

TURTLE TALK

Acknowledgements

SmartLOGO Turtle Talk

author:	Seymour Papert
contributing editors:	John Berlow Eric Brown Wynter Snow
word processing:	Marie Barbeau

SmartLOGO Reference Manual

contributing editors:	Eric Brown Barbara Mingie
word processing:	Marie Barbeau

Graphic design and layout

Lorraine Lavigne
Richard Lavigne
Julien Perron

Exploring SmartLOGO

author/programmer:	Eric Brown
contributing editor:	Barbara Mingie

SmartLOGO Software team

analysts:	Mario Bergeron Mario Bourgoin Mamadou Billo Diallo
testers:	Hani Fanous René Yelle

Project Management:	Michael Quinn
----------------------------	---------------

Contributions to Logo software and reference materials have been made by many members of Logo Computer Systems Inc. Many members of Coleco Industries Inc. have contributed to the preparation of the software and reference materials.

© 1984 Logo Computer Systems Inc.

All rights reserved. No part of the documentation contained herein may be reproduced, stored in retrieval systems, or transmitted, in any form or by any means, photocopying, electronic, mechanical, recording, or otherwise, without the prior approval in writing from Logo Computer Systems Inc.

Table of Contents

Chapter 1

1	Introduction: Welcome to SmartLOGO
---	---

Chapter 2

5	Starting Out
---	---------------------

5	Meet the Turtle
10	Some Fancy Stuff

Chapter 3

13	Motion, Shapes, and Colors
----	-----------------------------------

13	Let's Get Moving
14	Shapes and Colors
15	More Turtles
16	A Project
18	Making Your Own Shapes

Chapter 4

23 **Procedures**

- 23 Teaching the Computer a New Command
- 25 Changing the Procedure
- 26 Procedures with Inputs
- 29 Circles and "Magic" Numbers
- 31 Subprocedures
- 34 Printing on Paper

Chapter 5

37 **Reporters**

- 39 More Reporters

Chapter 6

43 **SmartLOGO as a Calculator**

Chapter 7

47 **Making a Movie**

- 49 Sound Effects

Chapter 8

51 **Naming Things**

- 52 Names for Turtles and Names for Shapes
- 53 Teams
- 54 Names of Things

Chapter 9

59 **Memory**

- 59 Saving Procedures
- 60 Saving Pictures
- 61 Saving New Turtle Shapes

Chapter 10

65 **P.S. Some Things to Experiment With**

90-DAY LIMITED WARRANTY

Coleco warrants to the original consumer purchaser in the United States of America that the physical components of this digital data pack (the "Digital Data Pack") will be free of defects in material and workmanship for 90 days from the date of purchase under normal in-house use.

Coleco's sole and exclusive liability for defects in material and workmanship of the Digital Data Pack shall be limited to repair or replacement at an authorized Coleco Service Center. This warranty does not obligate Coleco to bear the cost of transportation charges in connection with the repair or replacement of defective parts.

This warranty is invalid if the damage or defect is caused by accident, act of God, consumer abuse, unauthorized alteration or repair, vandalism or misuse.

ANY IMPLIED WARRANTIES ARISING OUT OF THE SALE OF THE DIGITAL DATA PACK INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE LIMITED TO THE ABOVE 90 DAY PERIOD. IN NO EVENT SHALL COLECO BE LIABLE TO ANYONE FOR INCIDENTAL, CONSEQUENTIAL, CONTINGENT OR ANY OTHER DAMAGES IN CONNECTION WITH OR ARISING OUT OF THE PURCHASE OR USE OF THE DIGITAL DATA PACK. MOREOVER, COLECO SHALL NOT BE LIABLE FOR ANY CLAIM OF ANY KIND WHATSOEVER BY ANY OTHER PARTY AGAINST THE USER OF THE DIGITAL DATA PACK.

This limited warranty does not extend to the programs contained in the Digital Data Pack and the accompanying documentation (the "Programs"). Coleco does not warrant the Programs will be free from error or will meet the specific requirements or expectations of the consumer. The consumer assumes complete responsibility for any decisions made or actions taken based upon information obtained using the Programs. Any statements made concerning the utility of the Programs are not to be construed as express or implied warranties.

COLECO MAKES NO WARRANTY, EITHER EXPRESS OR IMPLIED, INCLUDING ANY IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, IN CONNECTION WITH THE PROGRAMS, AND ALL PROGRAMS ARE MADE AVAILABLE SOLELY ON AN "AS IS" BASIS.

IN NO EVENT SHALL COLECO BE LIABLE TO ANYONE FOR INCIDENTAL, CONSEQUENTIAL, CONTINGENT OR ANY OTHER DAMAGES IN CONNECTION WITH OR ARISING OUT OF THE PURCHASE OR USE OF THE PROGRAMS AND THE SOLE AND EXCLUSIVE LIABILITY, IF ANY, OF COLECO, REGARDLESS OF THE FORM OF ACTION, SHALL NOT EXCEED THE PURCHASE PRICE OF THE DIGITAL DATA PACK. MOREOVER, COLECO SHALL NOT BE LIABLE FOR ANY CLAIM OF ANY KIND WHATSOEVER BY ANY OTHER PARTY AGAINST THE USER OF THE PROGRAMS.

This warranty gives you specific legal rights, and you may have other rights which vary from State to State. Some states do not allow the exclusion or limitation of incidental or consequential damages or limitations on how long an implied warranty lasts, so the above limitations or exclusions may not apply to you.

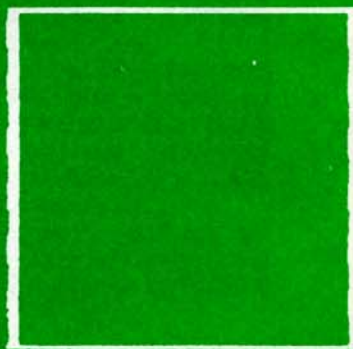
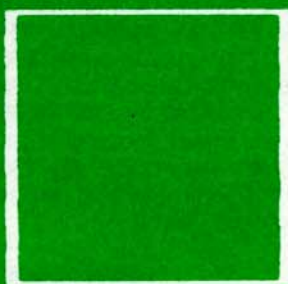
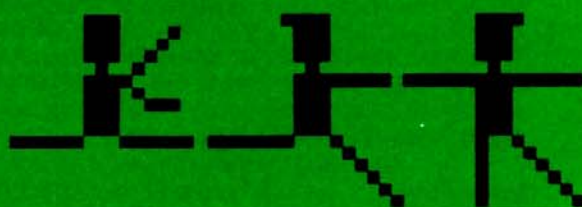
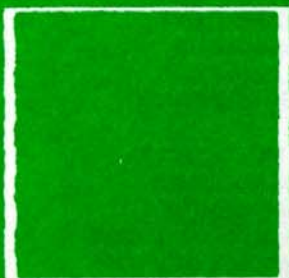
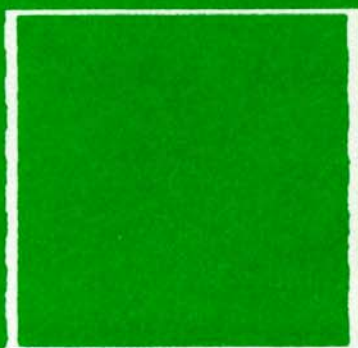
SERVICE POLICY

Please read your Owner's Manual carefully before using your Digital Data Pack. If your Digital Data Pack fails to operate properly, please refer to the trouble-shooting checklist in the Operating Tips Manual. If you cannot correct the malfunction after consulting this manual, please call Customer Service on Coleco's toll-free service hotline: 1-800-842-1225 nationwide. This service is in operation from 8:00 a.m. to 10:00 p.m. Eastern Time, Monday through Friday.

If Customer Service advises you to return your Digital Data Pack, please return it postage prepaid and insured, with your name, address, proof of the date of purchase and a brief description of the problem to the Service Center you have been directed to return it to. If your Digital Data Pack is found to be factory defective during the first 90 days, it will be repaired or replaced at no cost to you. If the Digital Data Pack is found to have been consumer damaged or abused and therefore not covered by the warranty, then you will be advised, in advance, of repair costs.

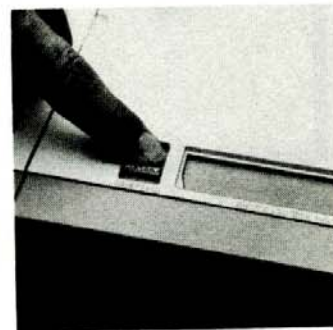
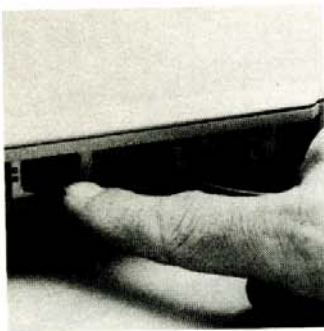
If your Digital Data Pack requires service after expiration of the 90 day Limited Warranty period, please call Coleco's toll-free service hotline for instructions on how to proceed: 1-800-842-1225 nationwide.

IMPORTANT: SAVE YOUR RECEIPTS SHOWING DATE OF PURCHASE.



Starting Out

Chapter 2



1. Turn the computer **ON**.
2. Insert the SmartLOGO digital data pack.
3. Press the **computer RESET** button.

Meet the Turtle

A journey of a thousand miles starts with a single step. Our first step toward programming is to meet the turtle.

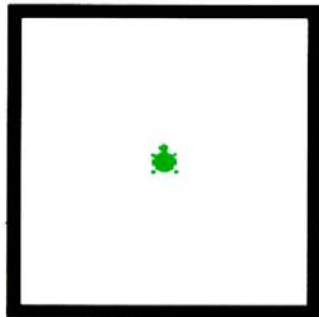
The turtle is a computer creature that lives on the screen. The turtle will draw if you tell it what to do. You talk to it by typing. Try:

? FORWARD 50

Don't forget the space. —

Computer people
make zero like this: Ø.
Ø is the number zero.
O is the letter O.

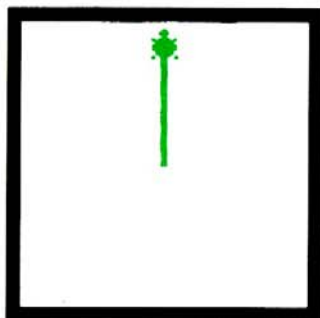
This is what you see:



Why didn't it go forward?...

Press **RETURN** to show the
computer you have come
to the end of your
instruction.

You should now see this:



If this doesn't work, do FORWARD 5Ø again. Perhaps you forgot the , or typed O instead of Ø.

What about:

?CS FD

Put in any number: try a *very* big number.

Leave out the space, and see what happens:

?FD5Ø

Leave out the number, and see what happens:

?FD

Each time, Logo came back with a message on the screen. Don't worry about it. Just try again.

Two more instructions

BACK or BK
LEFT or LT

Try FORWARD 25 and other numbers. To save typing, use FD. It works just like FORWARD.

This is a command.

This is the input.

`FORWARD 50`

The whole thing is an instruction.

A new command

? `CLEARSCREEN` or `CS`

No input needed `RETURN` is still needed.

Why no input? `FD` needs an input to tell it how far to move the turtle. But `CS` is just *clear the screen*. It doesn't need any more information.

