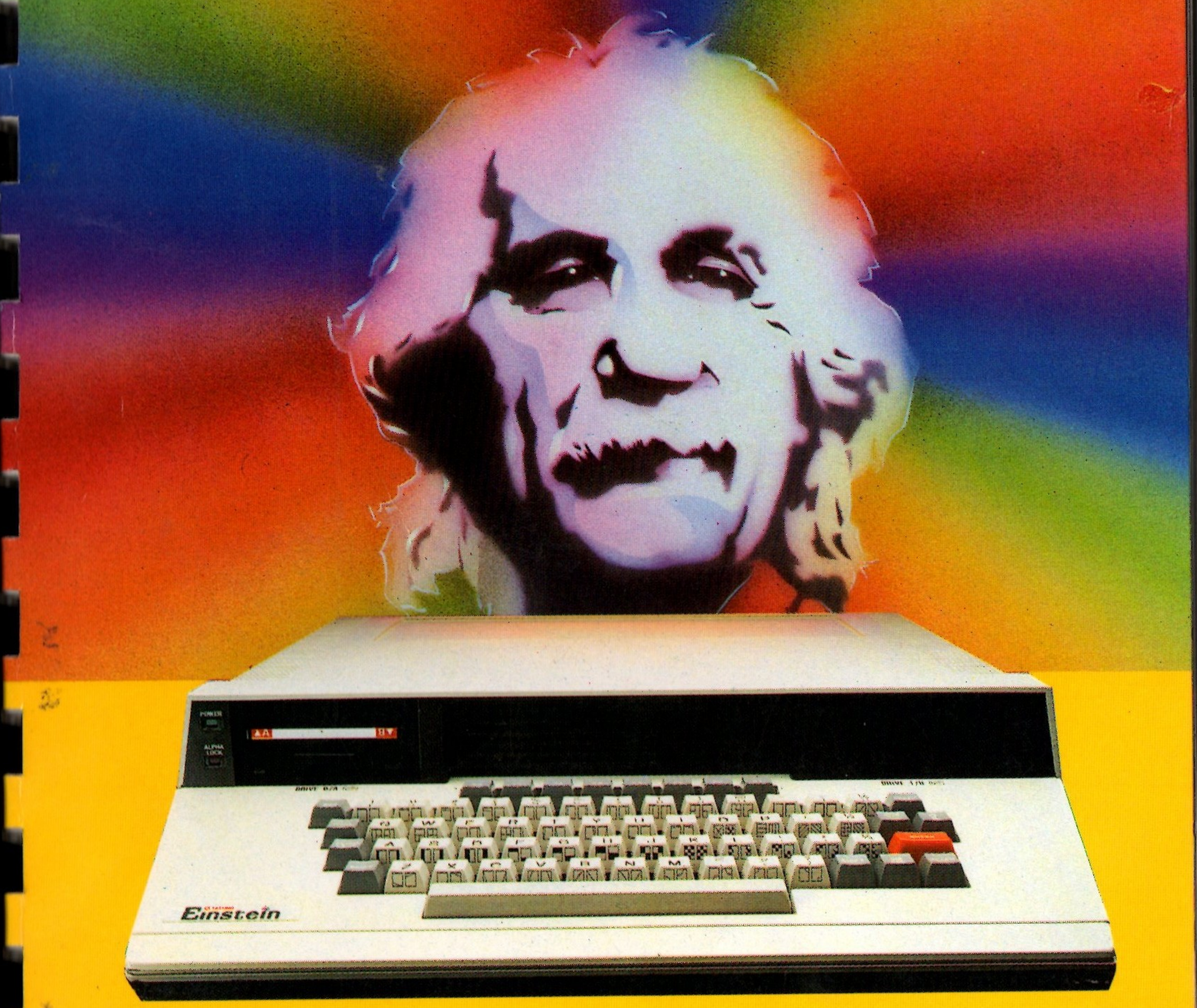
Dr. Logo Introduction ™



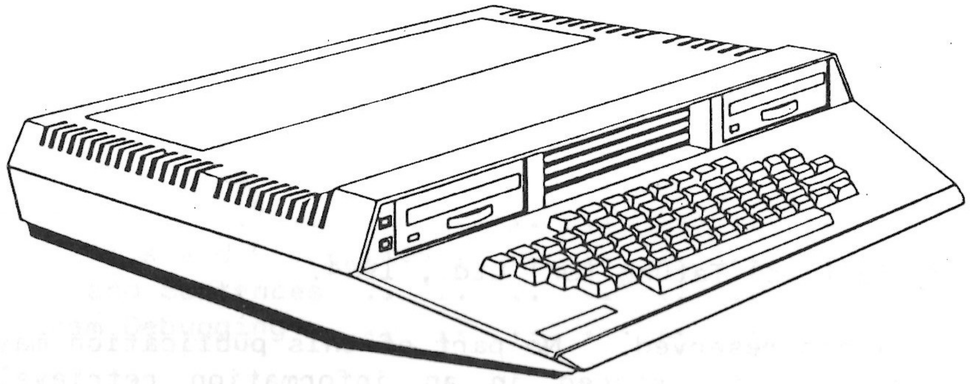
TATUNG

Einstein

COLOUR MICRO COMPUTER

TATUNG

Einstein



Dr. Logo
Introduction

ACKNOWLEDGEMENTS

Written By: J. Rangeley
Edited By: R.M. Clarke

Our grateful thanks to Digital Research Inc. for their kind permission to reproduce extracts from their manuals.

ISBN:1-85087-014-9. 1st Edition 1984
ISBN:
ISBN:
ISBN:

Tatung (UK) Ltd.,
Computer Division,
Stafford Park 10,
TELFORD, Shropshire, TF3 3BH.

First Printed in 1984 by:
Spellman Walker Ltd.,
Bradford, West Yorkshire.
ENGLAND

CONTENTS

A D D E N D U M

Information on the following primitives became available too late for their inclusion in the alphabetical list.

EQUAL PREDICATE

Syntax: **equalp** object object

Purpose: Outputs TRUE only if the input objects are equal numbers, identical words or identical lists; otherwise outputs FALSE.

Example:
```
?equalp 60 5*12
TRUE
?equalp [hello] [goodbye]
FALSE
```

GET LIST

Syntax: **?glist** propname

Purpose: Outputs a list of all objects in the workspace that have the input property in their property lists.

Example:
```
?make "one "two "three
?glist "two
one
?glist ".DEF
[square triangle]
```

GENERAL INTRODUCTION

This manual is divided into two sections. Section A introduces some of the more common commands of Dr. Logo and each is accompanied by some simple examples. Section B details all the Dr. Logo commands that are available on EINSTEIN, again with short examples.

Dr. Logo has a vocabulary of approximately 200 words, called primitives. Some are for arithmetic operations, some for graphics and others for creating and changing lists of information. One of the most powerful features of Dr. Logo is that new words (called procedures) can be added to the vocabulary using a sequence of primitives. The language encourages a well structured style of programming on the user.

The examples in Section A use only some of the many primitives that are available. However, they will indicate the versatile nature of Dr. Logo and provide 'food for thought' when developing your own programs and experimenting with some of the commands given in Section B.

This manual serves merely as an introduction to Dr. Logo and the following books are recommended for a more detailed understanding of the language.

Meet Dr. Logo Tutorial
Dr. Logo Language Reference Manual

SECTION A

LANGUAGE INTRODUCTION

Loading Dr. Logo

Dr. Logo is located on side A of the Einstein System Master Disc. It is recommended that this is copied onto another disc before attempting to use Dr. Logo. (See DOS/MOS manual, page 40).

