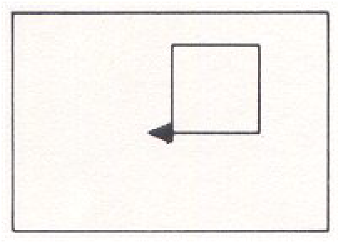
```
?FD 30
```



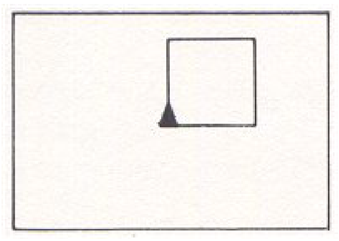
```
?RT 90
```



```
?FD 30
```



```
?RT 90
```



We chose 30, as an input to FD for our example, but we could have chosen any other number. The angle has to be 90, or the shape won't be square!

Before you define a new procedure, you should first choose its name. We might as well call the procedure for drawing a square SQUARE. You use the instruction TO to signal to Logo that you will be defining a procedure; then you write the name of the procedure on the same line.

```
?TO SQUARE
```

Then tell Logo what you want the procedure to do.

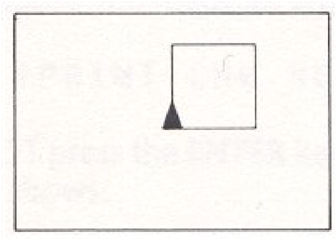
```
>FD 30 RT 90
>FD 30 RT 90
>FD 30 RT 90
>FD 30 RT 90
>END
```

Logo uses the > instead of the **?** as a prompt while you are defining a procedure. This is to remind you that Logo is not executing your commands, but remembering them. The word **END** - typed on a line of its own - signals to Logo that you have finished defining the procedure. Logo will now return:

```
SQUARE defined
?
```

SQUARE defined means that Logo now knows how to **SQUARE**, i.e., to carry out all the instructions contained under the name **SQUARE**.
The prompt **?** shows that Logo is ready to accept new instructions.
Now let's ask Logo to **SQUARE**.

```
?SQUARE
```



SNAGS

Suppose **SQUARE** does not work. Perhaps you have made a typing error. Soon, you will learn to *edit* your procedures so that you can change parts which do not work, or which you do not like.
In the meantime, you can erase your procedure by typing:

```
?ERASE "SQUARE
```

If you now ask Logo to:

```
?SQUARE
```

a Logo message appears:

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

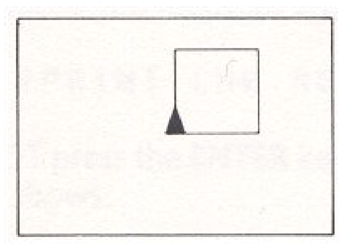
Now rewrite your SQUARE procedure.

Chapter 4

TEXTSCREEN, PRINT and REPEAT

TEXTSCREEN

If you have been drawing with your Logo turtle, the command **TEXTSCREEN** (**TS**) will make 22 lines of the screen available for text. The cursor is at the top left hand corner.



Try typing the following:

```
?PRINT [HOW ARE YOU?]       (press SYS Y for the [ )
 HOW ARE YOU?               (press SYS U for the ] )
 ?
```

Don't forget to leave a space between **PRINT** and the input.
Suppose you made an error and typed:

```
?PRINT [HW RE YOU?]
```

DO NOT press the **ENTER** key; press the **DELETE** (**CAPS 0**) key until your screen shows:

```
?PRINT [H
```

and then retype the rest of the line.

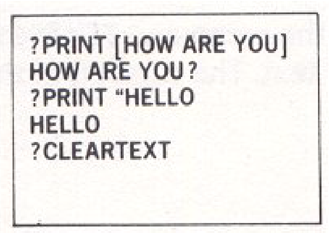
```
?PRINT CHOW ARE YOU?]
HOW ARE YOU?
```

The **DELETE** key (**CAPS 0**) is one of many editing facilities that Logo offers you, enabling you to change what you have typed without rewriting the entire instruction. You will meet more editing keys in the next chapter.
You can experiment with Logo printing by typing **PRINT**, and enclosing the sentence to be printed in square brackets **[ ] .**
You can also ask Logo to print single words by using **"**.

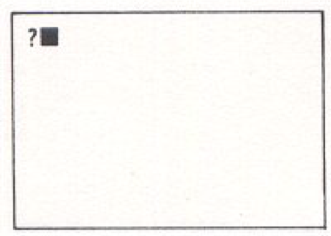
```
?PRINT "HELLO          (press SYS P for the " sign)
HELLO
```

CLEARING THE TEXTSCREEN

The command **CLEARTEXT (CT)** tells Logo to clear the screen of text, and puts the cursor at the top of the screen.



```
?PRINT [HOW ARE YOU]
HOW ARE YOU?
?PRINT "HELLO
HELLO
?CLEARTEXT
```

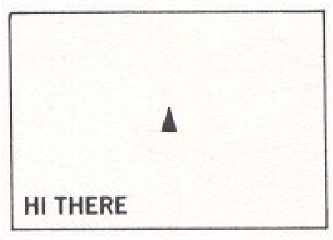


```
?■
```

If you are in graphics mode, the command **CT** will erase what is written on the two lines used for text. You remain in graphics mode with the cursor on the first of the two lines.

Let's write a procedure using **PRINT**.

```
?TO GREET
>SHOWTURTLE
>PRINT [HI THERE]
>END
GREET defined
?GREET
```



```
HI THERE
```

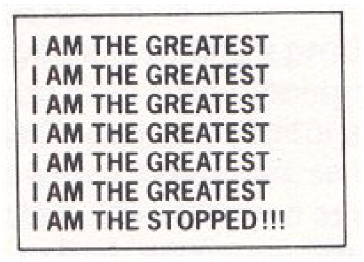
The **REPEAT** command

You can use the command repeat to tell Logo to REPEAT an instruction, for example:

```
?REPEAT 4 [GREET]
```

You can ask Logo to repeat something many, many times, and then stop it in the middle.
For example:

```
?TH
?REPEAT  100  [PR  [I AM THE GREATEST]]
```



```
I AM THE GREATEST
I AM THE GREATEST
I AM THE GREATEST
I AM THE GREATEST
I AM THE GREATEST
I AM THE GREATEST
I AM THE STOPPED!!!
```

If you press the CAPS and BREAK/SPACE keys simultaneously before Logo has finished executing the procedure, it will send you a message:

```
STOPPED !!!
?
```

SPECIAL KEYS

SYS P	"
CAPS BREAK/SPACE	STOPPED!!!
CAPS 0	deletes
SYS Y	[
SYS U	]

Chapter 5

The Sinclair Logo Editor

INTRODUCTION

While you are writing a procedure, you may wish to modify a previous line; after you have tried running it, you may want to change or rewrite it.

The Sinclair Logo Editor allows you to move the cursor anywhere on the screen within a program, so that you can easily erase, move or insert characters. This is known as a *fullscreen* editor.

Note that, when you use the editor you will lose whatever is on your screen; your turtle graphics, or text, will be replaced by the editing screen.

EDIT

EDIT (ED) followed by the name of a procedure tells Logo that you want to edit that procedure. You must put a " (quote mark) before the name; do not leave a space between the " and the name.

NOTE: the name of a procedure may not include spaces, but may include numbers. You may choose any name you like as long as it is not the name of a primitive, or a name already given to another procedure.

Is your procedure **SQUARE** still in the computer? If so, type

```
?EDIT "SQUARE
TO SQUARE
FD 30 RT 90
FD 30 RT 90
FD 30 RT 90
FD 30 RT 90
END
```

When a procedure is already defined, Logo reprints the entire definition.

The cursor is at the beginning of the top line.

There is no prompt symbol when in the Editor.

```
LOGO EDITOR (c) SOLI/LCSI
```

appears at the bottom of the screen.

To leave the Editor without having made any modifications to your procedures, type **CAPS BREAK/SPACE.**

You may also use the Logo Editor to write a new procedure. The advantage of this is that you may make any corrections or modifications you like while defining the procedure.

`?EDIT "SQUARE1` Logo enters the Editor and prints

```
TO SQUARE1
```

You may now type the instructions which make up SQUARE1.

```
FD 40 RT 90          FD 40 RT 90
FD 40 RT 90
FD 40 RT 90
END
```

To move back, type	⇦	(**CAPS 5**)
To move forward, type	⇨	(**CAPS 8**)
To move up, type	⇧	(**CAPS 7**)
To move down, type	⇩	(**CAPS 6**)

When the cursor passes over characters, they remain unchanged.
To erase one character left of the cursor, type **DELETE (CAPS 0).**
Note that typing CAPS 0 at the beginning of the line will move the next line of text to the end of the previous line, for example:

```
FD 40
RT 90
```

Becomes

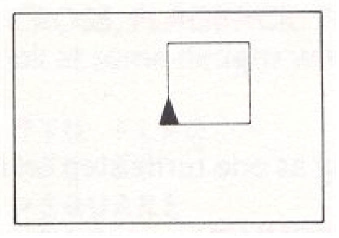
```
FD 40     RT 90
```

Leave the cursor where it is, and press **ENTER** to separate the line again.
When you have finished editing, type **END** and press **E MODE**, followed by **C**; Logo will tell you:

```
SQUARE1 defined
```

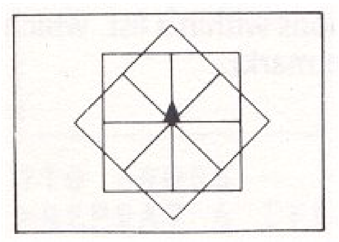
Note that entering **CAPS BREAK/SPACE** causes Logo to forget *all* the work you have done in this session of the Editor.
Try your new command; type:

```
?SQUARE1
```



If you type SQUARE1 again, the turtle will retrace its path. Using the REPEAT procedure we met earlier, we can tell Logo:

```
?REPEAT 8  [SQUARE1 RT 45]
```



Let's make a procedure for this design, and call it SQUARESTAR. We can do this using the EDIT command.

```
?EDIT "SQUARESTAR
TO SQUARESTAR
REPEAT 8 [SQUARE1 RT 45]
END
```

Don't forget to press E MODE C when you finish editing. Let's try our new procedure; put the turtle in its original position at the centre of the screen.

```
?CS
?SQUARESTAR
```

If you don't want to see the turtle, you can type

```
?HIDETURTLE
```

or its abbreviation HT.

SHOWTURTLE (ST) makes the turtle visible again.