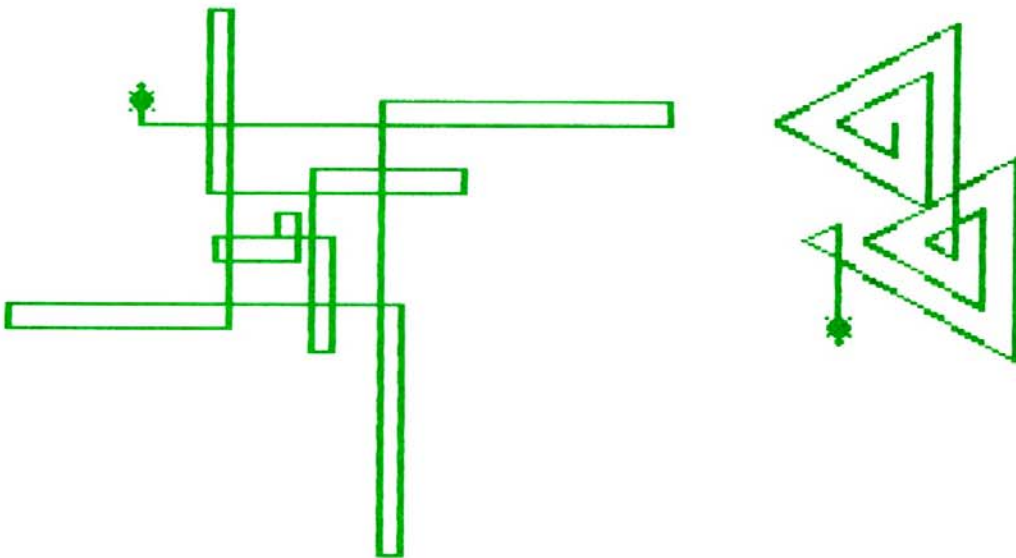
To make the turtle face another direction:

? `RIGHT 90` or `RT 90`

Space

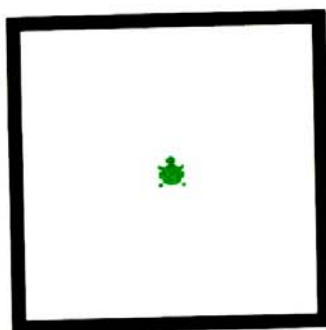
Don't forget `RETURN`.

Try `FD` `RT` `FD` `RT` and so on.

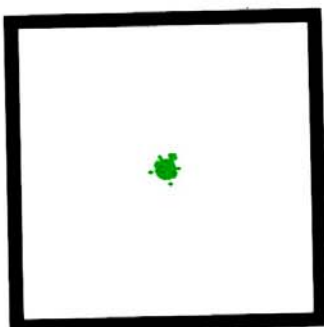


The means you fill in any numbers you want. If the numbers are small, the turtle moves only a little. If they are very big, the turtle may surprise you.

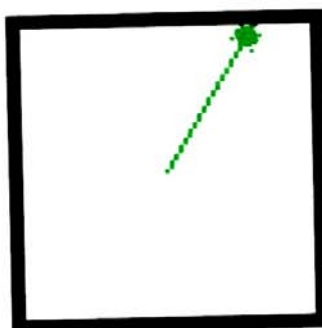
Try this:



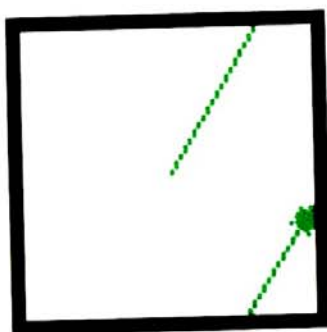
? CS



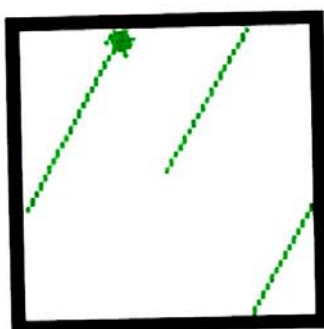
? RT 30



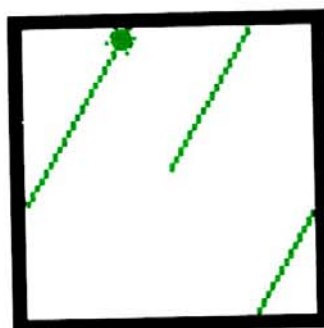
? FD 111



? FD 140



? FD 78



?

The turtle can move without drawing lines. You just have to lift its pen.

That's how you would draw something like this:



Here are the commands you need:

PENUP or PU

PENDOWN or PD

Try them out:

```
?CS RT 90
?PD FD 10
?PU FD 10
?PD FD 10
?PU PD 10
```

Another new command:

```
?REPEAT 6 [PD FD 10 PU FD 10]
```

Some more commands

?HOME

If you think nothing is
happening, try
FD 50 RT 90 FD 50 HOME.

?CLEAN

Hint: Try FD 50 CLEAN.

?HT (for *HideTurtle*)

Look for the turtle!

?ST (for *ShowTurtle*)

?PE or PENERASE Hint: Try PD FD 50 PE BK 25.

If the turtle has disappeared, type ST.
If the turtle won't draw, try PD.

Some Fancy Stuff

Two ways to draw a square:

● The long way

```
?FD 50 RT 90
?FD 50 RT 90
?FD 50 RT 90
?FD 50 RT 90
```

● The short way

```
?REPEAT 4 [FD 50 RT 90]
```

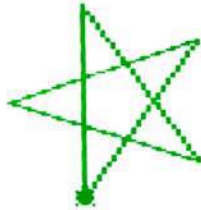
↑
This is a command.
It has two inputs.

↑
One input is a
number.

↑
The other is a *list*
of instructions:
[FD 50 RT 90]
A list is always inside an
envelope:
[].

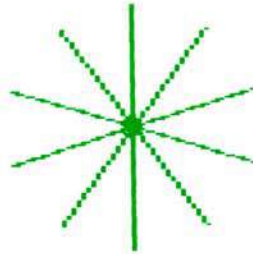
REPEAT can be a lot of fun. With different inputs, you
can get some amazing results.

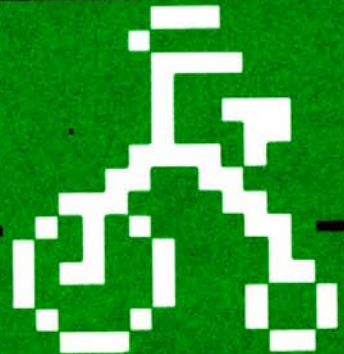
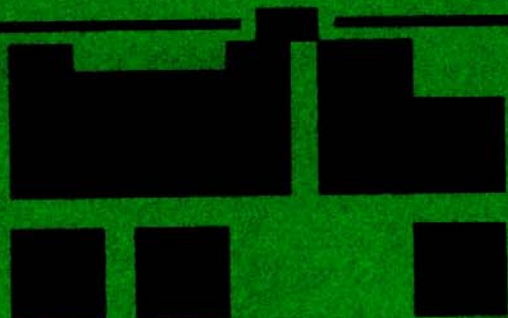
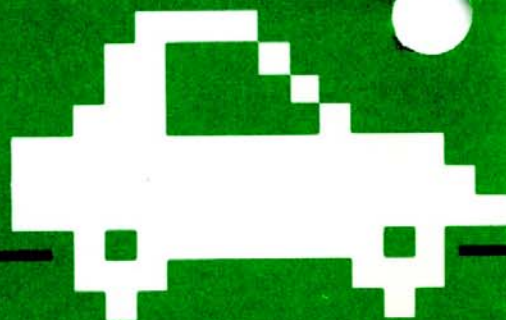
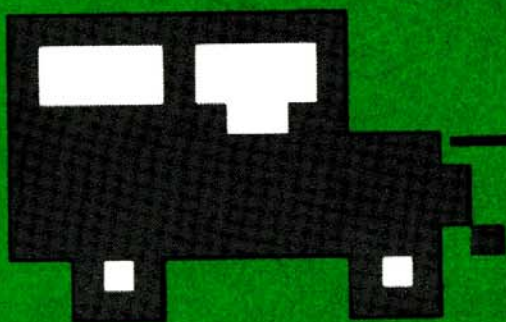
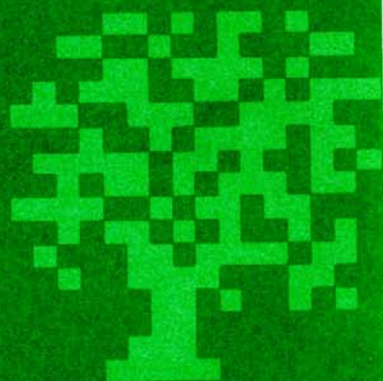
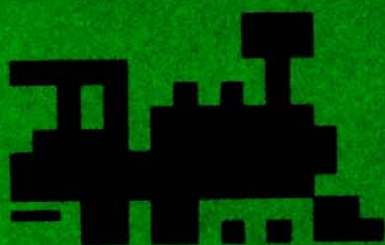
```
?REPEAT 5 [FD 75 RT 144]
```



The list envelope is [], not ().

```
?REPEAT 10 [FD 50 BK 50 RT 3→  
6]
```





Motion, Shapes, and Colors

Chapter 3

Let's Get Moving

? SETSPEED 25 or SETSP 25

The turtle should be moving. If it isn't, do
SETSPEED 25 **RETURN** again.

While it's moving, change directions with RT 90 or LT 15 or other inputs. Do PU, CLEAN, PD while it's moving, and see what happens. Then try FD 50.

See how fast you can make it move, then see how slow you can make it move.

How do I make it stop?

*Give it the smallest input
you can.*

Well... I gave it 1.


*Oh... Ø speed means it
doesn't move.*

Try something smaller.

Shapes and Colors

You can also set the turtle's shape and color.

? **SETSHAPE 5** or **SETSH 5**

The input is the number of the shape.

Each shape in the computer's memory has a number, starting with Ø.

*How many shapes are
there?*

Check it out and see!

Colors also
have numbers

? **SETCOLOR 1** or **SETC 1** makes the shape
black.

? **SETCOLOR 2** or **SETC 2** makes the shape
green.

The turtle can draw in different colors too! Try this:

? **SETPENCOLOR 9** or **SETPC 9**

and then:

? **FD 30**

To get back the original turtle shape:

? **SETSH 36**

To get back the original turtle color:

? **SETC 15**

More Turtles

So far you have only seen one turtle. But there are really 30 of them. Each turtle has a number. The turtle you already know is turtle 0.

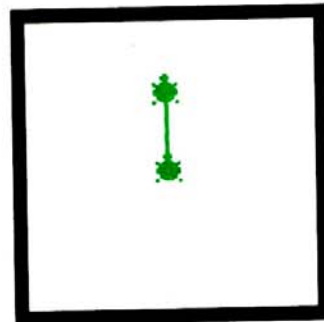
If it's the first turtle, why isn't it turtle 1?

Computer people like to count 0, 1, 2, 3... instead of 1, 2, 3, 4... like everyone else.

To get turtle 1 out from hiding:

? **TELL 1 ST**
? **FD 45**

*Oh yeah...
ST means Show Turtle.*



Now you should see two turtles on the screen. All instructions now go to turtle 1. Turtle 0 ignores them. Only turtle 1 is listening.