Insert the disc containing the file LOGO.COM in drive 0 and press **CTRL-BREAK** to load the Disc Operating System (DOS) in the normal way (See page 28 of 'An Introduction to Einstein').

Type **LOGO** and wait until the program is loaded. A sign on message similar to the following should appear.

```
Welcome to
Z80 DR. LOGO, EINSTEIN VERSION 1.0
Copyright (c) 1983, Digital Reasearch,
Pacific Grove, Califonia.

Dr. Logo is a trademark of
Digital Research
Please wait
```

After a short delay, the screen will clear and a '?' will be displayed at the top lefthand corner of the screen. This is the Dr. Logo prompt and indicates that the machine is awaiting instructions.

To exit Dr. Logo type **bye** and then press **ENTER** whenever the ? prompt appears. This will return the computer to DOS.

General Instructions

When using the EINSTEIN computer with Dr. Logo, three of the keys on the keyboard generate characters which are different to those indicated on the key caps. These are as follows:

Key cap symbols	Dr. Logo characters
←	[
→	]
½	\

In the following examples, the characters to be entered by the user are underlined; the computers response is shown in normal type. Each line entered by the user should be terminated by pressing the **ENTER** key, except when in edit mode. Care should be taken when typing, to include spaces as required. Spaces are used to separate primitives and associated characters. If you make typing mistakes use the **DEL** key to erase the last character entered.

To exit a program use the **ESC** key: the computer will then reply with the message **'stopped'**.

Arithmetic

Arithmetic operations can be performed using '*' to represent 'multiply' and ' ' to represent 'divide'.

e.g. ?(2 + 3) *5
25

Calculations can be performed from within a program using the **pr** (print) primitive.

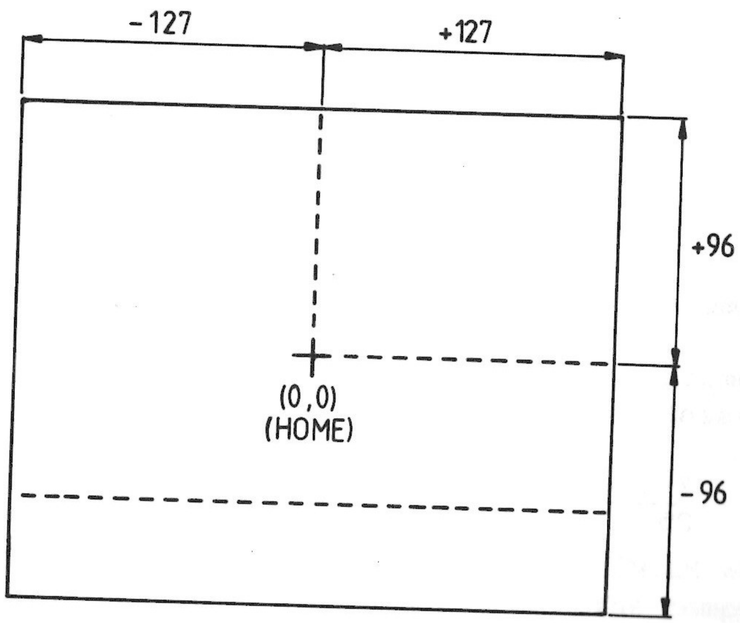
e.g. ?pr 6 + 9
15

Commands and Turtle Graphics

A command is a line of instructions terminated by pressing **ENTER.** If Dr. Logo cannot understand what is typed, an error message is displayed.

e.g. ?hello
I don't know how to hello

The screen is divided into steps as shown in the following diagram.



DR. LOGO SCREEN CO-ORDINATES IN FULL SCREEN MODE

To draw graphics on the screen, use the **ss** (split screen) primitive. This divides the screen into two areas; the lower part for commands; the upper part for drawing the graphics. To change the screen back to normal, use the **ts** (text screen command).

Graphics are drawn using a 'turtle' to guide a 'pen' about the screen. The main commands for moving the turtle are **fd** (forward), **bk** (backward), **rt** (right) and **lt** (left). These primitives are followed by a number, degrees for the **lt** and **rt** commands, and steps for the **fd** and **bk** commands.

e.g.
```
?ss
?fd 50
?lt 90
?fd 50
?bk 50
?rt 180
?fd 50
```

This will draw a T - shape on the screen.

NOTE: In split screen mode the lower portion of the screen cannot be used for plotting graphics. If the **setscrunch** command is used the vertical co-ordinates will be modified.

e.g. ?setscrunch 1.5

This would set the vertical axis to -64 to +64, the horizontal axis would remain set to -127 to +127.

Having drawn a pattern it may be necessary to erase it. This is performed by the **cs** (clear screen) primitive.

It may also be necessary to move the turtle without drawing a line. This is achieved using the **pu** (pen up) command. Normal turtle drawing is restored using the **pd** (pen down) command.

e.g.
```
?cs                clear screen
?lt 90 fd 50       left 90°, forward 50 units
?pu bk 150         pen up, back 150 units
?pd fd 50          pen down, forward 50 units
```

This will draw two separate horizontal lines.

In drawing some patterns it is desirable to not see the turtle. This can be achieved by **ht** (hide turtle) and can be made to reappear with **st** (show turtle).

Another primitive that is useful when creating repetitive patterns is **repeat.** This requires two inputs; a number and a list of instructions in square brackets.

e.g.
```
?cs
?repeat 4 [fd 50 rt 90]
```

This will produce a square on the screen.

Procedures

A procedure is a series of instructions and can be defined using the primitive **to.**

e.g.
```
?to square
>repeat 4 [fd 30 rt 90]
>end
square defined
```

Note the > prompt is used by the computer to indicate you are required to enter a command as part of a

procedure. The **end** primitive returns the system to the ? prompt after issuing the message 'square defined'.

Now, whenever square is typed, a square will be drawn on the screen. In this way new words can be added to Dr. Logo's vocabulary.

Procedure names may not contain spaces and cannot be the name of a primitive.

Editing a Procedure

Having created a procedure it may be necessary to modify it at some stage. This may be performed using the **ed** command in the following form: ed "procedure. The procedure will then be listed on the full text screen.

e.g.
```
?ed "square
to square
repeat 4 [fd 50 rt 90]
end
```

The procedure may then be edited by moving the cursor to the character to be changed and then retyping and/or deleting, as appropriate. Note the editor is always in the 'insert' mode so that any new characters will be added at the cursor position rather than overwriting existing text. The cursor movement and screen display can be further manipulated by using the CTRL key. A full list of control codes are given in Appendix A.

When you are satisfied with the text press **CTRL-C** to leave the editor and change the procedure to the

version you have just typed. If it is required to exit without making any changes to the procedure use the ESC key.

Note: It is also possible to screen edit the name of the procedure so as to create a new procedure.

Loading and Saving Procedures

Having defined a procedure it is usually required to make a permanent record on file. All procedures are stored in the computer workspace and this area can be transferred to disc by using the command: **save** "filename

A valid filename contains up to 8 alpha-numeric characters. This command will write to your disc. Use a copy of your master disc and set the write protect tabs to the unprotected position.

Each time the computer is used, the procedures saved on disc can be recalled using the command: **load** "filename. As the program is loaded a list of all the procedures will be displayed as they are loaded into the workspace.

The workspace contents can be reviewed using the primitive; **pots** (print out titles) and **po** (print out).

e.g. ?pots lists all the procedures in the workspace
```
to square
to line
```

e.g. ?po "square lists procedure square
```
to square
repeat 4 [fd 50 rt 90]
end
```


Procedures can be deleted using the **er** (erase) and the **erall** (erase all) primitives.

e.g. ?er "square erases the procedure square

?erall erases all procedures from the workspace.