SETSCRUNCH

If your squares look like rectangles, the problem may lie in your television or monitor, and not in Logo. The Logo command SETSCRUNCH allows you to change the aspect ratio (the ratio of one vertical turtle step to one horizontal turtle step) on the screen. Try:

```
?SETSCR [50 100]
```

then:

```
?CS SQUARESTAR
```

One turtle step on the y-axis will be twice as long as one turtle step on the x-axis.

Try other settings to vary your designs. SETSCRUNCH parameters are set in multiples of 100.

Your normal Logo screen should be SCRUNCH [100100]. Try different settings until you are satisfied with the results.

Although you may write to Logo in either upper or lower case letters, Logo will, in most cases, transform the lower case to upper case. However, Logo will keep the lower case letters for instructions within a list, which follows a : (colon) and in some cases a " (quote mark).

For example:

```
?to smile
>pr [pr "joke]
>end
SMILE defined

?edit "smile
TO SMILE
PR [pr "joke3
END
```

NOTE

If you type EDIT without an input, Logo will give you the last procedure you edited.

BUILDING ON YOUR PROCEDURES

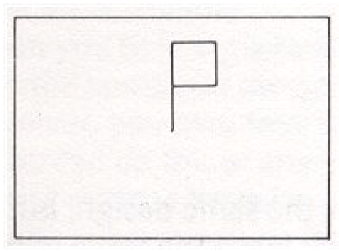
Once you have defined a procedure, it has the same status and behaviour as a primitive procedure. Even if you press ENTER or CS, Logo will retain its knowledge of the procedures you have defined.

Moreover, once you have defined a procedure, you may use it as you would any Logo primitive procedure, such as BK, RT, etc. A procedure you define may therefore be used as part of other procedures; this is one of Logo's powerful features.

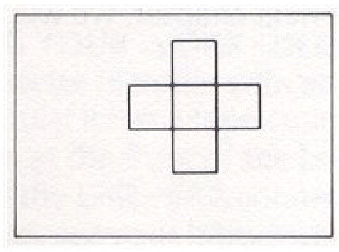
FLAG, CROSS, FLAGBACK, FLAGS, MANYFLAGS

Let's look at some designs which can use SQUARE 1.

```
?TO FLAG
>FD 30
>SQUARE
>END
FLAG defined
?FLAG
```

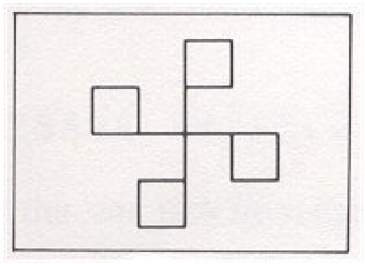


```
?TO CROSS
>REPEAT 4 [FLAG RT 90]
>END
CROSS defined
?CROSS
```

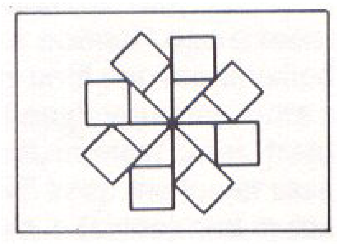


```
?TO FLAGBACK
>FLAG
>BK 30
>END
FLAGBACK defined
```

```
?TO FLAGS
>REPEAT 4 [FLAGBACK RT 903
>END
FLAGS de-fined
?FLAGS
```



```
?TO MANYFLAGS
>FLAGS
>RT 45
>FLAGS
>END
MANYFLAGS defined
?MANYFLAGS
```



Both FLAG and FLAGBACK make the turtle draw the same design but they leave the turtle in different states. Both procedures leave the turtle with the same heading, but FLAG leaves the turtle in a different Position.

FLAGBACK leaves the turtle in the same position on the screen as it Started in. We can see the effects of these differences in CROSS and FLAGS. CROSS runs FLAGS four times, while FLAGS runs FLAGBACK four times.

If you turn your computer off now, you will lose all the procedures you have written - not to mention Logo itself. In the next chapter, we will describe how you can save your procedures.

SPECIAL KEYS

(CAPS 5)	⇦
(CAPS 6)	⇩
(CAPS 7)	⇧
(CAPS 8)	⇨
(CAPS O)	delete
E MODE C	
CAPS BREAK/SPACE	

Chapter 6

Saving and retrieving your work

INTRODUCTION

While you are programming in Sinclair Logo, your Spectrum remembers all the procedures you have taught it. Unfortunately, when you turn the machine off, it 'forgets'.

When you define procedures, Logo puts them in your *workspace* –t he space in the computer memory that lasts only while the computer is on.

However, you may save the procedures you have written on a cassette tape. You can do this at any time during a Logo session using the command **SAVE**, and retrieve them later using the command **LOAD**. If you don't save your work, everything you have done will be destroyed when you turn the machine off.

Information is organised into *files*. You give the file a name and decide how many of your procedures - from one to all of them - you want to put into the file with that name.
Then you can name another file for more procedures, and so on.

SAVING YOUR WORK ON CASSETTE

Any cassette recorder with an input socket for use with a microphone will do. It is useful if it has a tape counter, but this is not essential. Look at the diagram at the front of the book. Before trying to save your work, you should remove the EAR – EAR connection and connect MIC – MIC. If you have any problems, see your Introduction to the Sinclair Spectrum.

When you are using cassette Logo, everything in your workspace at the time can be saved on one cassette. The first time you create a file of your workspace, insert a blank cassette into the tape recorder, rewind the tape to the beginning of the magnetic part (not the leader), and set the counter to 000. Next type:

```
?SAVE "MYFILE "SQUARE1
```

You may give your file any name you like, as long as it has no more than seven characters. The filename must be preceded by a **"** (quote mark), and followed by the name of the procedure to be saved, also preceded by a **"**.

You may save more than one procedure in a file by using brackets. For example:

```
?SAVE "MYFILE [SQUARE1 GREET]
```

Pull out the 'ear' jack from the ZX Spectrum.

Set the tape recorder by simultaneously pressing PLAY, RECORD and PAUSE. Type in the SAVE message and press ENTER. Logo will then tell you to press any key and start the tape; do so. While Logo is recording, the screen flashes. When it finishes flashing, stop the tape; Logo will have saved the procedures you have named.

When saving is complete, the prompt and cursor reappear on the screen, and you can turn the computer off.

The same side of your cassette can be used to save several different files.

It's a good idea to keep a written record describing each file; write down the beginning and end counter number each time you save. Advance the tape recorder approximately 10 counts before saving the contents of another workspace.

RETRIEVING YOUR WORK

Set up your Spectrum, and load Logo. (Don't forget to make sure that the tape recorder is connected for loading, rather than saving.) Prepare your tape, by setting it and the counter of your recorder to the number of the file that you want to retrieve. Then type:

```
?LOAD "MYFILE
```

(or whatever the name of the file is).

Replace the 'ear' jack if it is out; don't forget to press ENTER on your Spectrum, and start the tape.

Logo will print the name of the file it is loading followed by LOG.

While loading, your screen flashes. When the file is loaded, Logo will tell you that the procedures are defined, for example:

```
?LOAD "MYFILE
MYFILE LOG

(screen flashes)

SQUARE1 DEFINED
GREET DEFINED
?
```

Everything you saved in MYFILE will be loaded back into your workspace. The prompt and cursor reappear on the computer screen when the loading is complete.

Careful: If you give the same filename to two or more files, Logo will replace the older one with the more recent. So, for example, if you wish to have more procedures in MYFILE, load "MYFILE, and then save it again, containing your new selection of procedures.

Chapter 7

The turtle's pen and colour

INTRODUCTION

The turtle leaves a trace whenever you give it a graphics command: it has a pen with which it can draw. If you want the turtle to move without leaving a trace, you can ask it to lift its pen. You can also change the colour of the trace by changing the colour of the pen, the colour of the background, and the colour of the border. This chapter tells you how to use the pen and the colour graphics.

PEN COMMANDS

To lift the turtle's pen, type:

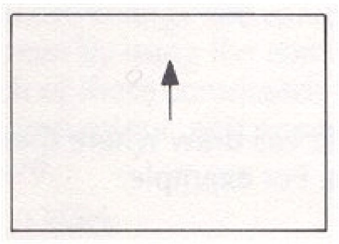
```
?PENUP          (PU)
```

To make the turtle draw again, type:

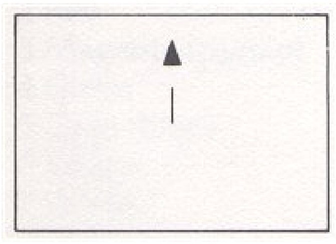
```
?PENDOWN       (PD)
```

Experiment with these two commands.

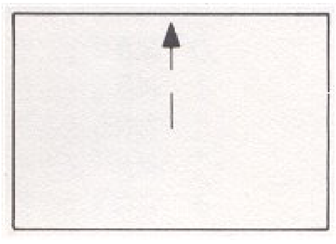
```
?FD 20
```



```
?PU FD 20
```



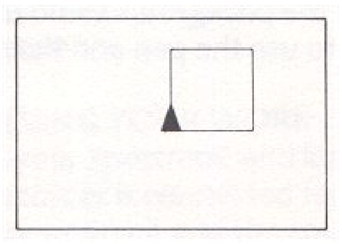
```
?PD FD 20
```



In addition to PEN UP (PU) and PENDOWN (PD), there are two other commands for changing the state of the turtle's pen, PENERASE (PE) and PENREVERSE (PX).

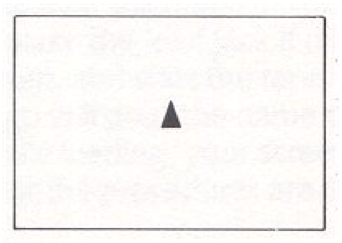
PENERASE turns the turtle into an eraser. When it travels over a line, it erases it. To make it draw again, type PENDOWN or PD. For example (assuming SQUARE1 is loaded):

```
?CS PD
?SQUARE1
```



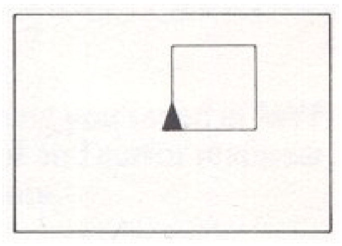
Now type:

```
?PE
?SQUARE1
```

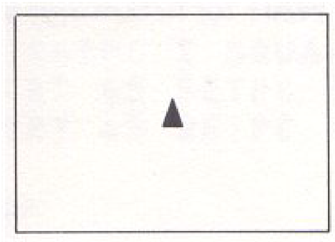


PENREVERSE (PX) is a mixture of PD and PE.
When you give Logo this command, the turtle will draw where there is a blank space and erase where a line already exists. For example:

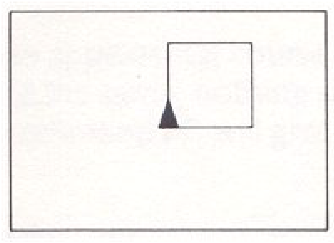
```
?PX
?SQUARE1
```



```
?SQUARE1
```



```
?SQUARE1
```



PENDOWN (PD) will return the pen to its normal drawing state.