To make turtle 0 listen again:

```
? TELL 0
```

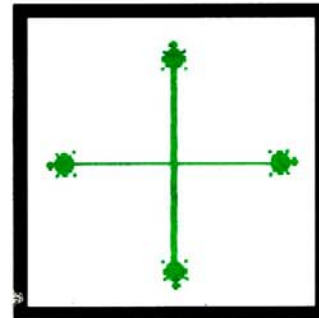
Or to make both turtles listen at the same time:

```
? TELL [0 1]
```

*Remember, this is a list,
and it needs a list envelope
[].*

Try this:

```
? TELL 1 RT 90  
? TELL 2 RT 90+90  
? TELL 3 RT 90+90+90  
? TELL [0 1 2 3]  
? ST SETSP 25
```



They should all be moving now.

To make Logo listen to only the first turtle
(like before):

```
? TELL ALL CS HT  
? TELL 0 ST PD
```

A Project

Turtles zooming along a road:



Do it yourself or follow our plan. (Best of all: do it yourself, *then* look at our plan to see if we had the same ideas as you did.)

Turtle Talk

```
? TELL ALL
? CS HT

? TELL Ø ST
? PU FD 16

? RT 90
? PD FD 256

? LT 90 PU
? BK 32 RT 90

? PD FD 256

? PU HOME

? RT 90
? REPEAT 16 [PD FD 8 PU FD 8]

? LT 90 BK 8
? RT 90
? SETSPEED 10

? TELL 1 ST
? PU FD 8
? LT 90
? SETSPEED 10
```

People Talk

Gets rid of everything.

Turtle Ø is listening and showing.
Turtle moves to the top of the road.

Turtle turns, ready to draw.
Draws the top of road, all the way across.

Turtle turns, facing up again.
Turtle backs down to other side of road.
Draws the bottom of the road, all the way across.

Puts the turtle back in home position.
Turns to draw the center line.

Draws the dotted center line.

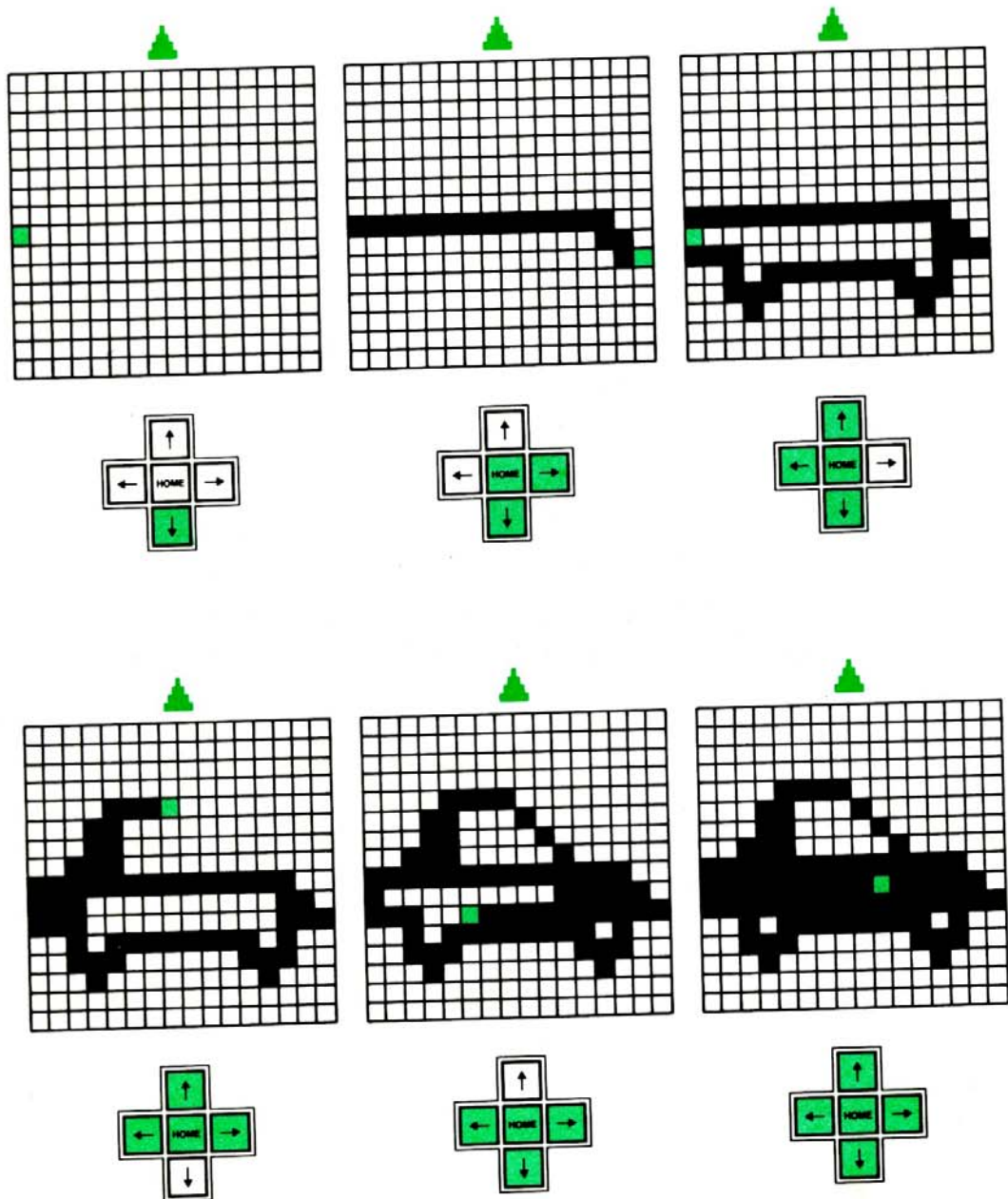
Turtle backs into the road.
Points to the right and drives away.

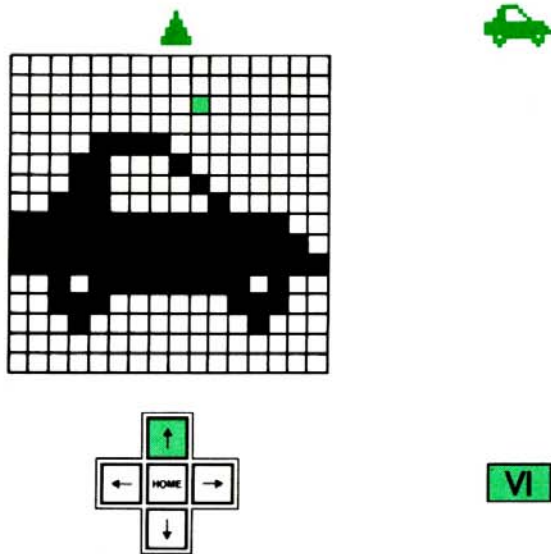
Turtle 1 is listening and showing.
It moves into the other lane points to the left and drives away.

Would you like the turtles to look like cars? All you have to do is make car shapes for them to wear.

Well, you've got it. You see how the cursor draws and how it erases. So now you can change the shape into whatever you want.

When you have finished making the new shape, press the **VI** key.





This tells the *Shape Editor* to go away until you want to change another shape. Look at the turtle! It's now wearing the shape 8 you just made.

If you don't like the changes you made to shape 8, and want to start again, press the **ESCAPE** key then to quit the Shape Editor and ES 8 again.

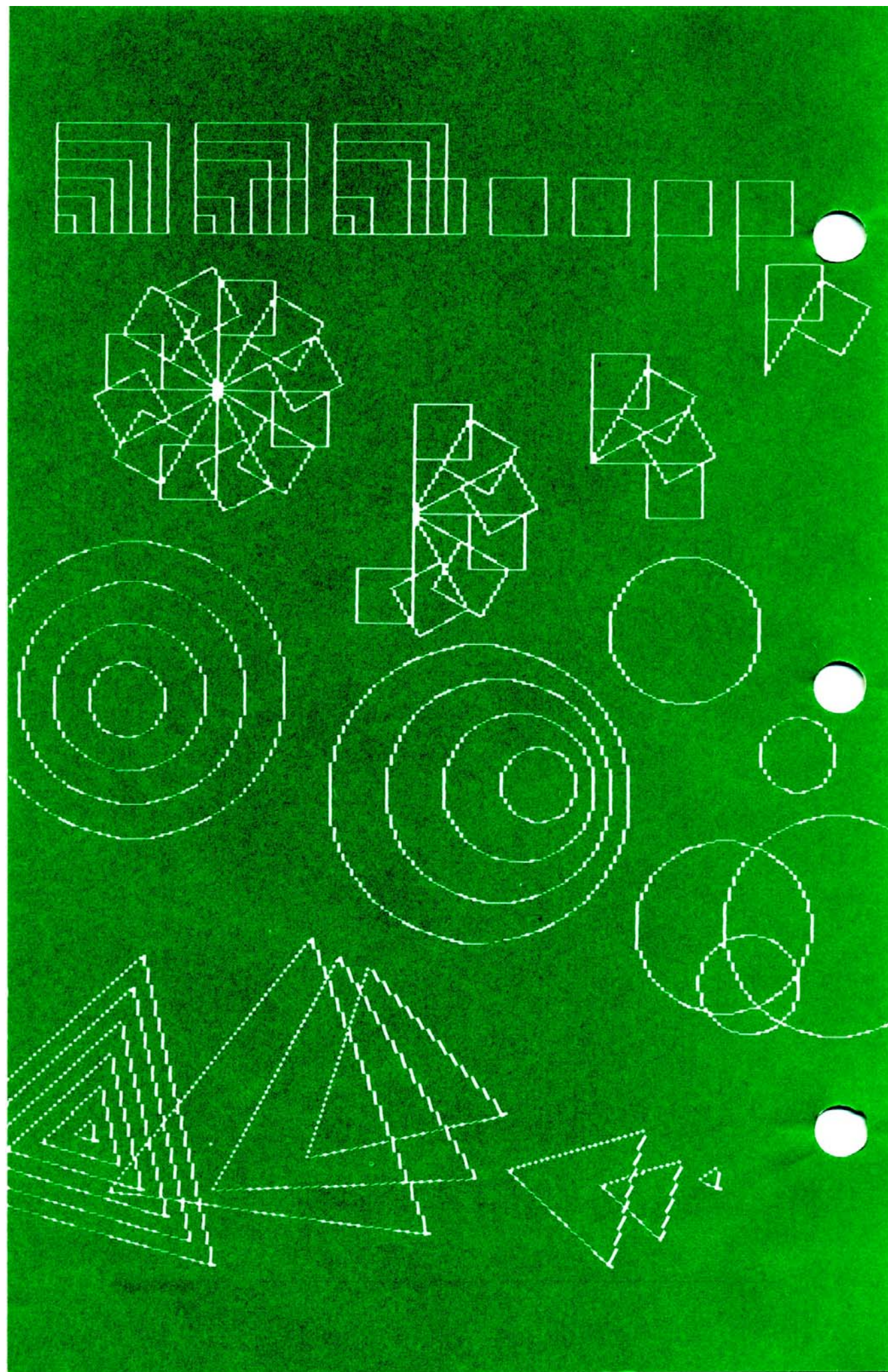
Now back to your project. Make a car shape and TELL two turtles to SETSHAPE 8. Then get them moving on the road.

Perhaps you want a second car shape? You'll have to edit another shape (say, shape 7) for that.









At last it's all working! But you had to type a whole set of instructions several times. Here's how to avoid that.

Teaching the Computer a New Command

If you wanted to draw a lot of squares, it would help if you could just type **SQUARE**. So let's try it. Something interesting might happen.

```
? SQUARE  
I DON'T KNOW HOW TO SQUARE
```

Instead of drawing a square, the computer complained: **I DON'T KNOW HOW TO SQUARE**.

