REPEAT PERFORMANCE

This is the final instalment in the tutorial section of our series on LOGO programming. Here we show you how to add new control structures to the language, and explain how to write procedures that can themselves write procedures.

The LOGO primitive RUN takes a list as its input, and causes this list to be executed just as if it were a line of a procedure. This can be used to add new control structures to the language as and when they are required. So we could define a WHILE procedure as follows:

```
TO WHILE :CONDITION :ACTION
   IF NOT ( RUN : CONDITION ) THEN STOP
   RUN :ACTION
   WHILE :CONDITION :ACTION
END
```

Here's an example of how we could use it. POWER prints all the powers of its input below 1000:

```
TO POWER :X
   MAKE "P :X
   WHILE [:P < 1000] [PRINT :P MAKE "P :P * :X]
END
```

Control structures, such as WHILE, REPEAT and FOR, are common in other languages, but they are not really necessary in LOGO. A more natural way to write POWER in LOGO would be:

```
TO POWER :P
   IF NOT :P < 1000 THEN STOP
   PRINT :P
   POWER P * :P
END
```

REPEAT is provided in all versions of LOGO, but it is not strictly necessary, since you could define an equivalent word, REPT, in the following way:

```
TO REPT :NO :LIST
   IF :NO = 0 THEN STOP
   RUN :LIST
   REPT :NO - 1 :LIST
END
```

RUN is an extremely useful primitive for more advanced LOGO work. A program can assemble a list and then pass it to RUN to have it obeyed. We'll see an example of this shortly.

TAKING PROCEDURES APART

First of all we must define a procedure to draw a triangle in the usual way:

```
TO TRI
   FD 50 RT 120 FD 50
   RT 120 FD 50 RT 120
END
```

Now type PRINT TEXT "TRI. The result should be:

The text of the procedure is given as a list of lists, where each 'inner' list is one line of the procedure. To see why there is an empty list at the start, define this replacement for addition:

```
TO ADD :A :B
   PRINT :A + :B
END
```

Now PRINT TEXT "ADD will give:

```
[:A :B][PRINT :A + :B]
```

Clearly, the first list contains the inputs for the procedure. So TEXT enables us to get inside a procedure and find out what is there. DEFINE, on the other hand, does the opposite: it lets us define a procedure as a list of lists without having to go into the editor. Now try DEFINE "L [[:A] [FD :A] [RT 90] [FD :A / 2]] and then run L using, for example, L 30. Using DEFINE in immediate mode in this way has no advantages over using the editor. The advantage that DEFINE gives us is the ability of one procedure to create another procedure.

GROWING

We are now going to develop a small system for investigating growth. The basic commands in our system are ASK, which selects the shape we will deal with, and GROW, which changes the size of the chosen shape. For example, ASK "SQUARE will draw a square, and then GROW [* 10] will erase the square and then redraw it with each of its sides increased by a factor of 10.

To keep the programs simple we will have to accept a few restrictions on what we can do with these commands. Firstly, the shape procedures given as inputs to ASK may not contain REPEAT or call subprocedures. Secondly, the system will break down if you get negative results. Neither of these problems is very difficult to deal with if you should wish to improve on what we give here.

ASK works by assigning the name of the shape to the global variable "CURRENT and then running the procedure. It does this by creating a list of one item — the procedure name — and then using RUN to execute it.

```
TO ASK :OBJECT
   HIDETURTLE
   MAKE "CURRENT :OBJECT
   RUN ( LIST :OBJECT )
END
```