Procedures with Inputs

Procedures may be written that use a variable which is passed to the procedure when it is used. This is achieved by using a colon (:) followed by the variable name (referred to as the placeholder).

e.g.
```
?to stair :step
>repeat :step [fd 10 rt 90 fd 10 lt 90]
>end
stair defined
```

This example uses the variable 'step' to define the number of times the repeat is executed. To use the routine type stair followed by a number.

e.g. ?stair 3

This will draw a staircase with three steps.

More than one input can be used with procedures.

e.g.
```
?to vsquare :size
>repeat [fd :size rt 90]
>end
vsquare defined
```

```
e.g.   ?to squares :a :b :c
       >vsquare :a
       >vsquare :b
       >vsquare :c
       >end
       squares defined.
```

Type squares 50 60 70 to draw three squares on the screen.

Procedures with Outputs

Procedures that produce a result need a method of presenting that result. This is performed by the **op** (output) primitive.

```
e.g.   ?to add :x :y          input two values
       >op :x + :y            add and output
       >end
       add defined.
```

This procedure adds two numbers. To use it, type **add**, followed by the two numbers, then press **ENTER**. The two numbers must be separated by a space.

```
e.g.   ?add 3 5
       8
```

Recursive Procedures

It is sometimes required to use a procedure to operate on a list of inputs without having to run the procedure

several times. This may be achieved by using the procedure name within its own definition.

```
e.g.   ?to square :size
       >fd :size rt 90
       >square :size
       >end
       square defined
```

Each time the procedure is used the program re-executes the procedure again.

It is a common requirement to test a particular set of conditions to check for the end of a procedure and then exit without having to use the ESC key. This can be achieved with the **if** and the **stop** primitives.

```
e.g.   ?to spiral :size
       >fd :size rt 90
       >if :size  90 [stop] spiral :size + 3
       >end
```

This program draws a square spiral using a call to itself and stops when the size is greater than 90 units. Note the use of square brackets to indicate the instructions that are executed if the condition is true.

```
e.g.   ?spiral 20
```

Lists

Dr. Logo has a set of commands for dealing with a list of characters. These include the primitives **list**, **first**, **last**, **bf** (but first), **bl** (but last) and **emptyp** (empty predicate). These are explained in the following example.

The list primitive indicates that a list of items is expected.
The primitive 'first' returns the first item in a list and the primitive 'last' returns the last item in a list.
The bf primitive returns the list of items less the first.
The bl primitive returns the list of items less the last.
The emptyp primitive checks the input that follows and returns TRUE if empty and FALSE if an item exists.

```
e.g.   ?to numbers :list
       >if emptyp :list  stop
       >pr first :list
       >number bf :list
       >end
```

Note: The Dr. Logo Tutorial and Language Reference Manuals, referred to earlier, often uses the word predicate. This means logical, as in 'predicate expression' meaning 'logical expression'. This is reflected in the primitive by the suffix 'p'. e.g. as in emptyp, above (empty predicate - outputs a logical expression TRUE/FALSE according to the nature of the input object).

Word Lists and Properties

Dr. Logo can be instructed to print out a sequence of words using pr and quotation marks (").

```
e.g.   ?pr "good "morning"
       good
       morning
```


A sequence of words can be enclosed by square brackets rather than using the quote.

```
e.g.   ?pr [good morning]
       good morning
```

The list primitive can be used to take two inputs and create a list.

```
e.g.   ?list "one "two
       [one two]
```

To create a list from one or more than two inputs, use brackets around the whole expression.

```
e.g.   ?(list "one "two "buckle "my "shoe)
       [one two buckle my shoe]
```

It is often required to link a name to another name or set of information e.g. A persons name and telephone number. The primitives **make** and **thing** are used in this context.

make uses two inputs; a name and a definition.
thing (or the symbol :) uses the name defined in the make primitive.

```
e.g.   ?make "mark "573467
       ?thing "mark
       573467
```

```
e.g.   ?make "team  mark david nigel julian
       ?:team
       [mark david nigel julian]
```

The primitive **pons** (print out names) can be used to display all the names defined so far.

e.g. ?pons
mark is 573467
team is [mark david nigel julian]

The make primitive allows one item to be attached to a name. Another set of primitives allow a larger number of items to be defined using a 'property list' technique. Each property that is attached to an object has a value.

e.g. Property List for Telephone Numbers

Object	Properties			
	name	code	number	
Smith	Fred	0274	36128	value
Jones	Ken	0512	53266	

The primitives **pprop** (put property), **gprop** (get property value) and **plist** (print property list) allows these lists to be generated and listed.

pprop needs three inputs: an object, a property name and value
gprop needs two inputs: an object, and a property name
plist needs one input: an object

e.g. ?pprop "Smith "Name "Fred
?pprop "Smith "Code "0274
?pprop "Smith "Number "36128
?pprop "Jones "Name "Ken
?pprop "Jones "Code "0512
?pprop "Jones "Number "53266

e.g. ?gprop "Smith "code
0274

e.g. ?plist "Jones
[Number 53266 Code 0512 Name Ken]

Lists and Sentences

Two primitives that allow the user to manipulate lists of words are **lput** (list put) and **fput** (first put). Both use two inputs: an object and a list, and add the object to the last and first items in the list.

e.g. ?fput "how [are you]
[how are you]
?lput "today [how are you]
[how are you today]

These primitives can be used in conjunction with the bf and bl primitives to modify the contents of a list of words.

e.g. ?fput "where bf [how are you today]
[where are you today]

In this example Dr. Logo takes the list 'how are you today' and removes the first word (but first primitive). The lput then places the word 'where' in the first position to give 'where are you today'.

Other useful primitives used to manipulate lists are **se** (sentence) and **shuffle.**

se uses two lists as an input and combines them to one list.
shuffle takes one list as an input and randomly mixes the elements.

e.g. ?se [mary had] [a little lamb]
[mary had a little lamb]

?shuffle [one two three four]
[four two three one]

The **rl** (read list) primitive wait for the user to enter a line and then prints that line as a list.

e.g. ?to greeting
>pr [Please enter your name]
>pr se [Hello,] rl
>end
greeting defined

?greeting
Please enter your name
?Lisa
Hello, Lisa

Program De-bugging

One of the most useful ways of finding errors in a program, is to step through each instruction one at a time. The primitive **watch**, instructs Dr. Logo to display each line of a procedure and its results, and waits for the ENTER key to be pressed, before moving on to the next line.

e.g. ?watch
?square
[1] In square, repeat 4 [fd 50 rt 90]

The number in square brackets indicates the level of the procedure. A procedure called from another procedure would be at level 2 etc.

The watch command is cancelled by **nowatch.**

It can be useful to insert pr (print) primitive into procedures during debugging so that the values of objects can be monitored during execution. This technique can be used with or without the watch facility.

SECTION B

LIST OF PRIMITIVES

This section lists all the primitives that are available, in alphabetical order of description. The command abbreviation that is accepted by Dr. Logo is shown in bold type under the heading Syntax. Primitives that are marked with an asterisk (*) are only implemented on EINSTEIN version 2 of the language.

The commands relating to disc drives use the notation 'A' for the left hand drive 'B' for the right hand side.

The commands used to control the background (setbg) and foreground (setpc) colours, use the input numbers to represent the colours as in the following table.

0	Transparent	8	Medium Red
1	Black	9	Light Red
2	Medium Green	10	Dark Yellow
3	Light Green	11	Light Yellow
4	Dark Blue	12	Dark Green
5	Light Blue	13	Magenta
6	Dark Red	14	Grey
7	Cyan	15	White

AND

Syntax: **and** logical.exp (...)