Okay, if it doesn't know how to square, you can *teach* it how to square:

```
? TO SQUARE
```

and Logo prints on the screen:

```
TO SQUARE ■  
END
```

The computer learned a new word: **SQUARE**. Here is a good word for you to learn: *procedure*. A procedure is simply a bunch of computer instructions with a name.

This whole thing is a procedure:

```
TO SQUARE
FD 50 RT 90
FD 50 RT 90
FD 50 RT 90
FD 50 RT 90
END
```

TO SQUARE is the title line of the procedure.

The procedure is called **SQUARE**.

END tells Logo that the procedure stops here.

Changing the Procedure

When you try your procedure, maybe you don't like what it does. If you want to change something, just type TO SQUARE again.

Make any changes you want, using the arrow keys to move the cursor and **DELETE** to erase things. Then type in what you want.

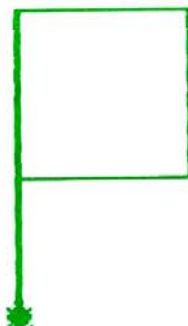
When you have finished, press **VI**.

Try this:

```
? REPEAT 4 [ SQUARE RT 90 ]
```

You can also use **SQUARE** as a command inside another procedure.

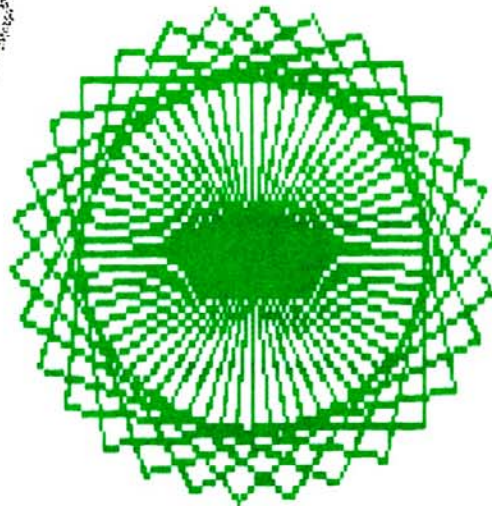
```
? TO FLAG
TO FLAG
FD 40 SQUARE BK 40
END
```



```
? TO STAR  
TO STAR  
REPEAT 30 [ SQUARE RT 12 ]  
END
```

Look at this instruction list.
There are two instructions:
SQUARE and RT 12.

*Who said this has to be 30?
Why 12?*



FLAG and STAR will work only if you have defined
SQUARE. SQUARE is called a *subprocedure* of FLAG or STAR.

Procedures with Inputs

This is a command. — This is its input.

FD 50

Together they are called an instruction.

This is also a command. — It has no input.

PU

It is a full instruction by itself.

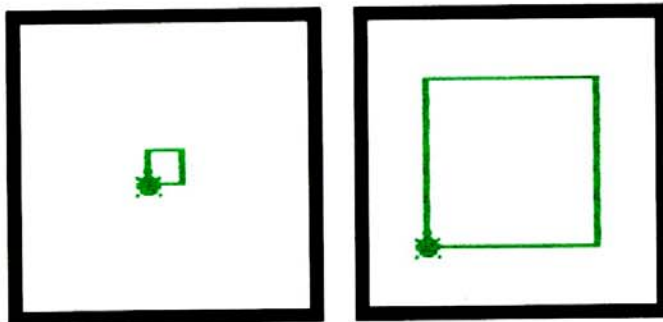
Some commands *need* inputs. FORWARD needs an input. Otherwise, it doesn't know how far to make the turtle move. But PENUP just tells the turtle to lift its pen. You don't have to tell it how high to lift the pen!

You can think of the command as a little person inside the computer. This person has a job to do, and maybe he or she needs some information to do it. For example, the FORWARD person needs information about how far to go.

SQUARE is like PENUP — you don't have to tell it anything more. It makes the turtle draw a square 50 by 50.

It would be great to have a SQUARE procedure that worked like this:

? SQUARE 10 making a *tiny* square, and
? SQUARE 100 making a much *bigger* square.



Let's begin by getting rid of the old SQUARE:

? ERASE "SQUARE or ER "SQUARE
 ↑ ↑
 Don't forget the quotes!

Now test to see if that worked:

? SQUARE
I DON'T KNOW HOW TO SQUARE

The machine has forgotten — SQUARE has been erased.

Old SQUARE procedure

```
FD 50 RT 90
FD 50 RT 90
FD 50 RT 90
FD 50 RT 90
```

New SQUARE procedure

FD what? You can't put in a number because the procedure must work for *all* sizes.

So, invent a *name* for the size of the square. *SIZE* is a good name for the number. Then you do the procedure like this:

```
FD :SIZE RT 90
FD :SIZE RT 90
FD :SIZE RT 90
FD :SIZE RT 90
```

↑ Remember the dots.

One important thing: on the TO line (the title line) in the editor we have to tell SmartLOGO that this procedure needs an input, and the name we are using for the input is *SIZE*. This is how to do it:

Old SQUARE procedure

```
TO SQUARE
FD 50 RT 90
FD 50 RT 90
FD 50 RT 90
FD 50 RT 90
END
```

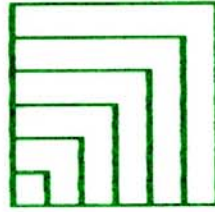
New SQUARE procedure

```
?TO SQUARE
TO SQUARE :SIZE
FD :SIZE RT 90
FD :SIZE RT 90
FD :SIZE RT 90
FD :SIZE RT 90
END
```

Press VI to save it.

Try it. (Don't forget to test what happens if you forget to give it an input!)

? SQUARE 10
? SQUARE 20
? SQUARE 30
? SQUARE 40
? SQUARE 50
? SQUARE 60



•
•
•

Try some numbers of your own!

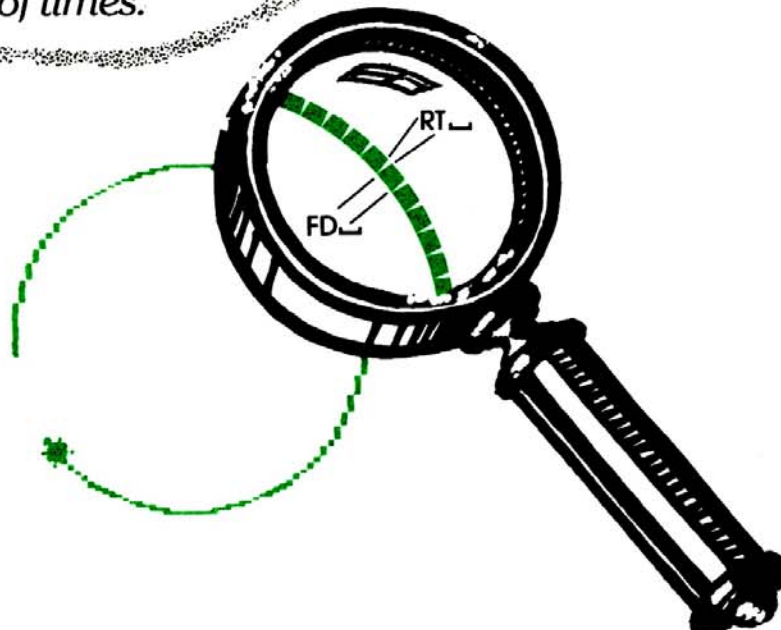
Circles and "Magic" Numbers

How do I make a circle?

*By playing turtle. Pretend
that you're a turtle and
walk in a circle.*

*I went forward a little,
turned a little, and did it
lots of times.*

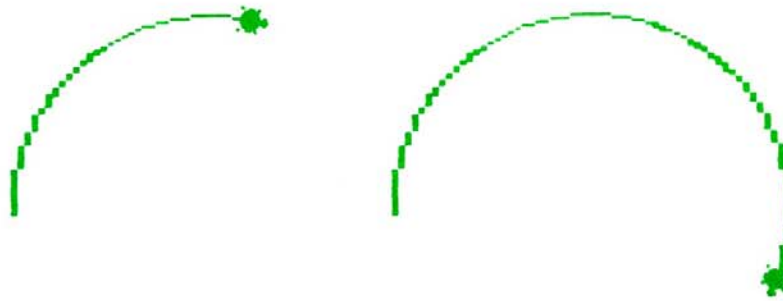
What did you do?



```
REPEAT  [FD 1 RT 1]
```

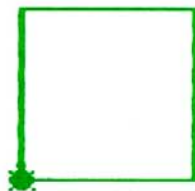
But how many times must the turtle do [FD 1 RT 1] to make a circle? It must be a lot, so you could try:

```
?REPEAT 100 [FD 1 RT 1]  
?REPEAT 200 [FD 1 RT 1]
```

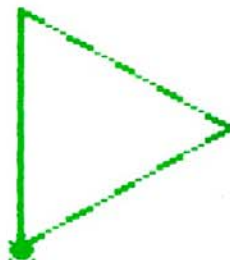


You can go on guessing or try something else.
Do these:

```
?REPEAT 4 [FD  RT 90]
```



```
?REPEAT 3 [FD  RT 120]
```



The turtle does RT 90 four times to make a square — and does RT 120 three times to make a triangle. Each time, the turtle turns RT 360 to get all the way around! This is the *Total Turtle Trip Theorem*.

Turtle Talk

REPEAT 4 [FD 50 RT 90]

REPEAT 3 [FD 50 RT 120]

So, to make a circle:

REPEAT 360 [FD 1 RT 1]

Theorem Talk

4 times RT 90 is a turn of 360: a square.

3 times RT 120 is a turn of 360: a triangle.

360 times RT 1 is a turn of 360: a circle.

The turtle goes forward 1 step, turns 1, and does this 360 times until it's back where it started. It's made a circle.

You can use the same idea to make curves. They're just unfinished circles.

Subprocedures

Remember the ROAD project? It's just the sort of thing to put into procedures. It's easy. Just invent a name for the procedure and write in all the instructions you need.

```
? TO ROAD
TO ROAD
TELL ALL CS HT
TELL 0 ST
PU FD 16
RT 90
PD FD 256
LT 90 PU
BK 32 RT 90
PD FD 256
PU HOME RT 90
REPEAT 16 [PD FD 8 PU FD 8]
END
```

That's the name.

These instructions were copied from page 17.

Press **VI** to save it.

Simple, but a little hard to read. A better idea is to see if *part* of the job can be done by a subprocedure.

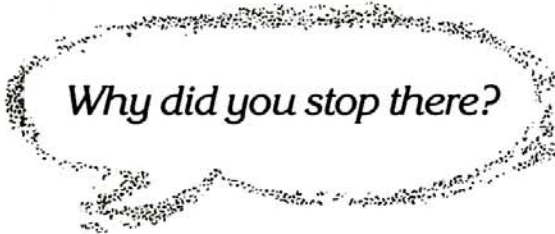
```
? TO EDGE  
TO EDGE  
PU FD 16 RT 90  
PD FD 256  
PU LT 90 BK 16  
END
```

Press **VI** to save it.

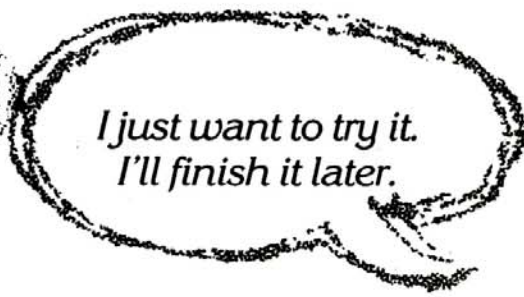
Now we can do:

```
? TO NEWROAD  
TO NEWROAD  
EDGE  
RT 180  
EDGE  
END
```

Press **VI** to save it.