USING SINCLAIR LOGO COLOUR GRAPHICS

This section is applicable only if you are using a colour television or monitor; in black and white you will see only shades of grey.

There are three types of colour changes you can make. You can change the colour of the turtle's field or **BACKGROUND** by using the command **SETBG.**

You can change the colour of the turtle's pen or **PENCOLOUR**, by using the command **SETPC**.

You can change the colour of the border of the turtle's field and of the text screen by using the command **SETBORDER** or **SETBR**.

Each of these commands takes one input: a number which corresponds to the desired colour, as it appears above the top row of keys on your Spectrum keyboard.

0	Black
1	Blue
2	Red
3	Magenta (purple)
4	Green
5	Cyan (blue)
6	Yellow
7	White

Changing the colour of the background

```
?SETBG      0
?SETBG      1
?SETBG      2
?SETBG      3
?SETBG      4
```

We can write a procedure which cycles through all the colours. Let's make use of the command WAIT, which tells Logo to **WAIT** for n/60ths of a second before executing the next command.

```
?TO COL.BK
>SETBG 0 WAIT 20
>SETBG 1 WAIT 20
>SETBG 2 WAIT 20
>SETBG 3 WAIT 20
>SETBG 4 WAIT 20
>SETBG 5 WAIT 20
>SETBG 6 WAIT 20
>SETBG 7 WAIT 20
>END
COL.BK    defined
```

Try:

```
?REPEAT 3 [COL.BK]
```

The command **BACKGROUND (BG)** will give you the current colour of the background.

```
?PR BG
?
```

We can modify our program:

```
?TO BACKGR
>SETBG BG + 1 WAIT 20
>END
BACKGR defined

?TO CB
>REPEAT 7 LBACKGR3
>END
CB defined
```

Changing the colour of the pen
Use the command **SETPC;** the code for the pen colours is the same as that for the background.
If you have changed the pen colour, and give the command **TEXTSCREEN,** Logo will write in the pen colour you have set. However, the two lines at the bottom of your graphics screen will always appear in black (or white if you have a dark border) regardless of the pen colour set.
Try typing the following:

```
?SETBG 0
?CS
?SETPC 2 SQUARE1
?RT 45 SETPC 3 SQUARE1
?RT 45 SETPC 4 SQUARE1
```

Now type:

```
RT 45 SETPC 0 SQUARE1
```

No square appears! Of course - since the pencolour and the background colour are the same, nothing shows on your screen.

The command PC will give you the current code number for the pen colour.

```
?PR PC
0
```

Try the following procedure:

```
?TO TOTO
>REPEAT 8 [FD 30 RT 45]
>END
TOTO defined

?SETBG 7
7SETPC 1
?TOTO

?TO TOTAL
>CS REPEAT 18 [TOTO RT 20]
>HT
>END
TOTAL defined
```

Now try changing the background and pen colours:

```
?TO CHANGECOL
>SETPC PC + 1
>SETBG BG + 1
>REPEAT 4 [FD 30 RT 45]
>RT 20
>END
CHANGECOL defined
?REPEAT 8 [CHANGECOL]
```

Changing the colour of the border

The primitive **SETBORDER** or **SETBR** allows you to change the colour of the border of your screen. The colour codes are the same as before. Try:

```
            ?SETBR 1
```

SNAGS

You will sometimes find that changing the background affects the colour of the traces already on the screen. This shows up most often with PENERASE and **PENREVERSE**.

Colours will vary depending on the type of television or monitor and its condition and adjustment.

Chapter 8

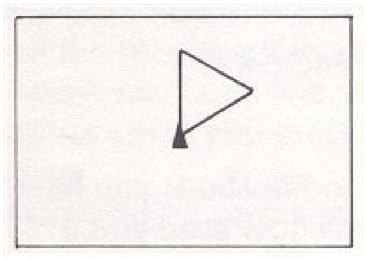
A second look at editing procedures

INTRODUCTION

You may use your Sinclair Logo Editor to change existing procedures as well as to define new ones. This is helpful if you want to correct an error or change what a procedure does.

For example, let's draw a triangle:

```
?TO TRIANGLE
>FD 45 RT 120
>FD 45 RT 120
>FD 45 RT 120
>END
TRIANGLE defined
?TRIANGLE
```



ENTERING THE EDITOR

You may enter the Editor in several ways. Each one has a slightly different result. If you type:

```
ED or EDIT
```

not followed by a procedure name, Logo will bring you the last procedure you wrote, or modified in the Editor.

```
ED "TOTO or EDIT "TOTO
```

or

```
ED [TOTO TOTAL] or  EDIT [TOTO TOTAL]
```

will tell Logo to look for the named procedure(s) and put them on your Editor screen. If the named procedure(s) have not been previously defined, Logo will bring you an empty Editor screen.

```
ED  [] or EDIT []
```

(with empty brackets), tells Logo to give you a blank Editor screen
Suppose we want to turn our triangle. Type:

```
?EDIT "TRIANGLE
```

Your screen will now show the text of the procedure TRIANGLE:

```
TO TRIANGLE
FD 45 RT 120
FD 45 RT 120
FD 45 RT 120
END
```

The cursor is positioned on the letter T of the word TO; to edit you move the cursor where you want to add or delete characters.

First, move the cursor to the end of the title line using the ⇨ key (CAPS 8). Now press the ENTER key; this will insert a line. You can now type:

```
RT 30
```

To move the cursor to the end of the text, type E MODE E.

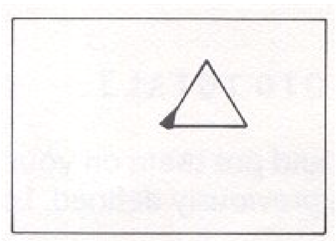
LEAVING THE EDITOR

Typing E MODE C tells Logo to incorporate the modifications you have made, and that you have finished editing. Logo will now send you a message:

```
TRIANGLE defined
```

If you change your mind and decide you do not want Logo to incorporate the modification you have made, type CAPS BREAK/SPACE. Logo will then exit from the Editor, leaving the program exactly as it was before you started editing.

```
?TRIANGLE
```



SUMMARY OF EDITING KEYS

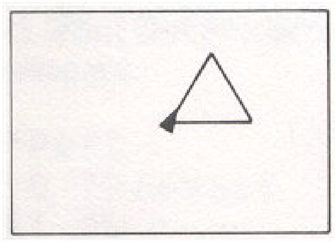
CAPS 5	Moves cursor left one character
CAPS 6	Moves cursor down one line
CAPS 7	Moves cursor up one line
CAPS 8	Moves cursor right one character
CAPS 0	Deletes character to left
E MODE CAPS 5	Moves cursor to beginning of line
E MODE CAPS 6	Moves cursor to end of screen
E MODE CAPS 7	Moves cursor to beginning of screen
E MODE CAPS 8	Moves cursor to end of line
E MODE B	Moves cursor to beginning of text
E MODE E	Moves cursor to end of text
E MODE N	Moves cursor to next page
E MODE P	Moves cursor to previous page
E MODE Y	Erases (yanks) line from screen
E MODE R	Re-inserts line just erased by the E MODE Y command At the beginning of a line, will order Logo to execute the instructions it just carried out

EDITING OUTSIDE THE EDITOR

While all the special editing keys work outside the Editor, many of them only work within one Logo line. A Logo line is a line which starts with the prompt ? and finishes when you press the **ENTER** key, and may contain up to 250 characters.

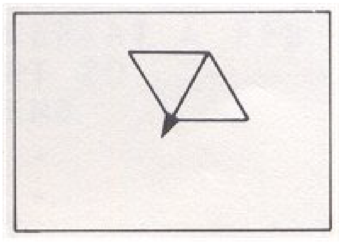
Certain editing commands work all the time. For example, type:

```
?TRIANGLE
```

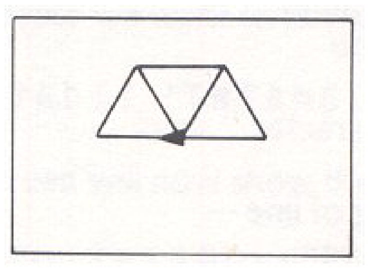


Now type **E MODE R**. This will copy the last line you typed, and **TRIANGLE** will reappear on your screen. The cursor is at the end of the line. Type **E MODE 5** to move cursor to beginning of the line.
Now enter:

```
?LT 90
```



Type **E MODE R** again, and press **ENTER**.



Logo 2, the Programming Reference Manual gives a more detailed Description of the Editor and editing keys.

Chapter 9

Your works pace

INTRODUCTION

Your workspace contains all the procedures you have defined. Logo has certain primitives which help you to organise your procedures in the workspace, and to eliminate those you no longer want.

To find out what is in your workspace, you can ask Logo to print out the titles of the procedures, or their definitions.

PRINTING OUT PROCEDURES

POTS (Print Out TitleS), prints out the title and the title lines of each of the procedures in the workspace.

```
?TS
?POTS
TO TRIANGLE
TO CHANGECOL
TO TOTAL
   -
   -
   -
```

If you type SYS S, Logo will stop its display and wait until you ask it to continue by pressing any key.

POPS (Print Out Procedures) prints the definitions of all the procedures in your workspace.

```
?POPS
TO TRIANGLE
RT 30
FD 45 RT 120
FD 45 RT 120
FD 45 RT 120
END

TO CHANGECOL
SETBG BG + 1
REPEAT 4  [FD 30 RT 45]
RT 20
END
 -
 -
?
```

You can print out the definition of a particular procedure with the command PO (Print Out).

```
?PO "SQUARESTAR
TO SQUARESTAR
REPEAT 8 [SQUARE1 RT 45]
END
```

PO can also be given a list of names; for example:

```
?PO [SQUARE1 SQUARESTAR TRIANGLE]
```

will tell Logo to print out the three procedures whose names are in the input list. Remember, you can use TS or TEXTSCREEN to get a full screen of text, which will make it easier to read.