Purpose: Outputs TRUE if the result of all input logical expressions are true.

Example:
```
?and (3<4) (7>4)
TRUE
?to exp :x
>if and (:x > 5) (:x < 9)
>      [pr "OK] [pr [not OK]]
>end
```

ARC TANGENT

Syntax: **arctan** n

Purpose: Outputs the arc tangent of the input number.

Example:
```
?arctan 2
63.4349488229219
```

ASCII

Syntax: **ascii** word

Purpose: Outputs the ASCII value of the first character in the input word.

Example:
```
?ascii "G
71
?ascii "g
103
```

BACK

Syntax: **bk** distance_n

Purpose: Moves the turtle 'n' number of pixels in the opposite direction of its heading.

Example: ?bk 50

BUT FIRST

Syntax: **bf** object

Purpose: Outputs all but the first element in the input object.

Example:
```
?bf "smiles
smiles
?bf [1 2 3]
[2 3]
```

BUT LAST

Syntax: **bl** object

Purpose: Outputs all but the last element in the input object.

Example:
```
?bl [1 2 3 4]
[1 2 3]
```


BUTTON PADDLE

Syntax: **buttonp** paddle_n

Purpose: Outputs TRUE if the button on the specified paddle (joystick) is down. Numbers in the range 0 to 2 identify the paddle buttons as follows:

0 identifies button on paddle 1
1 identifies button on paddle 2

BYE

Syntax: **bye**

Purpose: Exits the current session of Dr. Logo.

Example: ?bye

CATCH

Syntax: **catch** name instr_list

Purpose: Traps errors and special conditions that occur during the execution of the input instruction list.

Example:
```
>catch "error [+ [] []] pr [I am here]
I am here
?
```

CHANGE FILE

Syntax: **changef** new_fname old_fname

Purpose: Changes the name of a file in a disc directory.

Example: ?changef "myfile "yourfile

CHARACTER

Syntax: **char** n

Purpose: Outputs the character whose ASCII value is the input number.

Example: ?char 83
S

CLEAN

Syntax: **clean**

Purpose: Erases the graphic screen without affecting the turtle.

Example: ?fd 50
?clean

CLEAR SCREEN

Syntax: **cs**

Purpose: Erases the graphic screen and puts the turtle at [0 0] heading 0 (north) with the pen down.

Example: ?rt 90 fd 50
?cs

CLEAR TEXT

Syntax: **ct**

Purpose: Erases all text in the window that currently contains the cursor, then positions the cursor in the top left hand corner of the window.

Example: ?pots
?ct

CONTINUE

Syntax: **co** <object>

Purpose: Ends a pause caused by pause, a CTRL-B keystroke, or by system word ERRACT.

Example: ?co

COPYOFF

Syntax: **copyoff**

Purpose: Stops echoing text at the printer.

Example: ?copyoff

COPYON

Syntax: **copyon**

Purpose: Starts echoing text at the printer.

Example: ?copyon

COSINE

Syntax: **cos** degrees_n

Purpose: Outputs the cosine of 'n' (in degrees.)

Example:
```
?cos 60
0.5
```

COUNT

Syntax: **count** object

Purpose: Outputs the number of elements in the input object

Example:
```
?count "six
3
?count [0 1 2 3]
4
```

CURSOR

Syntax: **cursor**

Purpose: Outputs a co-ordinate list, [x y], that contains the column and line numbers of the cursor's position within the text window.

Example:
```
?cursor
[0 24]
```


DEFAULT DRIVE

Syntax: **defaultd**

Purpose: Outputs the name of the current default drive.

Example:
```
?defaultd
A:
```

DEFINE

Syntax: **define** procname defin_list

Purpose: Makes the input definition list the definition of the specified procedure name.

Example:
```
?define "say.hi [[] [pr [Hi!]]
?say.hi
Hi!
```

DIFFERENCE

Syntax: - a b

Purpose: Outputs the difference between 'a' and 'b'. This character is a prefix or infix primitive and delimiter.

Example:
```
?- 10 5
5
?10 - 5
5
```

DIRECTORY

Syntax: **dir** <fname>

Purpose: Outputs a list of Dr. Logo file names on the default or specified disc. It accepts an ambiguous file name.

Example:
```
?dir
[LOGOHELP STARTUP STARS]
?dir "b:
[DRLOGO AVERAGE TOOLS ADDRESSES]
?dir "b:a????????
[AVERAGE ADDRESSES]
?dir "??AR????
[STARTUP STARS]
```

DIRECTORY OF PICTURES*

Syntax: **dirpic** <fname>

Purpose: Outputs a list of picture file names from the default or specified disc. It accepts an ambiguous file name.

Example:
```
?dirpic
[FLOWER FACE SQUARE]
?dirpic "f?
[FLOWER FACE]
?dirpic "b:
[DESIGN STARS PENDEMO]
?dirpic "b:?E??????
[DESIGN PENDEMO]
```


DOT

Syntax: **dot** co-ord_list

Purpose: Plots a dot at the position specified by the input co-ordinate list in the current pen colour.

Example:
```
?dot [50 10]
```

DOT COLOUR

Syntax: **dotc** co-ord_list

Purpose: Outputs the colour number (0 to 3) of the dot at the specified co-ordinate list, or -1 if the location is not on the screen.

Example:
```
?dotc [50 10]
2
```

EDIT

Syntax: **ed** <name | name_list>

Purpose: Loads the specified procedure(s) and/or variable(s) into the screen editor's buffer and enters the screen editor.

Example:
```
?ed "box
```

EDIT ALL

Syntax: **edall**

Purpose: Loads all the procedures and variables in the workspace into the screen editor's buffer and enters the screen editor.

Example:
```
?edall
```

EDIT FILE

Syntax: **edf** fname

Purpose: Loads the specified disc file into the screen editor's buffer or creates a new file and enters the screen editor with an empty buffer.

Example:
```
?edf "startup
```

EMPTY PREDICATE

Syntax: **emptyp** object

Purpose: Outputs TRUE if the input object is an empty word or an empty list.

Example:
```
?emptyp "
TRUE
?emptyp []
TRUE
?emptyp [x]
FALSE
?make "x [ ]
?emptyp :x
TRUE
```


END

Syntax: **end**

Purpose: Indicates the end of a procedure definition. 'end' must stand alone at the beginning of the last line.

Example:
```
?to square
>repeat 4 [fd 50 rt 90]
>end
square defined
?
```

EQUAL

Syntax: = a b

Purpose: Outputs TRUE if 'a' and 'b' are equal. This character is a prefix or infix primitive and delimiter.

Example:
```
?= "LOGO " LOGO
TRUE
?1 = 2
FALSE
```

ERASE

Syntax: **er** procname | procname_list

Purpose: Erases the specified procedure(s) from the workspace.

Example:
```
?er "box
```

ERASE ALL

Syntax: **erall**

Purpose: Erases all the procedures and variables from the workspace.

Example:
```
?erall
```

ERASE FILE

Syntax: **erasefile** fname

Purpose: Erases the specified disc file. It accepts an ambiguous file name.

Example:
```
?dir
[LOGOHELP STARTUP STARS]
?dir "b:
[DRLOGO AVERAGE TOOLS ADDRESSES]
?erf "logohelp
?erf "??AR????
?erf "b:A?
```

ERASE NAME

Syntax: **ern** varname | varname_list

Purpose: Erases the specified variable(s) from the workspace.

Example:
```
?ern [side angle]
```


ERASE PICTURE FILE*

Syntax: **erpic** fname

Purpose: Erases the specified picture file(s). It accepts an ambiguous file name.