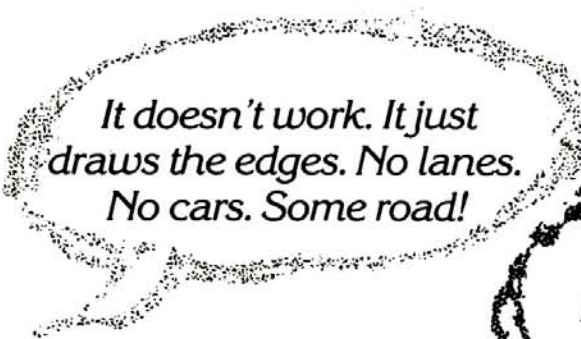


Why did you stop there?

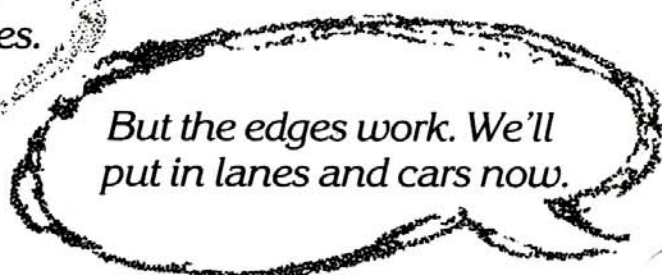


*I just want to try it.
I'll finish it later.*

```
? NEWROAD
```



*It doesn't work. It just
draws the edges. No lanes.
No cars. Some road!*



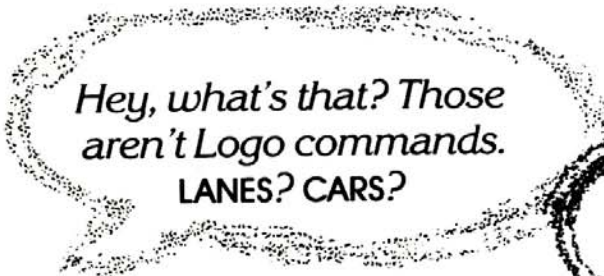
*But the edges work. We'll
put in lanes and cars now.*

Now type TO NEWROAD. Logo will print the procedure out for you as far as it's gone.

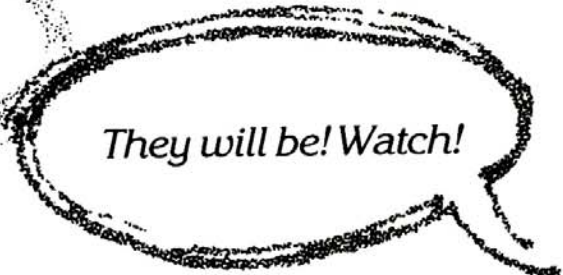
```
?TO NEWROAD
TO NEWROAD
TELL ALL CS HT
TELL 0 ST
EDGE
RT 180
EDGE
LANES
CARS
END
```

Now we're working on this line.

Press **VI** to save it.



*Hey, what's that? Those
aren't Logo commands.
LANES? CARS?*



They will be! Watch!

```
?TO LANES
TO LANES
PU RT 90 BK 128
REPEAT 16 [PD FD 8 PU FD 8]
BK 128 LT 90
END
```

Press **VI** to save it.

```
?TO CARS
TO CARS
TELL 0 ST
PU HOME
BK 8 RT 90
SETSP 10
TELL 1 ST
PU
FD 8 LT 90
SETSP 10
TELL [0 1]
END
```

Press **VI** to save it.

To test them out, type CS LANES, CS CARS. Then CS and try NEWROAD to see if the procedures work together.

Here are some new commands

POTS
POALL

POTS prints out the titles of all the procedures you've put into the computer so far. POALL prints out the titles *with* the instructions that make up the procedures.

Printing on Paper

Here's how to print your procedures out on the SmartWRITER printer:

?PRINTER

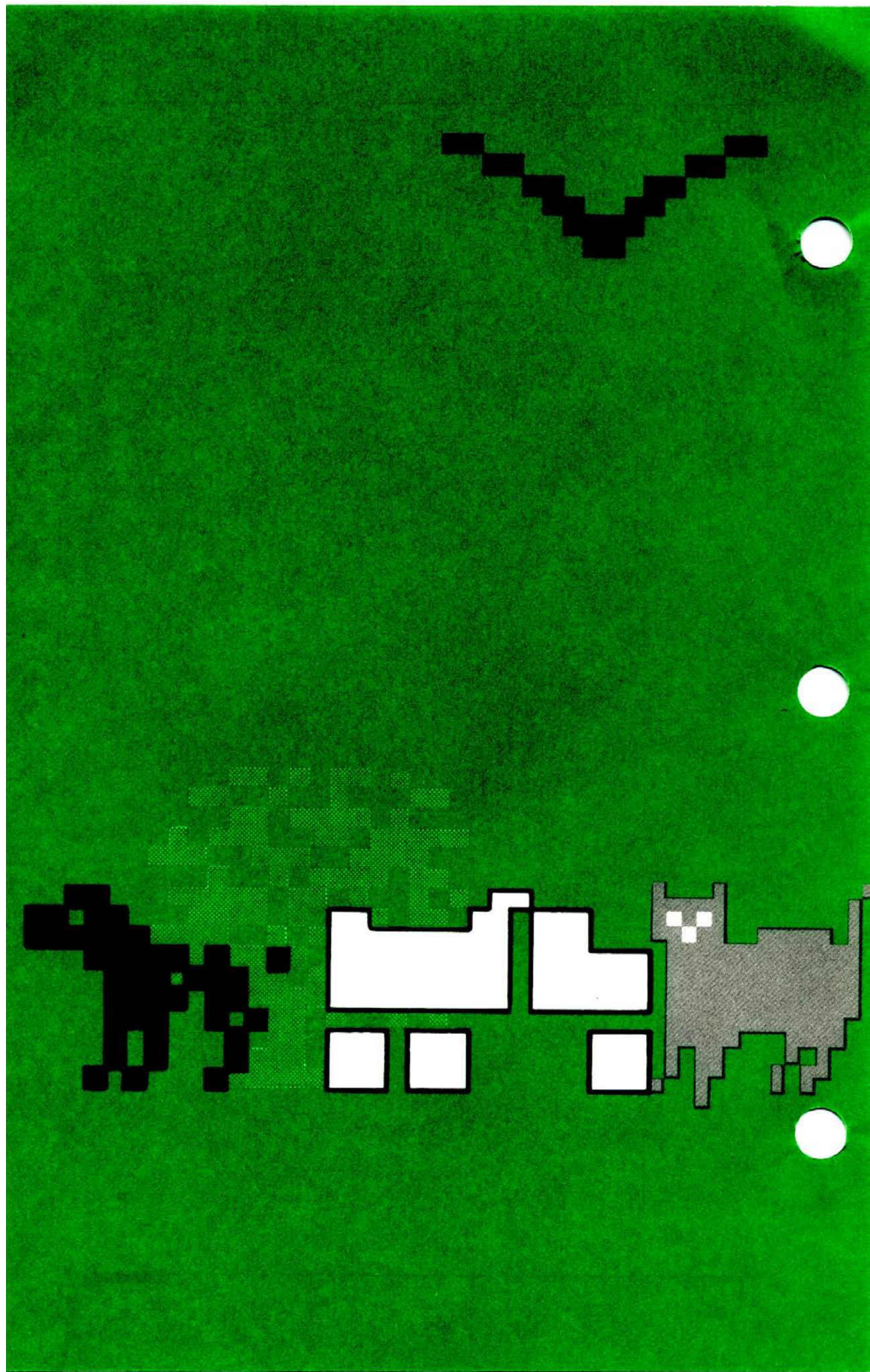
Turns the printer on.

?POALL

Prints out all your procedures on the screen and the printer.

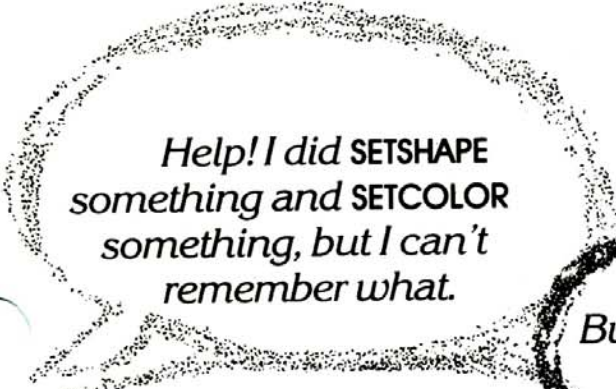
?NOPRINTER

Turns the printer off.

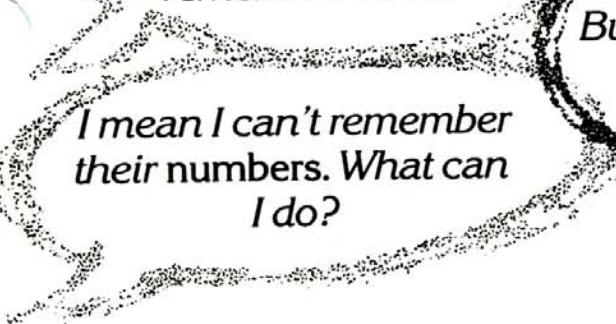


Reporters

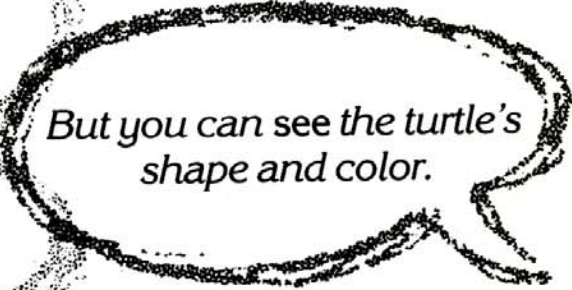
Chapter 5



Help! I did SETSHAPE something and SETCOLOR something, but I can't remember what.

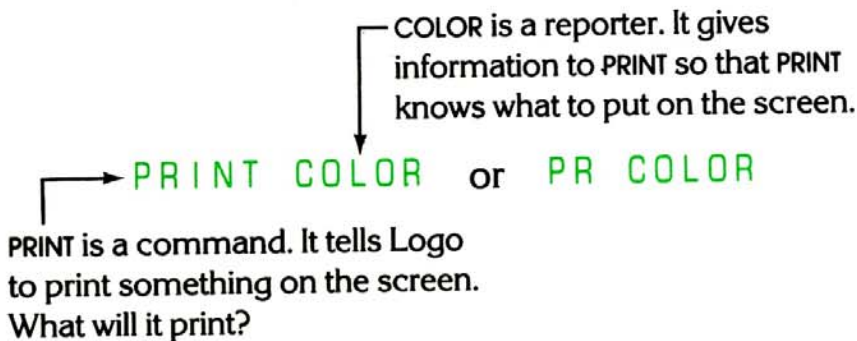


I mean I can't remember their numbers. What can I do?



But you can see the turtle's shape and color.

You can use a new kind of procedure. They're called *reporters* because they give reports. This is how you use them:



`PRINT COLOR` or `PR COLOR`

PRINT is a command. It tells Logo to print something on the screen. What will it print?