ERASING FROM THE WORKSPACE

You can erase procedures from your workspace. But, be warned! If you want your procedures and have not saved them, you will have to type them in again, so be sure that you really have finished with them before you erase them.

ERASE (ER) eliminates the definition of the named procedure.

```
?ER "TRIANGLE
```

will erase the procedure TRIANGLE.

```
?ER [TRIANGLE SQUARE1 FLAG]
```

will erase the procedures in the list.

```
?ERPS
```

(ERase Procedures) erases *all* your procedures from the workspace.

SAVING YOUR WORK ON THE PRINTER

In order to get a hard copy (on paper) of your workspace, you must have a ZX Spectrum Printer connected to your Sinclair Spectrum. If you do, typing:

```
?PRINTON
?PO "TRIANGLE
```

will tell Logo to print the procedure TRIANGLE. You may ask Logo to print more than one procedure by typing:

```
?PRINTON
?PO [TRIANGLE SQUARE]
```

To stop the Printer, type:

```
?PRINTOFF
```

If you wish to save a graphics screen, ie a design or drawing you have made, use the command:

```
COPYSCREEN
?TOTAL
?COPYSCREEN
```

Logo will copy everything on your screen except the two lines of commands at the bottom.

Chapter 10

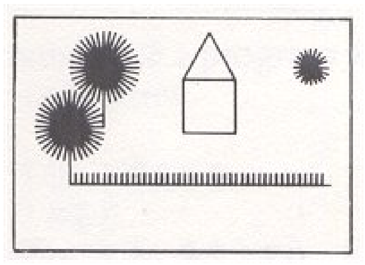
A first project: Drawing a garden

INTRODUCTION

Our first project will be to draw a GARDEN which contains a HOUSE, a SUN, two TREES and a LAWN.

Logo's capacity for defining new procedures allows us to divide the project into smaller parts.

Let's start by drawing a picture of our garden on a sheet of paper.



We can see that the GARDEN is made up of:

1 HOUSE
2 two TREES
3 LAWN
4 SUN

We can write separate procedures for each. Then we will think about putting them together.

1 HOUSE

As a first step, let's break the HOUSE down into its subparts. We see it is made up of a SQUARE and a TRIANGLE. Let's write a procedure for each.

We will call the house square SQUARE2

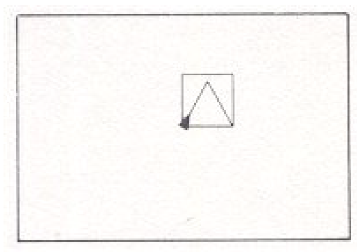
```
?TO SQUARE2
>REPEAT 4 [FD 45 RT 90]
>END
SQUARE2 defined
```

Let's print our previously defined procedure TRIANGLE.

```
?PO "TRIANGLE
TO TRIANGLE
RT 30
FD 45 RT 120
FD 45 RT 120
FD 45 RT 120
END
```

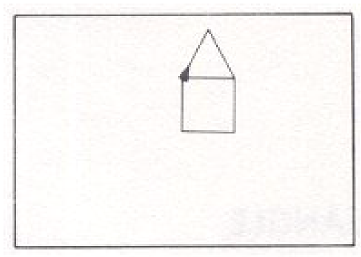
Now we'll put the two together to make a house.

```
?TO HOUSE
>SQUARE2
>TRIANGLE
>END
HOUSE defined
?HOUSE
```



Not quite what we had in mind!

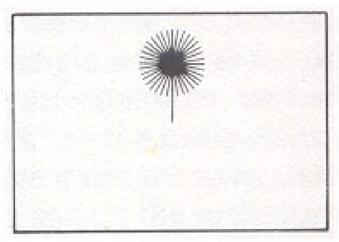
```
?ED "HOUSE
TO HOUSE
SQUARE2
FD 45
TRIANGLE
END
HOUSE defined
?HOUSE
```



That's better!

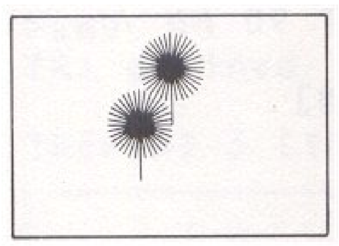
2 two TREES
Let's start by making one TREE.

```
?TO TREE
>FD 50
>REPEAT 36 [FD 30 BK 30 RT 10]
>BK 50
>END
TREE defined
?HT TREE
```



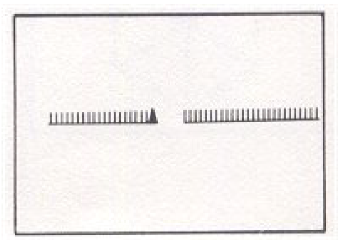
We can now write a program which draws a tree, moves the turtle, and draws a second tree.

```
?TO TREES
>TREE
>PU LT 90 FD 30 LT 90 FD 50 RT 1 !
80 PD
>TREE
>END
TREES defined
?TREES
```



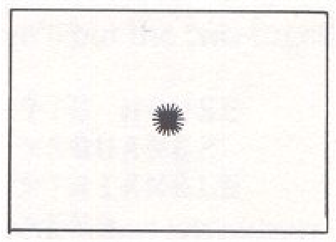
3 Let's now make the LAWN which will border our GARDEN.

```
?TO LAWN
>REPEAT 45 CFD 10 BK 10 RT 90 FD!
5 LT 90]
>END
LAWN defined
?LAWN
```



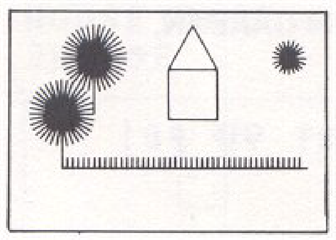
4 Finally, let's make the SUN which rises on the GARDEN!

```
?TO SUN
>HT
>REPEAT 26 [FD 15 BK 15 RT 18]
>END
SUN defined
?SUN
```



Let's think about how to put these procedures together to draw the GARDEN. We have to make sure that at the end of each procedure, the turtle is in the right position for executing the next procedure. Try drawing the garden before you look at the procedure below.

```
?TO GARDEN
>HOUSE
>PU LT 150 FD 80 RT 120 PD
>TREES
>LAUN
>PU FD 100 LT 90 FD 15 RT 90 PD
>SUN
>PR [WELCOME TO MY GARDEN]
>END
GARDEN defined
?GARDEN
```



Now try drawing each part in a different colour!

Chapter 11

Simple turtle geometry

POLYGONS

When we wrote the procedure **HOUSE**, we made an equilateral triangle, that is, a triangle where all the sides are equal and all the angles are equal.

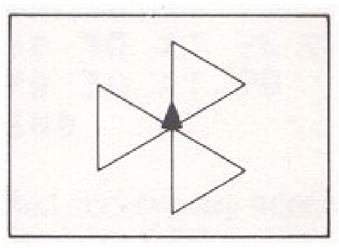
If you remember, we had to turn 120° at each corner. Here is the reason why. When the turtle starts a triangle trip, it must turn 360° - a complete turn - before it returns to its starting state: 3 x 120° = 360°. Remember that to draw a square the turtle turned 4 x 90° = 360°. As long as the sum of the angles is 360°, you will get a closed figure. We call this the turtle's *theorem!*

```
REPEAT 3 [FD 30 RT 120]   →    TRIANGLE
REPEAT 4 [FD 30 RT 90]    →    SQUARE
REPEAT 5 [FD 30 RT 72]    →    PENTAGON
REPEAT 6 [FD 30 RT 60]    →    HEXAGON
```

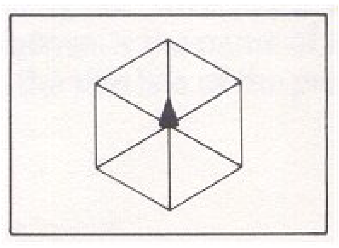
Let's make a new **TRIANGLE** procedure using the instructions we have written above, and then play with it a bit.

```
?TO TRI
>REPEAT 3 [FD 30 RT 120]
>END
TRI defined

?REPEAT 3 [TRI RT 120]
```



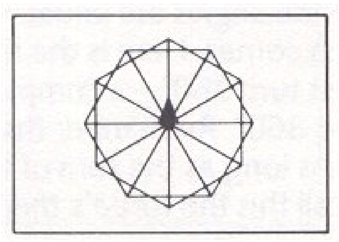
```
?REPEAT 6 [TRI RT 60]
```



If you want to work out how many times the turtle needs to repeat a set of instructions to make a closed figure, divide the number of degrees into 360. For example, if the turtle turns an angle of 30° each time, it has to repeat the instructions 360/30, or 12 times.

However, Logo can do arithmetic so it can therefore do the division for you:

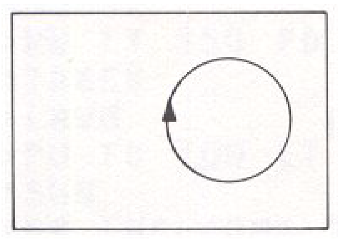
```
?REPEAT 360/30 [TRI RT 30]
```



CIRCLES

Have you noticed yet that the more sides a polygon has, the more it resembles a circle? If you have experimented with polygons by increasing the number of sides, you may have discovered the circle!

```
?REPEAT 360 [FD 1 RT 1]
```



This circle looks fine, but it takes a long time to draw. That's because it repeats the instructions 360 times.

Can you make a plausible circle more quickly?

Chapter 12

Introducing variables

INTRODUCTION

We are now quite familiar with the notion of inputs: the specific information which primitive procedures such as **FORWARD** and **RIGHT** need to give them meaning. The procedures you write can also have inputs. Because you can choose any input within an allowed range, it is sometimes called a *variable* - because it can vary. Let's see how we can use variables.

BIG SQUARES AND SMALL SQUARES

We might want the turtle to draw squares with sides of 50 or 60 or 100 or 10. One way to do this is to write many procedures:
SQUAREFIFTY, SQUARESIXTY etc.

But there is a short cut. We elm change the procedure **SQUARE** so that it takes a variable input. Then we can tell Logo how long to make its side by typing:

```
?SQUARE 50
?SQUARE 60       etc
```

Let's edit our **SQUARE** procedure:

```
?ED "SQUARE
TO SQUARE
FD 30 RT 90
FD 30 RT 90
FD 30 RT 90
FD 30 RT 90
END
```

Since what should vary here is the length of the **SIDE** of each square, we can call our variable **SIDE.**

Variables are always indicated on the title line by writing: (colon) followed by the name of the variable. The colon tells Logo that the word which follows is the name of a variable. So use your editing commands to change the title line of the procedure.