Example:
```
?dirpic
[FLOWER FACE SQUARE]
?dirpic "b:
[DESIGN STARS PENDEMO]
?erpic "f?
[FLOWER FACE] will be erased from the diskette in drive A.
Is this what you want (y/n)?n
?erpic "b:design
[DESIGN] will be erased from the diskette in drive B.
Is this what you want (y/n)?y
?erpic "b:?E??????
[DESIGN PENDEMO] will be erased from the diskette in drive B.
Is this what you want (y/n)?n
?
```

ERROR

Syntax: **error**

Purpose: Outputs a list whose elements describe the most recent error.

Example:
```
>catch "error [do.until.error]
>show error
```

FENCE

Syntax: **fence**

Purpose: Establishes a boundary that limits the turtle to on-screen plotting. The 'window' primitive is used to remove the boundary.

Example:
```
?fence
?fd 300
Turtle out of bounds
```

FILL*

Syntax: **fill**

Purpose: Paints an area with the current pen colour changing the dot under the turtle (and all horizontally and vertically contiguous dots of the same colour) to the current pen colour, using the current pen state.

Example:
```
?repeat 4 [fd 50 rt 90]
?rt 60 fd 10
?pu fd 4
?pd
?st
?fill
```


FIRST

Syntax: **first** object

Purpose: Outputs the first element of the input object.

Example:
```
?first "zebra
z
?first [1 2 3]
1
```

FIRST PUT

Syntax: **fput** object object

Purpose: Outputs a new object formed by making the first input object the first element in the second input object.

Example:
```
?fput "s "miles
smiles
?fput 1 [2 3]
[1 2 3]
```

FORWARD

Syntax: **fd** distance_n

Purpose: Moves the turtle 'n' number of pixels in the direction of its heading.

Example:
```
?fd 100
```

FULL SCREEN

Syntax: **fs**

Purpose: Selects a full graphic screen.

Example: ?ss
?fs

GO

Syntax: **go** word

Purpose: Executes the line within the current procedure following a label expression with the same input word.

Example: >go "loop

GET PROPERTY

Syntax: **gprop** name propname

Purpose: Outputs the property value of the input property name of the input-named object.

Example: ?make "height "72"
?gprop "height ".APV
72"

GREATER THAN

Syntax: > a b

Purpose: Outputs TRUE if 'a'is greater than 'b'. This character is a prefix or infix primitive and delimiter.

Example: ?> 19 20
FALSE
?20 > 19
TRUE

HOME

Syntax: **home**

Purpose: Returns the turtle to position 0 0 heading 0 (north).

Example: ?rt 120 fd 100 lt 60 fd 50
?home

HIDE TURTLE

Syntax: **ht**

Purpose: Makes the turtle invisible. Thus speeding up, and clarifying, drawing.

Example: ?ht
?fd 50
?st

IF

Syntax: **if** logical_exp inst_lst inst_lst

Purpose: Executes one of two instruction lists depending on the value of the input logical expression. Input instructions must be literal lists enclosed in brackets.

Example:
```
>if (a > b) [pr [a is bigger]]
>           [pr [b is bigger]]
```

INTEGER

Syntax: **int** n

Purpose: Outputs the integer portion of 'n'.

Example:
```
?int 4 / 3
1
```

ITEM

Syntax: **item** n object

Purpose: Outputs the specified element of the input object.

Example:
```
?item 4 "dwarf
r
```


KEYBOARD PREDICATE

Syntax: **keyp**

Purpose: Outputs TRUE if a character has been typed at the keyboard and is waiting to be read.

Example:
```
?to inkey
; outputs a keystroke if one is ready
>if keyp [op rc] [op " ]
>end
```

LABEL

Syntax: **label** word

Purpose: Identifies the line to be executed after a 'go' expression with the input word.

Example:
```
>label "loop
```

LAST

Syntax: **last** object

Purpose: Outputs the last element of the input object.

Example:
```
?last [0 2 4]
4
```

LAST PUT

Syntax: **lput** object object

Purpose: Outputs a new object formed by making the first input object the last element in the second input object.

Example:
```
?lput 4 [1 2 3]
[1 2 3 4]
```

LESS THAN

Syntax: < a b

Purpose: Outputs TRUE if 'a' is less than 'b'. This character is a prefix or infix primitive and delimiter.

Example:
```
?< 27 13
FALSE
?13 < 27
TRUE
```

LEFT

Syntax: **lt** degrees_n

Purpose: Rotates the turtle 'n' degrees to the left.

Example:
```
?lt 90
```


LIST

Syntax: **list** object (...)

Purpose: Outputs a list made up of the input objects, retains lists' outer brackets (compare with se).

Example:
```
?(list 1 2 3 4)
[1 2 3 4]
?list "big [feet]
[big [feet]]
?(list)
[]
```

LIST PREDICATE

Syntax: **listp** object

Purpose: Outputs TRUE if the input object is a list.

Example:
```
?listp "word
FALSE
?listp [1 3 5 7]
TRUE
```

LOWER CASE

Syntax: **lc** word

Purpose: Outputs the input word with all alphabetic characters in lower case.

Example:
```
?lc "SHEILA
sheila
```

LOAD

Syntax: **load** fname

Purpose: Reads the named file from the disc into the workspace.

Example:
```
?load "myfile
?load "b:shapes
```

LOAD PICTURE FILE*

Syntax: **loadpic** fname

Purpose: Paints the graphic design saved in the named picture file onto the graphics screen.

Example:
```
?loadpic "square
?loadpic "b:pendemo
```

LOCAL

Syntax: **local** varname (...)

Purpose: Makes the named variable(s) accessible only to the current procedure and the procedures it calls.

Example:
```
>(local "x "y "z)
```


MAKE

Syntax: **make** varname object

Purpose: Makes the named variable the value of the input object.

Example:
```
?make "side 50
?:side
50
```

MEMBER PREDICATE

Syntax: **memberp** object object

Purpose: Outputs TRUE if the first input object is an element of the second input object.

Example:
```
?memberp "y "only
TRUE
?memberp " "
FALSE
```

NAME PREDICATE

Syntax: **namep** word

Purpose: Outputs TRUE if the input word identifies a defined variable name.

Example:
```
?make "apple "red
?namep "apple
TRUE
```

NODES

Syntax: **nodes**

Purpose: Outputs the number of free nodes in the workspace (1 node = 5 bytes).

Example:
```
?nodes
34363
```

NO FORMAT

Syntax: **noformat**

Purpose: Removes procedure formatting, including comments, from the workspace.

Example:
```
?noformat
?recycle nodes
39487
```

NOT

Syntax: **not** logical_exp

Purpose: Outputs TRUE if the input logical expression outputs FALSE; FALSE if the input logical expression outputs TRUE.

Example:
```
?not (3 = 4)
TRUE
?not (3 = 3)
FALSE
```


NO TRACE

Syntax: **notrace**

Purpose: Turns off trace monitoring of procedure execution.

Example:
```
?trace
?square
[1] Evaluating square
?notrace
?square
```

NO WATCH

Syntax: **nowatch** procname | procname_list

Purpose: Turns off watch monitoring of all or specified procedures.

Example:
```
?watch
?square
[1] In square, repeat 4 [fd 50 rt 90]
?nowatch
?square
```

NUMBER PREDICATE

Syntax: **numberp** object

Purpose: Outputs TRUE if the input object is a number.

Example:
```
?numberp "two
FALSE
```

OR

Syntax: **or** logical_exp (...)

Purpose: Outputs FALSE if all input logical expressions output FALSE.