COLOR is a reporter. It gives information to PRINT so that PRINT knows what to put on the screen.

Just to see how it works, try this:

```
? SETC 3  
? PR COLOR  
3  
? SETC COLOR+1  
? PR COLOR  
4
```

```
? TO NEXT  
TO NEXT  
SETC COLOR+1  
END
```

That's a good trick: you're adding 1 to the number given by COLOR, then sending the result to SETC.

Press **VI** to save it.

NEXT will make it change color automatically.

```
? NEXT  
? NEXT  
? NEXT
```

Sometimes the turtle disappears! It becomes the same color as the background, or it becomes color Ø, the special invisible color. Another NEXT will make it visible again.

```
? REPEAT 100 [NEXT]
```

*Now it's going so fast
I can't see it!*

```
? REPEAT 50 [NEXT WAIT 30]
```

A new command. —↑

—↑ Play with its input.

Just to do something different, try this:

```
? TO WOW  
TO WOW  
NEXT  
WAIT 30  
WOW  
END
```

This procedure uses a trick called *recursion*. The WOW just before END tells Logo to do WOW again... and again... and again... and again...

To make it stop, press the ESCAPE key at the top of the keyboard.

More Reporters

Each SET-command has a reporter to go with it.

```
? SETSP 55  
? PR SPEED  
55  
? SETSH 6  
? PR SHAPE  
6
```

You can play the same games with SHAPE that you did with COLOR.

```
? SETSH 0  
? REPEAT 20 [SETSH SHAPE + 1 →  
WAIT 100]
```

To get the turtle shape back, use SETSH 36.

One of the most important reporters tells which turtle is listening. This reporter is called WHO.

```
?TELL 6  
?PR WHO  
6  
?TELL 0  
?PR WHO  
0
```

Watch how you can use WHO. Let's talk to a whole bunch of turtles:

```
?TELL [0 1 2 3 4 5 6 7]  
?SETC 1  
?HOME  
?SETSH 4
```

They're all black.
They're all at the same place.
They're all balls.

Do you see only one ball instead of eight? That's because they're all on top of each other.

To get them separated, you can give each a different speed:

```
?TELL 1 SETSP 10  
?TELL 2 SETSP 20  
?TELL 3 SETSP 30  
.  
.  
.
```

But you can save lots of typing by using the EACH command with WHO:

```
?TELL [0 1 2 3 4 5 6 7]  
?EACH [SETSP WHO * 10]
```

Each turtle gets a command to set its speed to it's own number times ten

Now I can't get them to stop!

Remember:
TELL ALL CS HT
TELL 0 ST PD

C

C

C

IAL :N
OUTPUT 1]
:N * FACTORIAL :N - 1

RINT

-0.65789

(.3

*

*

7

2

7

SmartLOGO as a Calculator

Chapter 6

Try this:

? PR 4 + 5
9

Logo added 4 and 5 and gave the answer 9.

? PR 350 * 50
17500

Logo multiplied 350 times 50 and gave the answer 17500.

? RT 180 - 135

Logo subtracted 135 from 180 and used the answer 45 as the input to RIGHT. How far right did the turtle turn?

SmartLOGO has all the arithmetic features of a calculator built in!

The multiply sign on computers is the *, so to multiply 90 times 4, type

? 90 * 4

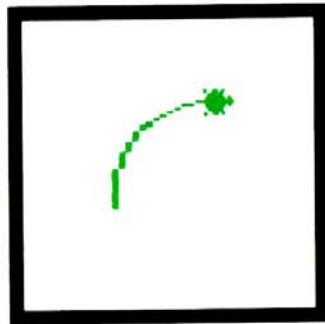
Like the reporters, the arithmetic *operations* send the answer somewhere — so you have to tell Logo where.

```
? 5+200  
YOU DON'T SAY WHAT TO DO WITH →  
H 205  
? PR 5+200  
205
```

Instead of trying to figure out how many times to REPEAT something, you can let Logo do the arithmetic for you. Try:

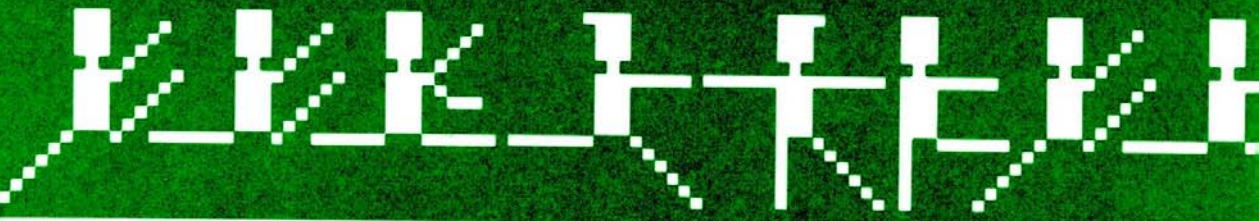
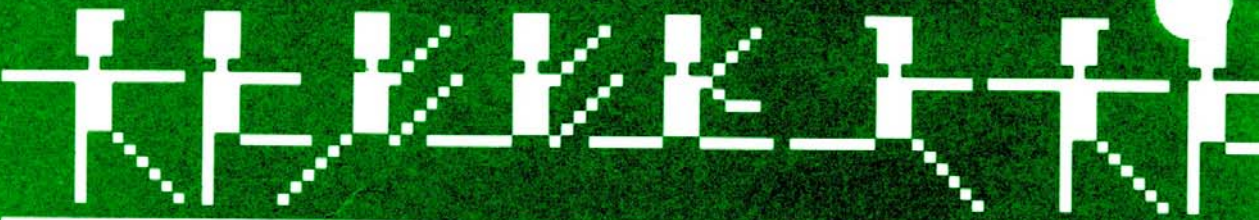
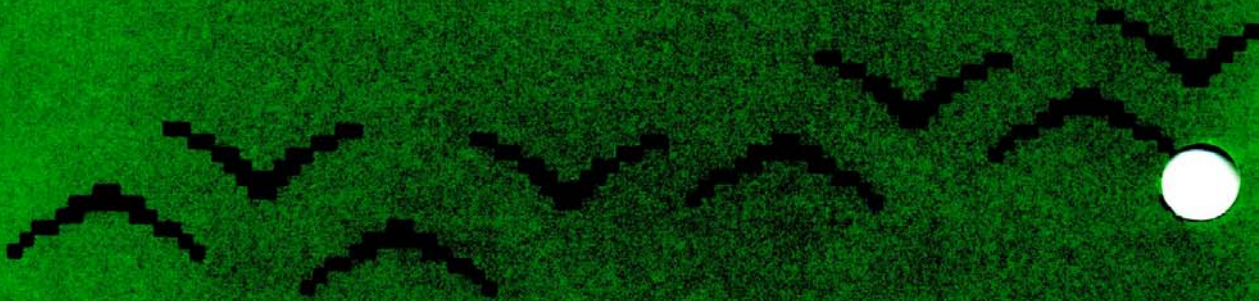
```
? REPEAT 360/4 [FD 1 RT 1]
```

The turtle draws a curve that's a quarter of a circle.










```
? SETSH 10
? SETSH 11
? SETSH 10
? SETSH 11
```

It should flap like a bird. Now you can animate it:

```
? TO FLAP
TO FLAP
  SETSH 10
  WAIT 20
  SETSH 11
  WAIT 20
  FLAP ← Here's that recursion trick again.
END
```

Press **VI** to save it.

When you did it by hand, you didn't need to use the command **WAIT**. Why? Because you did the waiting when you were typing.

First make sure that it worked:

```
? FLAP
```

Use the **ESCAPE** key to stop it. Now let's make it move:

```
? PENUP
? SETSP 20
? FLAP
```

Making more birds:

```
? TELL 2 ST PENUP
? SETC 1 HOME FD 50
? TELL [0 2]
? RT 45 SETSP 20 FLAP
```

How would you send all the birds home?

You mean to the middle of the screen? Easy:
TELL ALL HOME

Perhaps you've been making new shapes with the shape editor. Or maybe you've been playing around with the shapes that are already there. Each one needs a shape number — 25 for the truck, 19 for the flower, 27 for the rocket. After a while, you might get mixed up. It would be a lot simpler to give the shapes *names* instead of numbers.

In Logo there's a trick for doing that.

You can use the command NAME, like this:

```
? NAME 25 "TRUCK
```

↑ Watch that little quote mark.
You'll soon see why it's there.

Now you can say

```
? SETSH :TRUCK
```

↑ You'll soon see why those dots are there too.

You can give each shape a name.

```
? NAME 4 "BALL      You choose the name, any name  
? NAME 15 "STAR     you like.
```

Then you can change shapes by saying:

```
? SETSH :BALL  
? SETSH :STAR
```

And so on. By the way, the computer will still understand:

```
?SETSH 4  
?SETSH 5
```

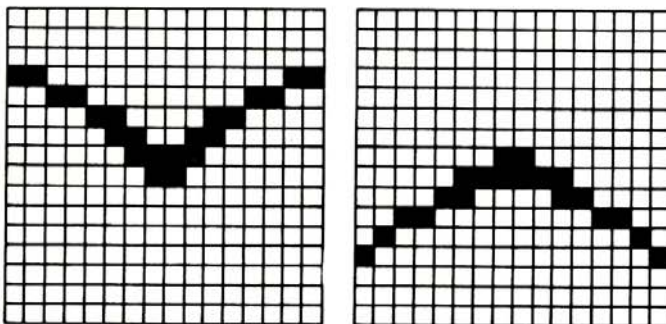
So you can use either *names* or *numbers*.

Names for Turtles and Names for Shapes

You can change the bird movie to work with names instead of numbers. First, make one bird fly. Choose a turtle to be the bird. Maybe 0.

```
?NAME 0 "BIRD
```

Choose two shape numbers, say 10 and 11, and make two shapes:




Now, choose two names for them:

```
?NAME 10 "UP  
?NAME 11 "DOWN
```

Watch this:

```
? TO BIRDS
TO BIRDS
TELL :BIRDS ST
PU CS
SETC 1
RT 60
EACH [FD WHO*10]
RT 30
SETSP 20
FLAP
END
```



*What about the sun and
the moon?*



Why not put them in?

Names of Things

Maybe you've wondered what the dots and quotes are all about. This section will help you think about it. Read it when you feel like taking a break from the computer.

You can explain dots and quotes by playing a game.

You say to someone: Say your name.

Maybe the person
says:

Barbara.

Then you say:

I didn't say Say Barbara,
I said **Say your name.**

Barbara will
probably say:

Your name.

Then you say:

Oh so that's what you
are called? **Your name?**

There are many forms of this joke. For example:

You: *What's the longest river in the U.S.?*
Victim: The Mississippi.
You: *Spell it.*
Victim: M-I-S-S-I-S-S-I-P-P-I
You: *Really? I thought it was spelled I-T.*

"Spell it" can mean two things:

Spell the *word* it: I-T

or

Spell *what* the word it *stands for*.

People usually understand which you mean from the tone of your voice. If you write it down, you use punctuation marks to show the difference. For example:

Spell it means spell the thing you're talking about.

Spell "it" means spell "it" itself.

Computers need punctuation marks too. In SmartLOGO, we do this with dots and quotes.

Quotes are a way to say that you mean the word itself.