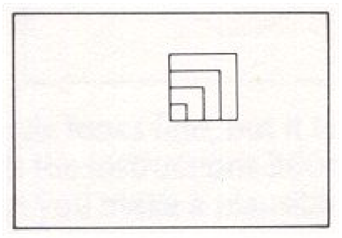
```
TO SQUARE :SIDE
FD :SIDE
RT 90
FD :SIDE
RT 90
FD :SIDE
RT 90
FD :SIDE
RT 90
END
SQUARE defined
```

The : (colon) tells Logo that the word which follows names a container that may have in it a number, another word, a list, or a list of lists.

Here, the expression :SIDE stands for 'whatever happens to be in the container SIDE'. There must be something in the container for Logo to carry out the command FORWARD :SIDE.

You can now give any value you wish to the variable SIDE; indeed, you must indicate the value of SIDE before you can execute the procedure.

```
?CS
?SQUARE 10
?SQUARE 20
?SQUARE 30
?SQUARE 40
```



If you now ask Logo to:

```
?SQUARE
```

you receive a message:

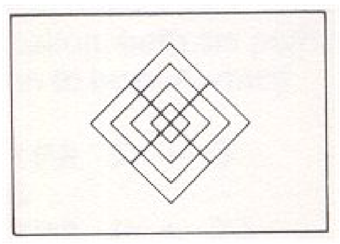
```
Not enough inputs to SQUARE
```

The container is filled when you type SQUARE 10 (or whatever value you like); Logo puts the value you type into the container named SIDE.

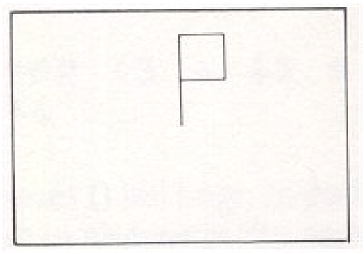
SOME PROCEDURES USING SQUARES

```
?TO SQUARES
>SQUARE 10
>SQUARE 20
>SQUARE 30
>SQUARE 40
>END
SQUARES defined
```

```
?TO DIAMONDS
>RT 45
>REPEAT 4 [SQUARES RT 90]
>HT
>END
DIAMONDS defined
?DIAMONDS
```



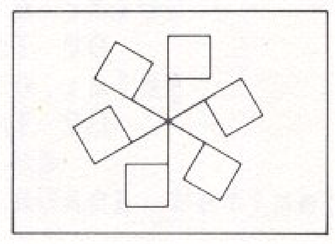
```
?TO FLAGR :SIZE
>FD :SIZE
>SQUARE :SIZE
>BK :SIZE
>END
FLAGR defined
?FLAGR
```



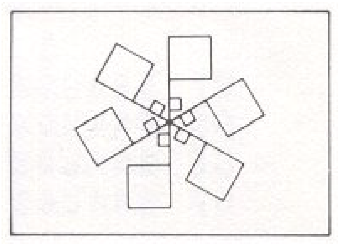
But note, our procedure was called SQUARE :SIDE. It had a different name for its input.

SIDE, which is the variable name for SQUARE, receives its value from :SIZE. A subprocedure may use different names for its inputs from those r given in the original procedure, as long as the total number of inputs does not change.

```
?TO 6FLAG :SIZE
>REPEAT 6 [FLAGR :SIZE RT 60]
>END
6FLAG defined
?6FLAG
```



```
?TO SPINFLAG :SIZE
>6FLAG :SIZE
>6FLAG :SIZE - 20
>END
SPINFLAG defined
?SPINFLAG
```



SNAGS

1 You forgot the space before the : (colon)
2 You typed a space between the : and **SIDE**
3 You typed : in front of a number
4 You typed :**SIE** or something like that
5 You forgot the :
6 You inserted an extra instruction
7 You accidentally erased an instruction

Chapter 13

Numbers and arithmetic

INFIX AND PREFIX

As we have seen in some of the examples, Sinclair Logo can carry out arithmetic operations. To do this, you use the computer symbols for the operations:

/	division
*	multiplication
-	subtraction
+	addition

These signs are written between the numbers, and are known as *infix* operations.

If there is more than one operation, division is performed before multiplication, both are performed before subtraction, and addition is the last operation to be performed.

```
?PR 5 + 3
8
?PR 4 * 23
92
?PR 345 - 32
313
?PR 25/5
5
```

Note:

```
?PR 3+4*2
11
```

but

```
?PR (3 + 4) * 2
14
```

Parentheses () tell Logo to perform what is within them first.

You can also write the name of the desired operation (**DIV, PRODUCT, SUM**) followed by the numbers to be figured. There is no *prefix* operation name for subtraction.

```
?PR SUM 3 4
7
?PR DIV 12 6
2
?PR PRODUCT 4 4
16
```

SINCLAIR LOGO NUMBERS

Logo can deal with both integers and fractions.

```
PR 25/6
4.1666667
?PR 4 * 2.3
9.2
?PR 19 - -2.5
21.5
```

Note the importance of the spaces in the expression 19 - - 2.5.
For more discussion about arithmetic in Logo, consult the Sinclair Logo *Programming Reference Manual*.

Chapter 14

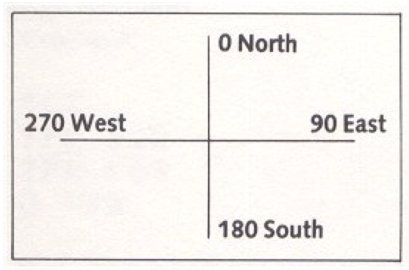
The turtle's field

INTRODUCTION

The turtle has a position and a heading.

Its heading is given in degrees like a compass, where 0°, or north, is facing straight up.

90° is directly east, 180° is directly south and 270° directly west. We might think of the screen as follows:



When Logo starts, the turtle's heading is 0. After CS, the heading is 0. You can find out the turtle's heading whenever you want.

```
?CS
RT 90
?PR HEADING
90
```

HEADING outputs the turtle's direction.

HEADING is a primitive procedure, but it is different from PRINT or FORWARD or the other commands we have seen.

HEADING is not a command; it is an operation. It does not cause something to happen; rather, it outputs something which can be used as an input.

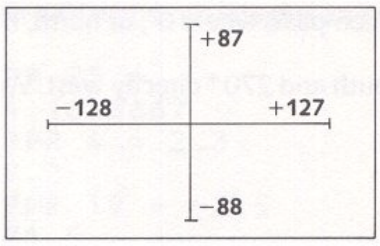
| If you don't tell Logo what to do with an operation, you will get a Logo message.

```
?PCS RT 90
?HEADING
You don't say what to do with 90
```

The turtle's position is described by two numbers, which indicate how far the turtle is from the centre of its field. When Logo starts, or after CS, the turtle's position is [0 0].

The first number indicates the turtle's location along the horizontal or x-axis. If the turtle is west of centre, the number is negative. The second number indicates the turtle's location along the vertical or y-axis. If the turtle is south of centre, the number is negative.

The turtle screen can be represented by a *grid* divided into *coordinates*. The x-coordinate runs along the horizontal and the y-coordinate runs along the vertical. The turtle at the centre has both XCOR and YCOR equal to 0. The screen dimensions, measured in turtle steps, are:



If you type:

```
?CS LT 90 FD 30
?PR POS
```

you will get:

```
 -30 0
```

If you now type:

```
?BK 60
?PR POS
```

you will get:

```
30 0
```

You can also find either coordinate by itself.

```
?PR XCOR
30
?PR YCOR
0
```

SETPOS, which stands for SET POSition, is a command that sets the turtle at a specific position on the screen. SETPOS is different from FORWARD or BACK in that the end result does not depend on the turtle's initial position. SETPOS does not change the turtle's HEADING. For example,

```
?SETPOS [50 - 52]
```

Leave a space before the -52, but do not leave a space between the - and the 52. If you do, Logo will think you are giving it three inputs, 50, - and 52. It will therefore send a Logo message:

```
SETPOS doesn't like - as input
```

WRAP, FENCE and WINDOW
The turtle starts out being able to WRAP; it can walk off one edge of the screen and reappear on the opposite edge along the same horizontal or vertical line. If it's facing at an angle, it will draw stripes as it moves. It does not change direction.
For example:

```
?CS
?FD 500
?PR POS
0 - 28
```

Notice that the turtle is not 500 steps from the centre.
By typing FENCE, you can set up the screen boundaries so that the turtle cannot move off the screen.
Type:

```
?FENCE
PCS
?FD 500
```

r Logo sends a message:

```
Turtle out of bounds
```

The turtle screen will act this way until you type WRAP or WINDOW.

WINDOW is a command which allows the turtle to move off the screen without wrapping. Thus the turtle may be invisible to you, but still carry out your orders. When you are in WINDOW mode, you may move the turtle up to +32767 or -32768 steps.
If you ask Logo to go more than that, you will receive a Logo message. For example:

```
?CS
?FD 50000
FD doesn't Like 50000 as input
```

USING POSITION TO DRAW

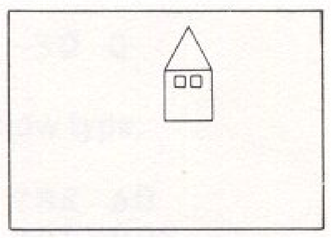
Now that we have learned about SETPOS and variables, we can add some features to our GARDEN.