Example:
```
?or "TRUE "TRUE
TRUE
?or (3 = 4) (1 = 1)
TRUE
```

OUTPUT

Syntax: **op** object

Purpose: Makes the input object the output of the procedure and exits the procedure at that point.

Example:
```
?to mult :a :b
>op :a * :b
>end
```

PADDLE

Syntax: **paddle**

Purpose: Outputs a number that represents a paddle (joystick) input co-ordinate; numbers in the range 0 to 3 identify a paddle's position as follows:

0 outputs x co-ordinate of paddle 1
1 outputs y co-ordinate of paddle 1
2 outputs x co-ordinate of paddle 2
3 outputs y co-ordinate of paddle 2

PAUSE

Syntax: **pause**

Purpose: Suspends the execution of the current procedure to allow interaction with the interpreter or editor.

Example:
```
>if :size > [pause]
```

PEN DOWN

Syntax: **pd**

Purpose: Puts the turtle's pen down. The turtle resumes drawing.

Example:
```
?fd 20 pu fd 20
?pd
?fd 20
```

PEN ERASE

Syntax: **pe**

Purpose: Changes the turtle's pen colour to 0. The turtle erases drawn lines.

Example:
```
?fd 50
?pe
?bk 50
?fd 50
?pd fd 50
```

PEN UP

Syntax: **pu**

Purpose: Lifts the turtle's pen up. The turtle stops drawing.

Example:
```
?fd 30
?pu
?fd 30
?pd fd 30
```

PIECE

Syntax: **piece** n n object

Purpose: Outputs an object that contains the specified elements of the input object.

Example:
```
?piece 3 6 "industry
dust
?piece 2 4 [a b c d e]
[b c d]
```

PRINT

Syntax: **pr** object (...)

Purpose: Displays the input object(s) on the text screen, removes lists' outer brackets, follows last input with a carriage return (compare with show and type).

Example:
```
?pr [a b c]
a b c
```


PRINT OUT

Syntax: **po** name | name_list

Purpose: Displays the definition(s) of the specified procedure(s) or variable(s).

Example:
```
?po "square
to square
repeat 4 [fd 4 rt 90]
end
?po "x
x is 10
```

PRINT OUT ALL

Syntax: **poall**

Purpose: Displays the definitions of all procedures and variables in the workspace.

Example: ?poall

PRINT OUT NAMES

Syntax: **pons**

Purpose: Displays the names and values of all variables in the workspace.

Example: ?pons

PRINT OUT PROCEDURES

Syntax: **pops**

Purpose: Displays the names and definitions of all procedures in the workspace.

Example: ?pops

PRINT OUT TITLES

Syntax: **pots**

Purpose: Displays the names and inputs of all procedures in the workspace.

Example: ?pots

PRODUCT

Syntax: * a b (...)

Purpose: Outputs the product of 'a' and 'b'. This character is a prefix or infix primitive and delimiter.

Example:
```
?* 4 6
24
?4 * 6
24
```


PROPERTY LIST

Syntax: **plist** name

Purpose: Outputs the property list of the named object.

Example:
```
?plist "square
[.DEF[[] [repeat 4 [fd 4 rt 90]]]]
```

PROPERTY PAIRS

Syntax: **pps**

Purpose: Displays the non-system property pairs of all objects in the workspace.

Example:
```
?pprop "Kathy "extension 82
?pps
Kathy's extension is 82
```

PUT PROPERTY

Syntax: **pprop** name propname propval

Purpose: Puts the input property pair into the named object's property list.

Example:
```
?pprop "master ".APV "Scott
?:master
Scott
```

QUOTIENT

Syntax: / a b

Purpose: Outputs the decimal quotient of 'a' and 'b'. This character is a prefix or infix primitive and delimiter.

Example:
```
?/ 25 5
5
?25/5
5
```

QUOTIENT

Syntax: **quotient** n1 n2

Purpose: Outputs the integer quotient of 'n1' and 'n2'. 'n1' and 'n2' are truncated to integers before dividing.

Example:
```
?5 / 2
2.5
?quotient 5 2
2
```

RANDOM

Syntax: **random** n

Purpose: Outputs a random non-negative integer less than 'n'.

Example:
```
?random 20
13
```


READ CHARACTER

Syntax: **rc**

Purpose: Outputs the first character typed at the keyboard.

Example:
```
?make "key rc
X
?:key
X
```

READ LIST

Syntax: **rl**

Purpose: Outputs a list that contains a line typed at the keyboard. The input must be followed by a carriage return.

Example:
```
?make "instr_list rl
repeat 4 [fd 50 rt 90]
?:instr_list
[repeat 4 [fd 50 rt 90]]
```

READ QUOTE

Syntax: **rq**

Purpose: Outputs a word that contains a line typed at the keyboard. The input must be followed by a carriage return.

Example:
```
?make "command rq
repeat 3 [fd 60 rt 120]
?:command
repeat 3 [fd 60 rt 120]
```

RECYCLE

Syntax: **recycle**

Purpose: Frees as many nodes as possible and reorganizes the workspace.

Example: ?recycle

REMAINDER

Syntax: **remainder** n1 n2

Purpose: Outputs the integer remainder obtained when 'n1' is divided by 'n2'.

Example: ?remainder 8 5
3

REMOVE PROPERTY

Syntax: **remprop** name propname

Purpose: Removes the specified property from the named object's property list.

Example: ?remprop "master ".APV
?remprop "Kathy "extension

REPEAT

Syntax: **repeat** n instr_list

Purpose: Executes the input instruction list the input number of times.

Example: ?repeat 4 [fd 50 rt 90]

REPRODUCE RANDOM

Syntax: **rerandom**

Purpose: Makes a subsequent random expression reproduce the same random sequence.

Example: ?random 20
13
?rerandom
?random 20
13

RIGHT

Syntax: **rt** degrees_n

Purpose: Rotates the turtle 'n' degrees to the right.

Example: ?rt 90

ROUND

Syntax: **round** n

Purpose: Outputs 'n' rounded to the nearest integer.

Example: ?round 3/2
2

RUN

Syntax: **run** instr_list

Purpose: Execute the input instruction list.

Example: ?make "instr_list [fd 40 rt 90]
?run :instr_list

SAVE

Syntax: **save** fname

Purpose: Writes the contents of the workspace to the named disc file.

Example: ?save "shapes

SAVE PICTURE FILE*

Syntax: **savepic** fname

Purpose: Writes the contents of the graphic screen to the named picture file.

Example: ?savepic "circle
?savepic "b:house

SCREEN FACTS

Syntax: **sf**

Purpose: Outputs information about the graphic screen in the form bgcolour screen-state split-size window state scrunch where bgcolour is the background colour number of the graphic screen; screen-state indicates splitscreen, textscreen or fullscreen; split-size is the number of text lines displayed on the splitscreen's text window; and window state indicates window, wrap, or fence mode. Scrunch indicates the current aspect ratio of the graphic screen.

Example: ?setbg 2
?ss
?setsplit 25
?wrap
?setscrunch 2
?sf
[2 SS 25 WRAP 2]

SENTENCE

Syntax: **se** object (...)

Purpose: Outputs a list made up of the input objects, removes lists' outer brackets (compare with list)

Example: ?make "instr_list rl
repeat 4 [fd 50 rt 90]
?run (se "cs :instr_list "ht

SET BACKGROUND

Syntax: **setbg** n

Purpose: Sets the graphic screen background to the colour represented by 'n'.

Example: ?setbg 1

SET CURSOR

Syntax: **setcursor** co-ord_list

Purpose: Positions the cursor at the location specified by the input text screen co-ordinate list.

Example: ?setcursor [30 15] pr "Hi!