Dots say that you mean what the word stands for.

```
?NAME 55 "AGE
```

```
?PR "AGE
```

```
AGE
```

It printed the word age.

```
?PR :AGE
```

```
55
```

It printed what the word age stands for.

```
?PRINT "BIRDS
```

```
BIRDS
```

```
?PRINT :BIRDS
```

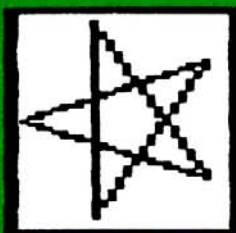
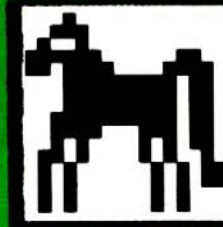
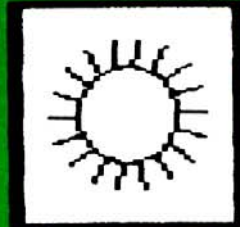
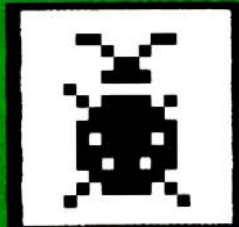
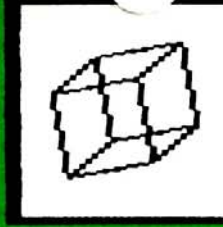
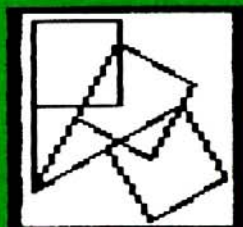
```
2 3 4 5 6 7 8 9 10 11
```

```
?PRINT "UP  
UP
```

```
?PRINT :UP  
10
```







Then, when you turn the computer back on, you type:

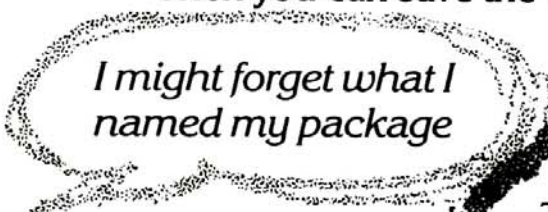
? **LOAD "TODAY**

and the computer will copy all the procedures in the TODAY package back into the workspace.

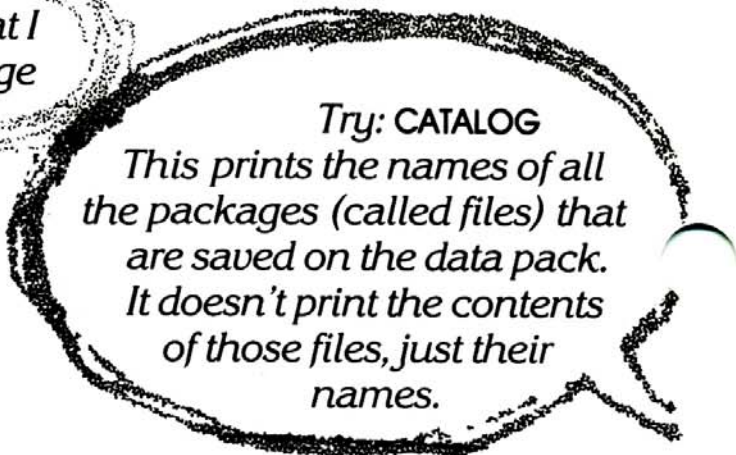
If you don't want to save everything in your workspace, you have to erase those things you don't want. Use POTS to see the titles, find which procedures you don't want (for instance SQUARE) and type:

? **ERASE "SQUARE**

Then you can save the workspace.



*I might forget what I
named my package*



*Try: CATALOG
This prints the names of all
the packages (called files) that
are saved on the data pack.
It doesn't print the contents
of those files, just their
names.*

Saving Pictures

You can also save any picture the turtle draws on the screen.

? **SAVEPICT "DRAWING** Or any other name you choose.

When you turn the computer back on, type:

? **LOADPICT "DRAWING**

SAVEPICT copies everything that is on the screen (including text) onto the data pack. So you may want to learn this command:

?CLEARTEXT or CT

This clears all the text from the screen, but doesn't change the picture.

Saving New Turtle Shapes

Saving new turtle shapes is a little more complicated.

The computer uses numbers to remember the patterns that make up a shape. These numbers are like a code. In order to save a new shape, you have to save those numbers. You do it with the command GETSH, which gets the code for a shape number.

Remember the FLAP procedure? You used shapes 10 and 11 as the wings.

```
?NAME GETSH 10 "WINGSUP  
?NAME GETSH 11 "WINGSDOWN
```

And the code for each shape now has a name. Next, save all the names and the things that go with them.

```
?SAVENS "WINGSHAPES
```

Then when you turn the computer back on, type:

```
?LOAD "WINGSHAPES
```

and

```
?PUTSH 10 :WINGSUP  
?PUTSH 11 :WINGSDOWN
```

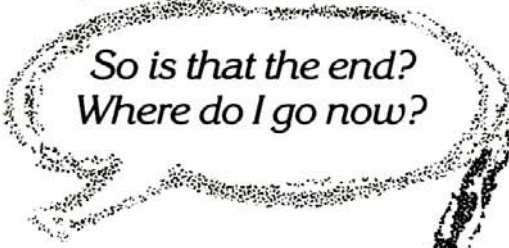
PUTSH (for putshape) tells shape 10 to become the pattern that we called WINGSUP. And shape 11 becomes the pattern we called WINGSDOWN.

When you do:

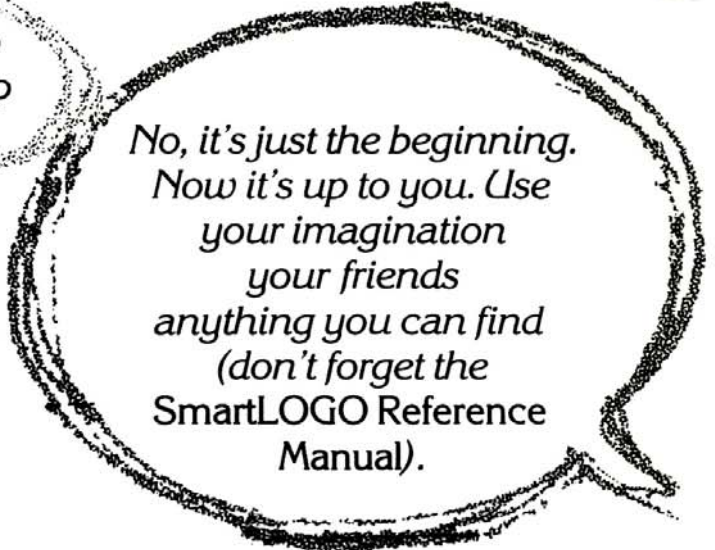
? SETSH 10

? SETSH 11

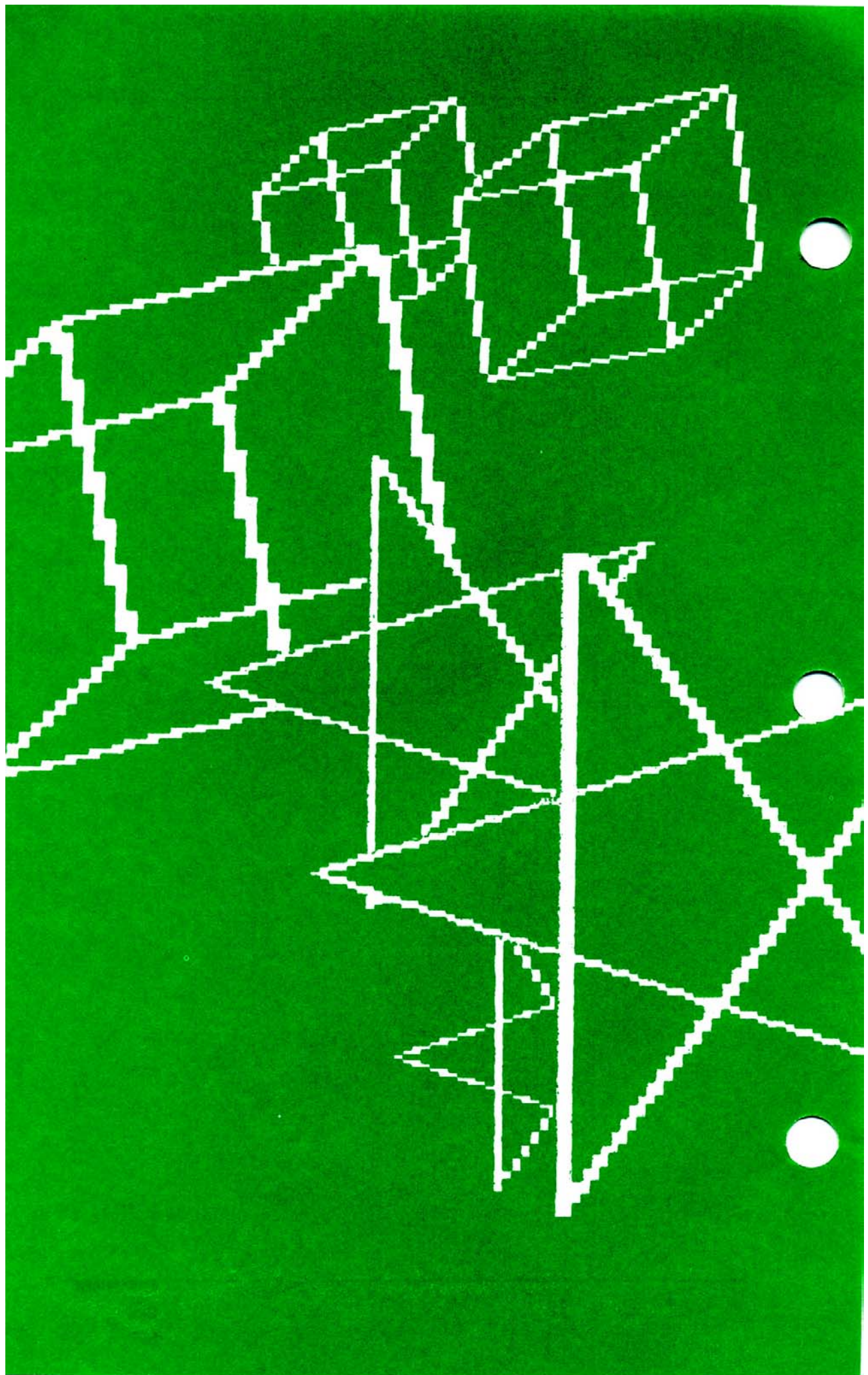
it flaps like a bird again.



*So is that the end?
Where do I go now?*



*No, it's just the beginning.
Now it's up to you. Use
your imagination
your friends
anything you can find
(don't forget the
SmartLOGO Reference
Manual).*



**P.S.
Some Things
to Experiment
With**

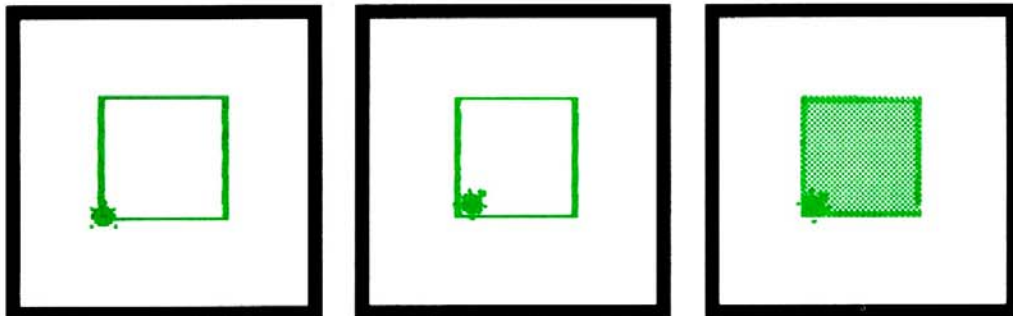
Chapter 10

Here are some projects you can try out. You'll find some new commands and ideas in them. If you play around with them you'll figure out how they work.