1 Let's start by putting windows on our HOUSE.
Note: You cannot name a procedure WINDOW, because, as we have just seen, there is already a primitive of that name.

```
?ED "HOUSE
TO HOUSE
SQUARE 45 FD 45 TRIANGLE
PU SETPOS [10 26] PD
SETH 0
SQUARE 10
PU SETPOS [25 26] PD
SQUARE 10
END
HOUSE defined
```

Try it:

```
?HOUSE
```



2 We can also draw a little person who lives in the GARDEN. Let's make a stick figure.

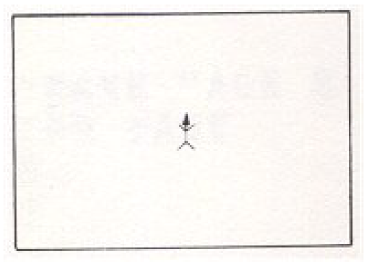
```
?TO V :SIZE
>LT 50
>DRAW :SIZE
>RT 100
>DRAW :SIZE
>LT 50
>END
V defined
```

```
?TO DRAW :SIZE
>FD :SIZE
>BK :SIZE
>END
DRAW defined
```

```
?TO PERSON :SIZE
>SETH 180
>V :SIZE
>RT 180
>FD :SIZE
>V :SIZE
>FD :SIZE /2
>END
PERSON defined
```

Try it:

```
?PERSON 10
```

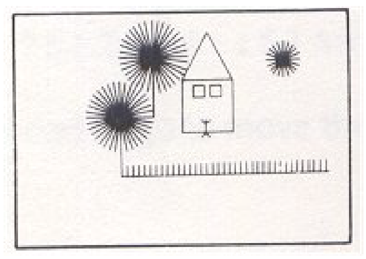


3 Let's now add these features to our GARDEN

```
?ED "GARDEN
TO GARDEN
WINDOW
HOUSE
PU SETPOS [-50 15] SETH 0 PD
TREES
PU SETPOS [-98 -45] PD
LAWN
PU SETPOS [90 60] SETH 0 PD
SUN
PU SETPOS [20 0] PD
PERSON 7
PR [WELCOME TO MY GARDEN]
END
```

Try it:

```
?GARDEN
```



Chapter 15

Assigning values to variables: the procedure MAKE

INTRODUCTION

In Logo, MAKE allows you to assign a value to a word or list. (Remember that a number is considered as a word in Logo.) In Chapter 12 we said that variables can be thought of as containers which contain Logo object(s) - a word or a list. Within the container we find the value that was given to the object.

```
?MAKE "AGE 8
?PR :AGE
8
```

the : tells Logo to look for the value assigned to the name AGE.

MAKE gives the value 8 to the name AGE.
MAKE needs two inputs. The first is the name of the variable; the second is its value.

USING MAKE TO DRAW

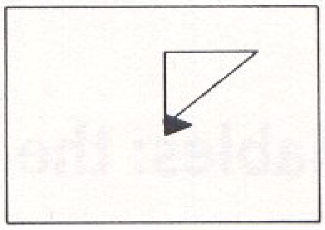
There is an easy way to draw a right angled triangle, provided that you know the lengths of the two sides forming the angle. Using the command MAKE, we can record the starting position of the turtle:

```
?CS
?MAKE "START POS
?PR :START
0 0
```

We can now ask the turtle to draw the two sides:

```
?FD 33
?RT 90
?FD 42
?SETPOS :START
```

We instruct Logo to move the turtle to the position indicated by the value of START.

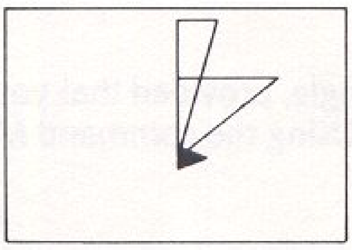


Since the pen is down, Logo draws a line.
We can write a procedure for this:

```
?TO TRE :SIDE1 :SIDE2
>MAKE "START POS
>FD :SIDE1
>RT 90
>FD :SIDE2
>SETPOS :START
>END
TRE defined
```

Try:

```
?CS
TRE 40 50
?SETH 0
?TRE 75 20
```



Chapter 16

More circles and arcs

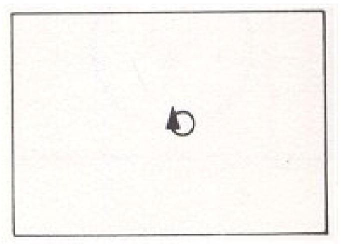
CIRCLES

Let's write a procedure for experimenting with circles of different sizes:

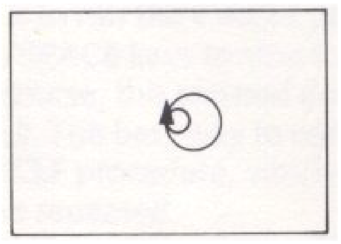
```
?TO CIRCLE :STEP
>REPEAT 36 [FD :STEP RT 10]
>END
CIRCLE defined
```

Now try it with various inputs:

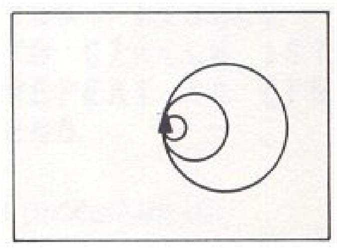
```
?CIRCLE  1
```



```
?CIRCLE 5
```



```
?CIRCLE 10
```



Notice that the circle's size changes in proportion to its input. This is not surprising because each circle has the same number of FORWARDs in it. The FORWARD distance determines the length of the circumference.

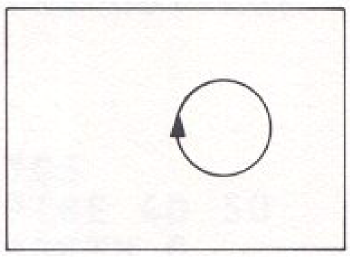
THE RADIUS

Sometimes it is more convenient to choose the size of a circle by stating its radius - the distance from the centre to any point on the circumference. With the **CIRCLE** procedure we need to calculate the radius of each circle. Why not let Logo calculate it? To do this, we'll write another procedure which uses **CIRCLE**, and call it **CIRCRAD**.

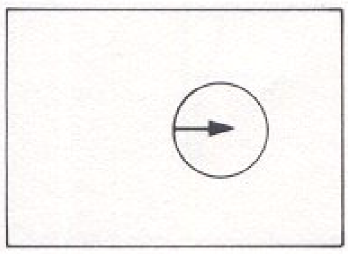
```
?TO CIRCRAD :RADIUS
>CIRCLE 2 * 3.14 * :RADIUS/36
>END
CIRCRAD defined
```

Note: **2 * 3.14 * : RADIUS** represents the circumference of a circle (2 π r). The circumference has **36 FORWARDS**. Thus we divide by 36 to get the step size.
Now try:

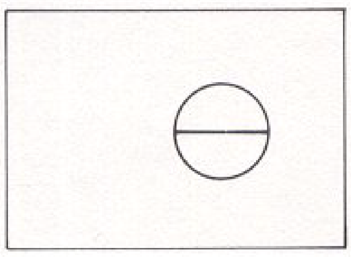
```
?CIRCRAD 30
```



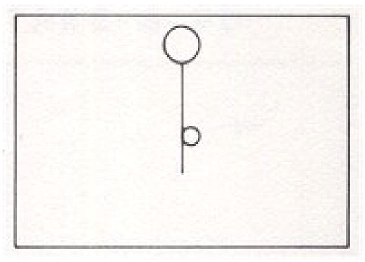
```
?RT 90 FD 30
```

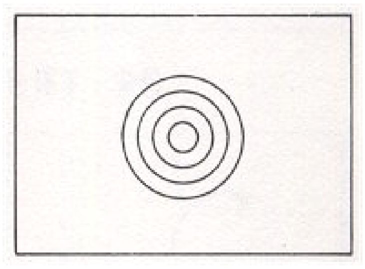


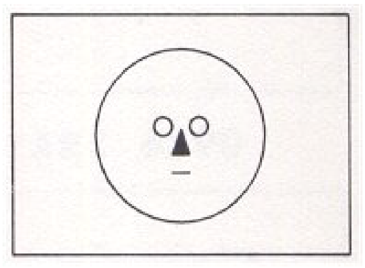
```
?FD 30 HT
```



Here are some drawing using circles. See if you can write programs for them!


FLOWER


TARGET


FACE

ARCS

Many projects require only arcs (pieces of circles). One way to draw an arc of a circle is to run the CIRCLE procedure and quickly press the CAPS BREAK/SPACE keys to stop the turtle before it finishes drawing.

Of course, this method doesn't allow you to control the size of your arcs very well. The best way to control the size of an arc is to give another input to the CIRCLE procedure, which varies the number of times the small steps and turns are repeated.

Let's change the name of the procedure to ARC.

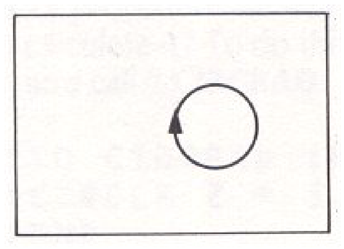
```
?ED "CIRCLE
TO CIRCLE :STEP
REPEAT 36 [FD :STEP RT 10]
END
```

Edit this procedure to:

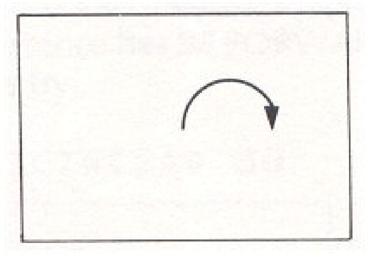
```
TO ARC :STEP :TIMES
REPEAT :TIMES [FD :STEP RT 10]
END
ARC defined
```

Now try:

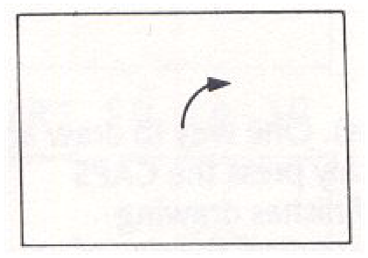
```
?ARC 10 36
```



```
?CS
?ARE 10 18
```



```
?CS
?ARC 10 9
```

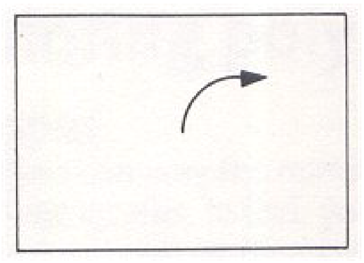


We can use the number of degrees that we want in our arc as the input and let Logo calculate how many times to repeat.

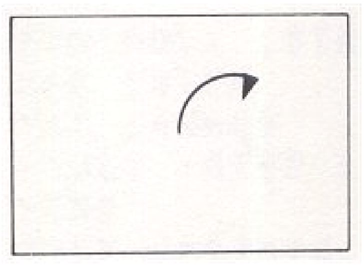
```
?ED "ARC
TO ARC :STEP :DEGREES
REPEAT :DEGREES/10 [FD :STEP RT 10]
ARC defined
```

USING ARCS

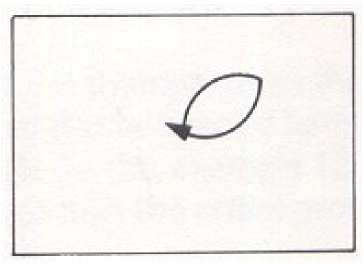
```
?ACR 6 90
```



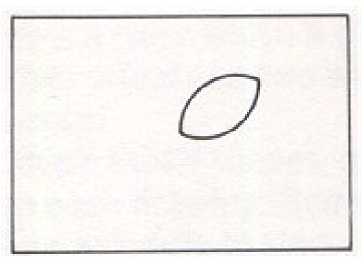
```
?RT 90
```



```
?ARC 6 90
```



```
?HT
```



A petal!