SET DISC

Syntax: **setd** d:

Purpose: Makes the specified drive the default drive.

Example: ?setd "b:

SET HEADING

Syntax: **seth** degrees_n

Purpose: Turns the turtle to the absolute heading specified by 'n' (in degrees). Positive numbers turn the turtle clockwise; negative numbers turn the turtle anticlockwise.

Example: ?seth 90

SET PEN COLOUR

Syntax: **setpc** n

Purpose: Sets the turtle's pen to the colour specified by 'n'.

SET POSITION

Syntax: **setpos** co-ord_list

Purpose: Moves the turtle to the position specified in the input co-ordinate list.

Example: ?setpos [30 20]

SET SCRUNCH

Syntax: **setscrunch** n

Purpose: Sets the graphic screen's aspect ratio to the input number.

Example: ?setscrunch 1

SET SPLIT

Syntax: **setsplit** n

Purpose: Sets the number of lines in the splitscreen's text window. The input number must be in the range 1 to 25.

Example: ?setsplit 25

SET X

Syntax: **setx** n

Purpose: Moves the turtle horizontally to the x co-ordinate specified by the input number.

Example: ?setx 150
?setx -150

SET Y

Syntax: **sety** n

Purpose: Moves the turtle vertically to the y co-ordinate specified by the input number.

Example: ?sety 100
?sety -99

SHOW

Syntax: **show** object

Purpose: Displays the input object on the text screen. Retains lists' outer brackets. Follows the input with a carriage return (compare 'show' with 'pr' and 'type').

Example: ?show [a b c]
[a b c]

SHOW TURTLE

Syntax: **st**

Purpose: Makes the turtle visible.

Example: ?ht
?fd 50
?st

SHUFFLE

Syntax: **shuffle** list

Purpose: Outputs a list that contains the elements of the input list in random order.

Example: ?shuffle [1 2 3 4]
[3 2 4 1]

SIN

Syntax: **sin** degrees_n

Purpose: Outputs the sine of the input number of degrees.

Example:
```
?sin 30
.5
```

SPLIT SCREEN

Syntax: **ss**

Purpose: Displays a window of text on the graphic screen.

Example:
```
?ss
?fs
```

STOP

Syntax: **stop**

Purpose: Stops the execution of the current procedure, and returns to the top level (the ? prompt) of the calling procedure.

Example:
```
>if remainder :x :y > 5 [stop]
>run :instr_list
```


SUM

Syntax: + a b (...)

Purpose: Outputs the sum of 'a' and 'b'. This character is a prefix or infix primitive and delimiter.

Example:
```
?+ 2 2
4
?2 + 2
4
```

TEXT

Syntax: **text** procname

Purpose: Outputs the definition list of the specified procedure.

Example:
```
?text "square
[[] [repeat 4 [fd 50 rt 90]]]
```

TEXT SCREEN

Syntax: **ts**

Purpose: Selects a full text screen.

Example:
```
?ts
?fs
```

THING

Syntax: **thing** varname

Purpose: Outputs the value of the named variable.

Example:
```
?make "varname "gadget
?thing "varname
gadget
```

THROW

Syntax: **throw** name

Purpose: Executes the line identified by the input name in a previous 'catch' expression.

Example:
```
>if remainder :x :y > 5
>   [throw "toobig]
>run :instr_list
```

TO

Syntax: **to** procname inputs

Purpose: Indicates the beginning of a procedure definition.

Example:
```
?to square
>repeat 4 [fd 50 rt 90]
>end
square defined
?
```


TONES

Syntax: **tones** n_freq n_dura

Purpose: Outputs a note of the frequency and duration specified in the input note_list. The range of n_freq is 32 to 10000 Hz. The range of n_dura is 0 to 65535 in increments of 1/60 second.

Example: ?tones 450 10

TOWARDS

Syntax: **towards** co-ord_list

Purpose: Outputs a heading that would make the turtle face the position specified in the input co-ordinate list.

Example:
```
?fd 50 rt 90
?towards [0 0]
180
```

TRACE

Syntax: **trace**

Purpose: Turns on trace monitoring of procedure execution. Displays the name of each procedure as it is called and the name and value of each variable as it is defined. It allows observation of the procedure's execution without interruption.

Example:
```
?trace
?square
[1] Evaluating square
```

TURTLE FACTS

Syntax: tf

Purpose: Outputs information about the turtle in the form: xcor ycor heading pencolour penstate shownp

xcor is the turtle's x co-ordinate.
ycor is the turtle's y co-ordinate.
heading indicates the compass direction the turtle is facing.
shownp is TRUE if the turtle is visible.
penstate indicates PENDOWN (PD), PENERASE (PE), PENREVERSE (PX), or PENUP (PU).
pencolour identifies the pen's colour number.

Example:
```
?setpos [15 30]
?rt 60
?setpc 3
?pe
?ht
?tf
[15 30 60 3 PE FALSE]
```

TYPE

Syntax: **type** object (...)

Purpose: Displays the input object(s) on the text screen. Removes lists' outer brackets. Does not follow the last input with a carriage return (compare 'type' with 'pr' and 'show').

Example:
```
?type [a b c]
a b c
```


UPPER CASE

Syntax: **uc** word

Purpose: Outputs the input word with all alphabetic characters in upper case.

Example:
```
?uc "sheila
SHEILA
```

WAIT

Syntax: **wait** n

Purpose: Stops procedure execution for the amount of time specified by 'n'. The amount of time = input number * 1/4 seconds.

Example: ?wait 50 pr "Hi!

WATCH

Syntax: **watch** procname | procname_list

Purpose: Turns on the expression-by-expression procedure execution monitor that pauses before the execution of each statement, to allow interaction with interpreter or editor.

Example:
```
?watch
?square
[1] In square, repeat 4 [fd 50 rt 90]
?nowatch
?square
```

WHERE

Syntax: **where**

Purpose: Outputs the item number of the most recent successful 'memberp' expression.

Example:
```
?memberp "r [q r s]
TRUE
?where
2
```

WINDOW

Syntax: **window**

Purpose: Allows the turtle to plot outside the visible graphic screen after a wrap or fence expression.

Example:
```
?fence fd 300
Turtle out of bounds
?window
?fd 300
```

WORD

Syntax: **word** word (...)

Purpose: Outputs a word made up of the input words.

Example:
```
?word "sun "shine
sunshine
```


WORD PREDICATE

Syntax: **wordp** object

Purpose: Outputs TRUE if the input object is a word or a number.

Example:
```
?wordp "hello
TRUE
?wordp []
FALSE
```

WRAP

Syntax: **wrap**

Purpose: Makes the turtle reappear on the opposite side of the graphic screen when it exceeds the boundary.

Example:
```
?wrap
?rt 10 fd 3000
?window
?rt 10 fd 3000
?
```

SEMI COLON

Syntax: ;

Purpose: A delimiter which indicates comments to be ignored by the interpreter.

Example:
```
?to triangle
>;an equalateral triangle
>repeat 3 [fd 50 rt 120]
>end
triangle defined
?
```

OPEN BRACKETS

Syntax: (

Purpose: A delimiter which begins an enclosed expression that contains multiple inputs or groups of numeric expressions, and specifies the order of operations.

Example:
```
?5 / 4 + 1
2.25
?5 / (4 + 1)
1
```

CLOSE BRACKETS

Syntax:)

Purpose: A delimiter which ends an enclosed expression that contains multiple inputs or groups of numeric expressions and specifies the order of operations.

Example:
```
?run (se "cs :instr_list "ht)
```


QUOTE MARKS

Syntax: "

Purpose: Forces Dr. Logo to interpret a word as an object instead of a procedure name.

Example:
```
?to say.hello :user
>pr "Hello :user
>end
say.hello defined
?say.hello "Pauline
Hello Pauline
?
```

SQUARE BRACKETS

Syntax: [ ]

Purpose: Delimiters which enclose elements of a list.

Example:
```
?make "groc.list [apples peaches peas]
?:groc.list
[apples peaches peas]
```

OBLIQUE

Syntax: \

Purpose: Forces Dr. Logo to interpret a special character as a literal character.

Example:
```
?pr "\[
[
```

APPENDIX A

DR. LOGO CONTROL CODE COMMANDS

Dr. Logo uses the following control character commands to control screen display and cursor movement. To enter a control character command, hold down the control key and press the indicated letter key.

BEG_LINE - CTRL-A

Moves the cursor to the beginning of the line.

BS - CTRL-H

Moves the cursor back one space; that is, it moves the cursor one position to the left.

END_ED - CTRL-C

Ends the editing session; exits the screen editor, updates Dr. Logo's workspace with the definitions of all the procedures and variables in the screen editor's buffer. This code is valid only in the screen editor.

DEL_CHAR - CTRL-F

Deletes the character at the current cursor position.

END_LINE - CTRL-E

Moves the cursor to the end of the current line.

FS - CTRL-D

Moves the cursor one space to the right.

STOP - CTRL- ←

Outside the screen editor, it immediately terminates the currently executing procedure (an immediate throw "TOPLEVEL). Inside the screen editor, it exits the screen editor without updating Dr. Logo's workspace, discarding any changes made during the screen editing session; retrieve abandoned edit buffer with an ed command.

BS_DEL - CTRL-Y

Deletes the character to the left of the cursor and moves the cursor back one space.

TAB - CTRL-I

Moves the cursor to the next tab setting (column 5, 9, 13...) inserting up to 4 spaces in the current line.

INS_LF - CTRL-N

Inserts one line feed.

DEL_EOL - CTRL-O

Deletes all characters to the right of the cursor up to 134 characters towards the end of the line. Deleted characters are stored in a buffer and can be restored with a RECALL (CTRL-R).

CNTRL_LINE - CTRL-L

Inside screen editor, it readjusts the display so that the line currently indicated by the cursor is positioned midway down the screen. If the cursor is less than 12 lines from the beginning of the buffer, the screen editor simply re-displays the screen when CTRL-L is pressed. Outside screen editor, displays a full graphic screen, devoting the monitor to graphics.

CR - CTRL-M

Generates a carriage return; enters information into the computer.

DN_LINE - CTRL-J

Moves the cursor to the next line; in the screen editor, the cursor moves down one line towards the end of the buffer.

OPEN_LINE - CTRL-N

Opens a new line in the screen editor; it is equivalent to pressing CR (Carriage Return or ENTER) followed by BS (Back Space or CTRL-H).

UP_LINE - CTRL-K

Moves the cursor up to the previous line; the cursor moves up one line towards the beginning of the buffer.

INS_BK - CTRL-Q

Generates a backslash, \ , and makes Dr. Logo treat a delimiter character as a literal character. Delimiter characters are: [ ] ( ) " : ; = < > + - * /

TOP - CTRL-↑

Positions the cursor at the top or beginning of the workspace.

SPLIT - CTRL-S

Displays a 'split screen'; opens a text window on the graphic screen.

TEXT - CTRL-T

Displays a full text screen, devoting the monitor to text.

PGUP - CTRL-U

Displays the previous page of text in the screen editor's buffer, the previous 24 lines towards the beginning of the buffer.

PGDN - CTRL-V

Displays the next page of text in the screen editor's buffer, the next 24 lines towards the bottom of the buffer. This code is valid only in the screen editor.

WAIT - CTRL-W

Interrupts the scrolling of a text display, waits until the next keystroke to continue scrolling the display.

BOT - CTRL-X

Positions the cursor at the bottom or end of the file.

RECALL - CTRL-R

Transfers text from the buffer; i.e. it redisplays the line most recently stored in the buffer by a CR (Carriage Return or Enter) or a DEL_EOL (Delete to End of Line or CTRL-K) keystroke.

PAUSE - CTRL-B

Interrupts the currently executing procedure, displays a pause prompt to allow interactive procedure debugging. Enter **co** to continue the execution of the interrupted procedure. Enter **throw** "TOPLEVEL to exit to the outer most level. Enter **stop** to exit to the prior level.

APPENDIX B

DR. LOGO SYSTEM PRIMITIVES

A Dr. Logo System Primitive is preceded by a full stop, and allows the user to control auxiliary hardware peripherals or look at memory locations.

Note: These system primitives should be used only by programmers familiar with assembly language.

Primitive	Input	Definition and Example
.contents		Displays the contents of the Dr. Logo symbol space.
.deposit	n1 n2	Puts n2 into the absolute memory location in the current data segment specified by n1.
.examine	n	Displays the contents of the absolute memory location in the current data segment specified by n (byte value).
.in	port_n	Outputs the current value of port n (0 to 65535).
.out	port_n1 n2	Assigns the value n2 to port_n1 (0 to 65535).

APPENDIX C

DR. LOGO SYSTEM WORDS

Words	Definition
ERRACT	When TRUE causes a pause when an error occurs, then returns to TOPLEVEL.
FALSE	System value.
PD	PENDOWN value of turtle's pen state.
PE	PENERASE value of turtle's pen state.
PX	PENREVERSE value of turtle's pen state.
PU	PENUP value of turtle's pen state.
REDEFP	When TRUE allows redefinition of primitives.
TOPLEVEL	A throw "TOPLEVEL will exit all pending procedures.
TRUE	System value.

APPENDIX D

DR. LOGO SYSTEM PROPERTIES

The following list shows the property pairs Dr. Logo assigns to global variables, procedures, and other objects in the workspace. A property pair is a property name and its property value.

Property Name	Property Value
.APV	Associated Property Value; the value of a global variable.
.DEF	Definition of a procedure.
.ENL	End of a procedure line that is broken by a carriage return and spaces.
.FMT	Beginning of a procedure line that is broken by a carriage return and spaces.
.PRM	Memory location of a primitive.
.REM	Remark or comments that follows a semicolon.
.SPC	Number of spaces that follow a carriage return in a broken procedure line.

APPENDIX E

DR. LOGO ERROR MESSAGES

Number	Message
2	Number too big.
6	(symbol) is a primitive.
7	Can't find label (symbol).
8	Can't (symbol) from the editor.
11	I'm having trouble with the disc.
12	Disc is full.
13	Can't divide by zero.
15	File already exists.
17	File not found.
21	Can't find catch for (symbol).
23	I'm out of space.
25	(symbol) is not true or false.
29	Not enough inputs to (procedure).
32	Too few items in (list).
34	Turtle out of bounds.
35	I don't know how to (symbol).
36	(symbol) has no value.
37	)without(
38	I don't know what to do with (symbol).
40	Disc is write protected.
41	(procedure) doesn't like (symbol) as input.
42	(procedure) didn't output.
44	!!! Dr. Logo system bug !!! (should not occur. Please write to Digital Research if it does.)
45	The word is too long.
46	I don't have enough buffer space.
47	If wants [ ]'s around instruction list.

48	Varies according to disc error: o I can't find that drive o Drives can only be 1 or 2 sided o you need at least 2 drives to do this. o Source and Destination diskettes differ.
49	(symbol) isn't a parameter.
50	I can't (symbol) while loading.
51	The file is write proteced.
52	I can't find the disc drive.

INDEX

PREFIX OR INFIX SYMBOLS

SPECIAL CHARACTERS