Now try using some negative number.

Chapter 17

Exploring polygons and spirals

POLYGONS

Just as you can vary the number of steps the turtle takes, you can also vary how much it turns. In fact, you can get beautiful and surprising designs by varying these two components of the turtle's state.

Let's look at some examples:

```
?TO POLY :STEP :ANGLE
>FD :STEP
>RT :ANGLE
>POLY :STEP :ANGLE
>END
POLY defined
```

Try:

```
?POLY 30 120
```

To stop the triangle, press the CAPS BREAK/SPACE.

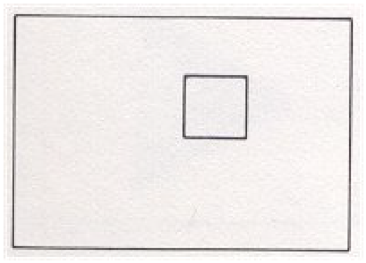
What has happened here? After turning right the number of degrees in the angle (in this example 120), the procedure calls itself as an instruction, and Logo runs the entire procedure again and again and again, until you tell it to stop.

A procedure which calls itself as a subprocedure is known as a recursive procedure.

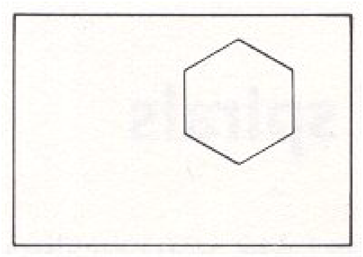
There is a story which is often told to explain recursivity. A fairy godmother offers you two wishes. Your second wish is always to have two more wishes!

Try these POLY shapes, or use inputs of your own. It's a good idea to CS between each drawing. Remember to press CAPS BREAK/SPACE when you want the procedure to stop.

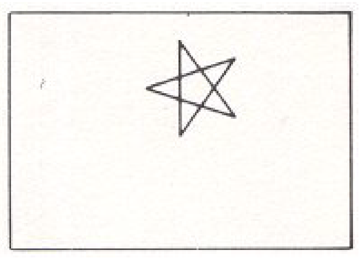
```
?POLY 30 90
```



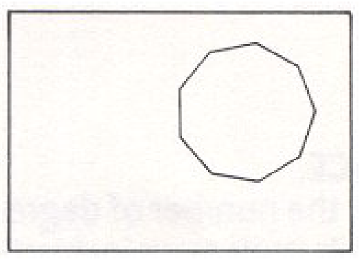
```
?POLY 30 60
```



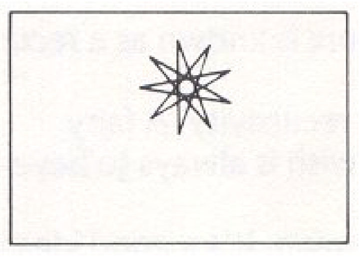
```
?POLY 30 144
```



```
?POLY 30 40
```



```
?POLY 30 160
```



Let's now make a polygon which turns and changes its colour!

```
?TO POLYT :N :SIDE :ROT
>POLY1 :N :SIDE
>RT :ROT
>SETPC PC+1
>POLYT :N :SIDE :ROT
>END
POLYT defined

?TO POLY1 :N :ANGLE
>REPEAT :N [FD :SIDE RT 360/:N]
>END
POLY1 defined
?POLYT 6 40 30
```

Note that when the pen has the same colour as the background, the polygon is invisible.
To stop POLYT, press CAPS BREAK/SPACE.

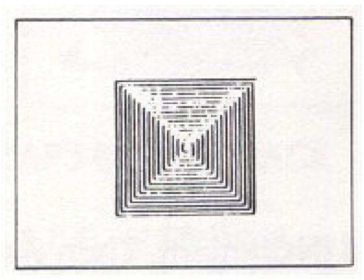
SPIRALS

The POLY procedure draws closed figures. The turtle moves forward and rotates so that it eventually gets back to where it started. (However, if the turtle turns 0 or 360° - or a multiple of 360° - on each round, it walks in a straight line.)
To draw a spiral, the turtle must not return to where it started; instead, it should increase its forward step on each round so that it gets further and further away from its starting point.
We can make it do this by adding a little bit to :STEP each time POLY instructs itself to start the procedure again, ie, on the recursion instruction.

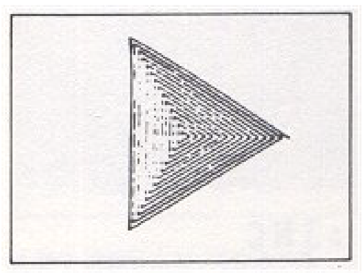
```
?TO SPI :STEP :ANGLE
>FD :STEP
>RT :ANGLE
>SPI :STEP+6 :ANGLE
>END
SPI de-fined
```

Now try SPI!

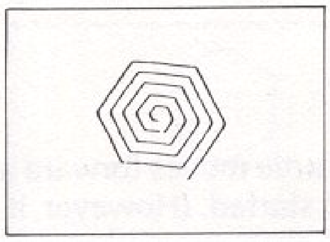
```
?HT
?SETSCRUNCH [50 50]
?SPI 5 90
```



```
?SPI 5 120
```

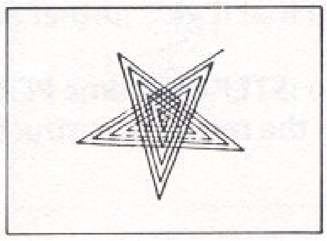


```
?SPI 5 60
```

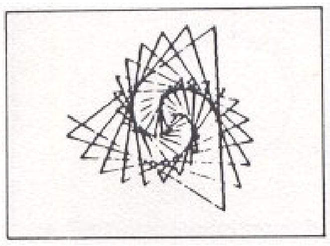


Remember, type **CAPS BREAK/SPACE** to stop the **SPI** procedure.

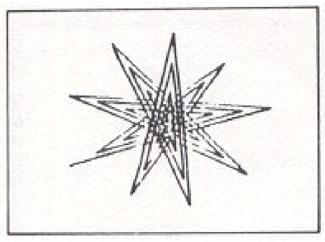
```
?SPI 5 144
```



```
?SPI 5 160
```



```
?SPI 5 160
```

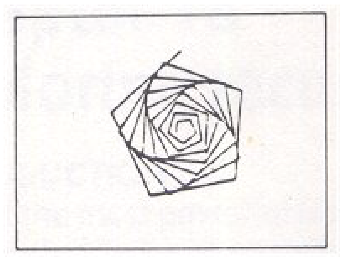


Let's now modify **SPI**, giving it a third input called **INC**rement. Then we can change how much the turtle's step increases by choosing different numbers for the third input.

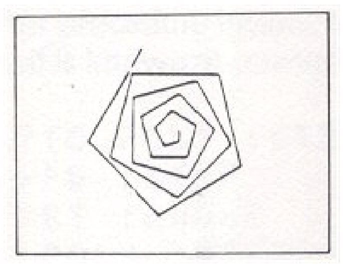
```
?ED "SPI
TO SPI :STEP :ANGLE :INC
FD :STEP
RT :ANGLE
SPI :STEP + :INC :ANGLE :INC
END
SPI defined
```

Now try:

```
?SETSCRUNCH   [100 100]
?SPI  5  75  1
```

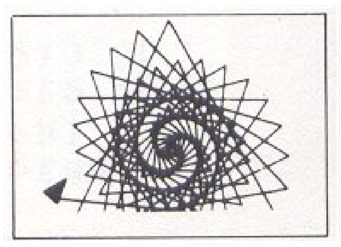


```
?SPI  5  75  2
```



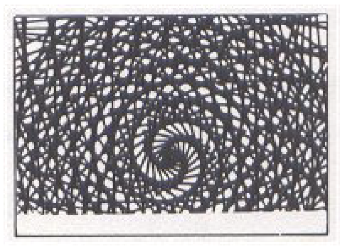
Try stopping the turtle at different places. Make up your own inputs. You can
Also try:

```
?CS
?FENCE
?SPI  5  125  2
```



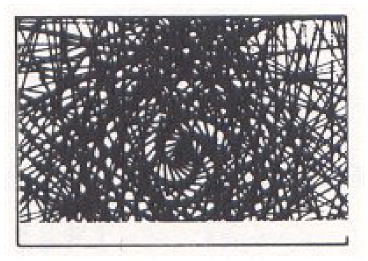
Press CAPS BREAK/SPACE

```
?CS WINDOW
?SPI  5  125  2
```



Press CAPS BREAK/SPACE

```
?CS
?WRAP
?SPI  5  125  2
```



Chapter 18

Exploring recursive procedures

INTRODUCTION

One of the most powerful features of Logo is that you can divide a complicated task into procedures, each of which has its own name and is completely separate from the others. A procedure can call, or be called by, any other procedure including itself. As we have seen, a procedure which calls itself is known as a recursive procedure.

```
?TO POLY :STEP :ANGLE
>FD :STEP
>RT :ANGLE
>POLY :STEP :ANGLE (this is the recursive call)
END
POLY defined
```

POLY calls POLY as part of its definition.

Recursion allows repetition of a procedure. Recursive calls may be directly within the procedure (as in POLY), or may cross procedures, for example:

```
?TO GO          ?TO HI
>FD 10          >PE BK 10PD
>HI             >GO
>END            >END
```

GO calls HI and HI calls GO ... until you tell Logo to stop by pressing CAPS BREAK/SPACE.

Not all recursive procedures work this way; they can be made to stop. In fact, making up appropriate 'stop rules' is an important part of writing recursive procedures. We will look at some stop rules here; consult the *Programming Reference Manual* for further examples.

STOPPING RECURSIVE PROCEDURES

Let's look at some ways of stopping recursive procedures within a program.

Example 1: Stopping the SPI procedure

```
?TO SPI :STEP :ANGLE :INC
>IF :STEP >150 CSTOP3
>FD :STEP
>RT :ANGLE
>SPI :STEP + :INC :ANGLE :INC
>END
```

In this example we have told Logo to stop if the size of the step is greater than 150.

In brief, the **statement IF :STEP >150 [STOP]** can be translated as:

If the value of **STEP** is greater than **150**, **STOP** the procedure; if not, continue executing the procedure. Now try:

```
            SPI 5 125 10
```

IF expects its first input to be either **TRUE** or **FALSE.**

> is a special kind of operation, which outputs either **TRUE** or **FALSE**. We call this kind of operation a predicate. Predicates are used as the first input to **IF**; see the *Programming Reference Manual* for a more detailed discussion.

Example 2: This recursive procedure introduces two new primitives: **FIRST** and **BUTFIRST**

FIRST and **BUTFIRST** deal with Logo objects (words and lists). **FIRST** instructs Logo to look for the first element of a word, or the first element of a list. **BUTFIRST** instructs Logo to look for everything **BUT** the first letter of a word or list.

There are many primitive procedures to put Logo objects together, and to take them apart and examine them. See the section WORDS and LISTS in the *Programming Reference Manual*.

```
            ?TO VERTICAL :WD
            >IF :WD = " ESTOP]
            >PR FIRST :WD
            >VERTICAL BUTFIRST :WD
            >END
            VERTICAL defined
            ?VERTICAL "NONSENSE
            N
            O
            N
            S
            E
            N
            S
            E
```

What happens when we **VERTICAL "NONSENSE?**

1 he instruction **IF** :**WD = " [STOP]** tells Logo to **STOP** if the value of **WD** is the *empty word* (a **"** followed by a blank space).

2 f it is not, Logo goes to the second instruction, **PRINT FIRST :N**, which ells Logo to print the first character of :**WD**.

3 The third instruction, VERTICAL BUTFIRST :WD is a recursive call, and ells Logo to look for the procedure VERTICAL "ONSENSE which is the UTFIRST of VERTICAL "NONSENSE.

4 his continues until the value of :WD is empty. The IF statement is then rue, and Logo stops.

Example 3: A recursive twist
Simple recursion is quite simple. But sometimes recursion can be quite complex, even subtle! If you would like to see an example look at the procedures below. If not, just skip this section.

These two procedures appear quite similar:

```
?TO COUNTS :N          ?TO ACCOUNT :N
>IF :N = 0 ESTOP]      >IF :N = 0 ESTOP]
>PRINT :N              >ACCOUNT :N - 1
>COUNTS :N - 1         >PRINT :N
>END                   >END
COUNTS defined         ACCOUNT defined

?COUNTS 3              ?ACCOUNT 3
3                      1
2                      2
1                      3
```

The COUNTS :N procedure instructs Logo to stop when the value of :N is 0, otherwise print N and re-execute the procedure, subtracting 1 from the value of N.

This procedure is quite straightforward and similar to the other examples we have examined.

The ACCOUNT procedure is more complicated.

1 The first instruction tells Logo to check if the value of :N is 0. If yes, the procedure stops, If no, Logo continues to the next instruction.

2 The second instruction tells Logo to look for the procedure ACCOUNT which has an input of :N -1 (the value of N minus 1).
But what has happened? The second instruction is a recursive call instructing Logo to start the procedure again. Logo cannot proceed to the next instruction until the IF statement (IF :N=0) is true. When :N = 0, Logo will no longer be able to execute the ACCOUNT :N-1 instruction.
It is only then that Logo passes to the next instruction.

3 PRINT :N.
But Logo now has several N values to print, which are all waiting in Logo's memory. Logo prints the number of the last instruction it carried out.
Thus, we see a 1 on the screen.

4 The next instruction is **END**. Logo cannot yet end because it still has some values in its memory. So, it passes back one procedure. This was **CCOUNT** with the value of **2**. Thus, it prints a **2** on the screen. It does his until there are no more unfinished instructions left in the procedure.

Thus, in placing a recursive call in the midst of a procedure, rather than at the end, there may be several 'results' existing at the same time. In this case, the last procedure called is the first one to stop.

Chapter 19

A game project

CREATING A GAME

Let's make up a game. A target and a turtle appear somewhere on the screen. The player tries to get the turtle into the target with the smallest number of moves.

For our first version, we will use regular Logo commands such as **LT 45** or **FD 80**. Later, we will refine the game by assigning Spectrum keys to direct the turtle. Developing the game in stages illustrates the kind of 'project management' to which Logo is well suited.

First, we need to set up a target; then we need to set up the turtle. We can write one procedure that will perform both tasks. An example of a **SETUP** procedure is printed below. **SETUP** sets the turtle up in a random position on the screen. It leaves the turtle heading in the same direction as it was at the start of **SETUP**.

```
?TO SETUP
>PU
>RT RANDOM 360
>FD RANDOM 85
>SETHEADING 0
>PD
>END
SETUP defined
```

The Logo operation **RANDOM** returns a number which Logo chooses randomly between 0 and one less than the number given as **RANDOM**'S input.

In **SETUP**, for example, the turtle turns some angle which can be as small as 0 or as large as 359. The actual number is computed each time RANDOM is used. The input to FD is also a random number. Here the number can be no larger than 84. Notice that **SETUP** leaves the turtle facing north.

SETUP can be used to set up the turtle as well as the target. It's a good idea to put the turtle back at the centre first.

The following procedure, **SETGAME**, sets up the game.

```
>TO SETGAME
>CS
>SETUP
>TARGET
```

```
>PR [TRY TO HIT THE TARGET]
>SETPOS [O 0]
>SETUP
>END
SETGAME defined

?TO TARGET
>BOXR 10
>END
TARGET defined

?TO BOXR :SIDE
>REPEAT 4 [FD :SIDE RT 90]
>END
BOXR defined
```

Try SETGAME a few times. It's hard at first. For example:

```
?SETGAME
?RT 45
?FD 100
```

A miss!

MAKING A KEY INTO A GAME BUTTON

You can write many kinds of interactive programs. You can ask Logo questions and receive answers in words or sentences: see Chapter 9 of the *Programming Reference Manual,* Interaction with the Machine, for further examples and explanations.

Sometimes you may want to trigger Logo into action by a touch of a key. This requires the operation READCHAR or RC. Type:

```
?PR RC
```

Logo waits for a key to be pressed. Type the letter A. RC receives the Character A and passes it to the PRINT command. The PRINT command them Puts an A on the screen.

```
?PR RC
A
```

Logo does not wait for you to do anything else. It acts immediately Try RC a few more times. Note that if you type RC (followed by ENTER) ,and then type in a character (for example T),Logo sends a message :

```
?RC
Now type T
You don't say what to do with T
```

RC is an operation, like HEADING or POSITION. It is used as an input to another command or operation. We could name RC's output using MAKE, for example:

```
?MAKE "KEY RC
```

Now type the character z. :KEY will be the character z. To verify, type:

```
?PRINT :KEY
Z
```

We can use this idea of giving things names so that we can talk about them. Imagine we have a procedure called PLAY.

```
?TO PLAY
>MAKE "ANSWER RC
>IF ANSWER = "F [FD 10]
>IF ANSWER = "R [RT 15]
>IF ANSWER = "L [LT 15]
>PLAY
>END
PLAY defined
```

F makes the turtle move forward 10 steps.
R makes the turtle turn right 15°.
L makes the turtle turn left 15°.

In PLAY, the value of :ANSWER is what RC outputs. PLAY then checks :ANSWER using the Logo primitive, IF. IF requires two inputs. The first input is either TRUE or FALSE. The second is a list of instructions to be carried out when the first input is TRUE.

Notice that PLAY is recursive; that is, the last line of the procedure PLAY calls PLAY. PLAY does not stop unless it has a bug or you press CAPS BREAK/SPACE. Try it.

EXPANDING THE GAME PROJECT

In this section we build a better target game out of SETGAME and PLAY. Some of the techniques used in this section are new. We can write a procedure, GAME, which uses SETGAME and then PLAY:

```
?TO GAME
>SETGAME
>PLAY
>END
GAME defined
```

Try GAME.

Perhaps we should raise the turtle's pen. We can ask GAME to print some instructions:

```
?TO GAME
>RULES
>SETGAME
>PU
>PLAY
>END
GAME defined

?TO RULES
>PR [HIT THE TARGET WITH THE TURTLE]
>WAIT 100
>PR [TYPE R OR L TO TURN AND F TO ADVANCE]
>WAIT 100
>END
RULES defined
```

Try GAME now.

This is much better, but there is still room for improvement. The game plays too slowly; let's make it more challenging by giving the player only one chance to land on the target. The player can turn the turtle many times, but will have only one chance to tell it how far to go forward.

Here is the plan: after Logo has set up the scene for the game, we want it to let you play. Once you've had your try, you can see if you've landed in the target. Logo should leave the screen unchanged for a little while and then start the game again with a brand new target and position.

We use a 'top - down' approach to plan this game. That means we plunge in and write the overall structure of the game before we know how we are going to write all the details.

```
?TO GAME
>RULES
>SETGAME          (This sets up each game)
>PU
>PLAY
>WAIT 100          (Logo pauses before restarting)
>GAME
>END
GAME defined
```

We can now edit the procedure PLAY to give you only one chance to move the turtle forward into the target. The point of the game is to judge the distance.

When you press the T key (T for try), you get your only chance to land in the target.

```
?TO PLAY
>MAKE "ANSWER RC
>IF ANSWER = "R  [RT 15]
>IF ANSWER = "L   [LT 15]
>IF ANSWER = "T   [TRYLANDING ST
OP]
>PLAY
>END
PLAY defined
```

Now edit the RULES and change F to T.

```
?ED "RULES
TO RULES
PR [HIT THE TARGET WITH THE TURTLE]
WAIT 100
PR [TYPE R OR L TO TURN AND T
 TO TRY LANDING]
WAIT 100
END
```

We've used the 'top-down' approach again; we've asked PLAY to call a procedure name TRYLANDING which we haven't yet defined.

Let's define it now.

```
?TO TRYLANDING
>PR [HOW FAR DO YOU WANT TO MO
  VE FORWARD?]
>FD READWORD
>END
TRYLANDING defined

?TO READWORD
>OUTPUT FIRST READLIST
>END
READWORD defined
```

READWORD functions like RC, except that you can type a word instead of a single character: it waits for you to press the ENTER key to signal that you have done so.
READWORD is an operation which returns the a word you typed.
READLIST (RL) is a primitive procedure; it too is an operation, but it returns a list. So READWORD uses READLIST, but only takes the first word you type

Operations return a word or a list. The command OUTPUT or OP outputs something, and stops the procedure at that point.

Now we have written the whole game. To try it, type:

```
?GAME
```

Remember, you can give the commands R and L to turn the turtle and T to try landing on the target. After you type T, Logo will wait for you to type a number and then ENTER.

You may try adding to this program yourself!

TAILPIECE

We have now come to the end of our introduction to Logo, but we hope you will explore other turtle projects on your own. The *Programming Reference Manual* describes many other features of Logo which you will want to try as you become more familiar with the language.

Index