**Filling
in a square**

Try:

```
? SQUARE 50  
? PU RT 45 FD 5  
? PD SETPC 7 FILL  
? SETPC 14 FILL
```



Another way to stop and start

Try:

```
? TELL ALL
? CS ST PU
? EACH [SETH 12*WHO]
? SETSP 50
? FREEZE
? THAW
```

Getting turtles to bounce

Try this procedure:

```
TO BOUNCE
  TELL 0
  SETSH 4
  SETC 8
  ST PU
  FD 60
  TELL 1
  SETSH 4
  SETC 3
  ST PU
  BK 60
  TELL 2
  SETSH 4
  ST PU
  ON.TOUCH 2 1 [RT 180]
  ON.TOUCH 2 0 [RT 180]
  SETSP 20
END
```

This means that
when turtle 2
touches turtle 1 it
turns RIGHT 180.

ON.TOUCH needs 3 inputs: two numbers for turtles,
and a list of instructions to do when the turtles touch.

☾

☾

☾

██████████

Every effort has been made to ensure the accuracy of the product documentation in this manual. However, because we are constantly improving and updating our computer software and documentation, Logo Computer Systems Inc. is unable to guarantee the accuracy of printed material after the date of publication and disclaims liability for changes, errors or omissions.

No reproduction of this document or any portion of its contents is allowed without the specific written permission of Logo Computer Systems Inc.

© 1984 Logo Computer Systems Inc.
All rights reserved.



Printed in U.S.A. 14381



Logo Computer Systems Inc.
9960 Côte de Liesse Road
Lachine, Québec, Canada H8T 1A1

REFERENCE MANUAL

Acknowledgements

SmartLOGO Turtle Talk

author:	Seymour Papert
contributing editors:	John Berlow Eric Brown Wynter Snow
word processing:	Marie Barbeau

SmartLOGO Reference Manual

contributing editors:	Eric Brown Barbara Mingle
word processing:	Marie Barbeau

Graphic design and layout

Lorraine Lavigne
Richard Lavigne
Julien Perron

Exploring SmartLOGO

author/programmer:	Eric Brown
contributing editor:	Barbara Mingle

SmartLOGO Software team

analysts:	Mario Bergeron Mario Bourgoïn Mamadou Billo Diallo
testers:	Hani Fanous René Yelle

Project Management:	Michael Quinn
----------------------------	---------------

Contributions to Logo software and reference materials have been made by many members of Logo Computer Systems Inc. Many members of Coleco Industries Inc. have contributed to the preparation of the software and reference materials.

© 1984 Logo Computer Systems Inc.

All rights reserved. No part of the documentation contained herein may be reproduced, stored in retrieval systems, or transmitted, in any form or by any means, photocopying, electronic, mechanical, recording, or otherwise, without the prior approval in writing from Logo Computer Systems Inc.

Table of Contents

1 Preface

Chapter 1

5 Introduction

- 6 What You Need
 - 8 The Keyboard
 - 13 How Primitives are Described
 - 13 How We Describe Formats
 - 16 Special SmartLOGO Considerations
-

Chapter 2

21 Logo Grammar

- 21 Procedures
 - 23 Punctuation and Inputs to Procedures
 - 25 Commands and Operations
 - 26 Variables
 - 28 Global and Local Variables
 - 29 Understanding a Logo Line
-

Chapter 3

33 Defining Procedures

- 34 SmartLOGO Editor
-

Chapter 12

115 **Words, Numbers and Lists**

117 SmartLOGO Object Manipulators

118 SmartLOGO Object Reporters

119 Variables as Inputs

119 Inputs from the Keyboard

Chapter 13

137 **Variables**

Chapter 14

145 **Mathematical Operations**

147 Order of Mathematical Operations

Chapter 15

163 **Flow of Control and Conditionals**

Chapter 16

179 **Logical Operations**

Chapter 17

187 **Interacting with SmartLOGO: Peripheral Devices**

187 The Game Controllers

191 The Keyboard

197 The SmartWRITER Printer

199 Optional Devices

Chapter 18

201 **Workspace Management**

Chapter 19

209 **File Management**

214 Creating a Digital Data Pack for File Storage

	Chapter 20
217	Property Lists
222	A Sample Project Using Property Lists
	Chapter 21
227	Special Primitives
	Appendix A
233	Error Messages
	Appendix B
239	Program Files Included on the SmartLOGO Digital Data Pack
240	Using the Tutorial Files
241	Using the Demonstration Programs
242	Using the EASY Programs
242	Using the Useful Tools
	Appendix C
249	Startup
249	The STARTUP File
250	The STARTUP Variable
250	Changing the STARTUP File: A Note of Caution
251	A Sample STARTUP File and Variable
	Appendix D
253	Memory Space
253	How it Works
254	How Space is Used
255	Space Saving Hints

Appendix E

257 **Parsing**

- 257 Delimiters
- 258 Infix Procedures
- 258 Brackets and Parentheses
- 259 Quotes and Delimiters
- 260 The Minus Sign

Appendix F

263 **Summary of SmartLOGO Primitives**

Appendix G

287 **Glossary**

297 Index

90-DAY LIMITED WARRANTY

Coleco warrants to the original consumer purchaser in the United States of America that the physical components of this digital data pack (the "Digital Data Pack") will be free of defects in material and workmanship for 90 days from the date of purchase under normal in-house use.

Coleco's sole and exclusive liability for defects in material and workmanship of the Digital Data Pack shall be limited to repair or replacement at an authorized Coleco Service Center. This warranty does not obligate Coleco to bear the cost of transportation charges in connection with the repair or replacement of defective parts.

This warranty is invalid if the damage or defect is caused by accident, act of God, consumer abuse, unauthorized alteration or repair, vandalism or misuse.

ANY IMPLIED WARRANTIES ARISING OUT OF THE SALE OF THE DIGITAL DATA PACK INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE LIMITED TO THE ABOVE 90 DAY PERIOD. IN NO EVENT SHALL COLECO BE LIABLE TO ANYONE FOR INCIDENTAL, CONSEQUENTIAL, CONTINGENT OR ANY OTHER DAMAGES IN CONNECTION WITH OR ARISING OUT OF THE PURCHASE OR USE OF THE DIGITAL DATA PACK. MOREOVER, COLECO SHALL NOT BE LIABLE FOR ANY CLAIM OF ANY KIND WHATSOEVER BY ANY OTHER PARTY AGAINST THE USER OF THE DIGITAL DATA PACK.

This limited warranty does not extend to the programs contained in the Digital Data Pack and the accompanying documentation (the "Programs"). Coleco does not warrant the Programs will be free from error or will meet the specific requirements or expectations of the consumer. The consumer assumes complete responsibility for any decisions made or actions taken based upon information obtained using the Programs. Any statements made concerning the utility of the Programs are not to be construed as express or implied warranties.

COLECO MAKES NO WARRANTY, EITHER EXPRESS OR IMPLIED, INCLUDING ANY IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, IN CONNECTION WITH THE PROGRAMS, AND ALL PROGRAMS ARE MADE AVAILABLE SOLELY ON AN "AS IS" BASIS.

IN NO EVENT SHALL COLECO BE LIABLE TO ANYONE FOR INCIDENTAL, CONSEQUENTIAL, CONTINGENT OR ANY OTHER DAMAGES IN CONNECTION WITH OR ARISING OUT OF THE PURCHASE OR USE OF THE PROGRAMS AND THE SOLE AND EXCLUSIVE LIABILITY, IF ANY, OF COLECO, REGARDLESS OF THE FORM OF ACTION, SHALL NOT EXCEED THE PURCHASE PRICE OF THE DIGITAL DATA PACK. MOREOVER, COLECO SHALL NOT BE LIABLE FOR ANY CLAIM OF ANY KIND WHATSOEVER BY ANY OTHER PARTY AGAINST THE USER OF THE PROGRAMS.

This warranty gives you specific legal rights, and you may have other rights which vary from State to State. Some states do not allow the exclusion or limitation of incidental or consequential damages or limitations on how long an implied warranty lasts, so the above limitations or exclusions may not apply to you.

SERVICE POLICY

Please read your Owner's Manual carefully before using your Digital Data Pack. If your Digital Data Pack fails to operate properly, please refer to the trouble-shooting checklist in the Operating Tips Manual. If you cannot correct the malfunction after consulting this manual, please call Customer Service on Coleco's toll-free service hotline: 1-800-842-1225 nationwide. This service is in operation from 8:00 a.m. to 10:00 p.m. Eastern Time, Monday through Friday.

If Customer Service advises you to return your Digital Data Pack, please return it postage prepaid and insured, with your name, address, proof of the date of purchase and a brief description of the problem to the Service Center you have been directed to return it to. If your Digital Data Pack is found to be factory defective during the first 90 days, it will be repaired or replaced at no cost to you. If the Digital Data Pack is found to have been consumer damaged or abused and therefore not covered by the warranty, then you will be advised, in advance, of repair costs.

If your Digital Data Pack requires service after expiration of the 90 day Limited Warranty period, please call Coleco's toll-free service hotline for instructions on how to proceed: 1-800-842-1225 nationwide.

IMPORTANT: SAVE YOUR RECEIPTS SHOWING DATE OF PURCHASE.

ABSOLUTELY OUT OF SPACE

Your workspace is completely filled. If you want to continue, erase some procedures and names from your workspace.

YOU DON'T SAY WHAT TO DO WITH *OBJECT*

A Logo operation or object was given without a command preceding it.

TOO MUCH INSIDE ()'S

Parentheses were incorrectly placed in a Logo instruction. For example, parentheses surround more than one Logo expression.

NOT ENOUGH INPUTS TO *PROCEDURE*

A procedure or primitive requires more inputs to run.

) WITHOUT (

A closing parenthesis has no corresponding opening parenthesis or a closing parenthesis was found when an input was expected.

I DON'T KNOW HOW TO *PROCEDURE*

Logo has tried to execute "PROCEDURE" but can't find its definition.

PROCEDURE DIDN'T OUTPUT TO *PROCEDURE*

A procedure or primitive that requires an input was not given one and was followed on the same line by another procedure or primitive.

NUMBER TOO BIG

The result of an arithmetic operation is more than 1E39 or less than 1N38.

Preface

The *SmartLOGO Reference Manual* should be used for reference, to look for specific primitives or to browse around, looking for new ideas. **For a general introduction to SmartLOGO, read the first section of this book, *Turtle Talk*. It was written to help and encourage a beginner. First-time users should also go through the *Exploring SmartLOGO* tutorials provided on the SmartLOGO digital data pack. To use the tutorials or view the demonstration program, type **YES** after the question **TUTORIALS? YES OR NO**.**

The previous section of this book *Turtle Talk*, is a casual introduction to some of the concepts involved in programming turtle graphics. The following section, *SmartLOGO Reference Manual*, is much more formal. The first two chapters give a general introduction to SmartLOGO and the conventions used to define and use procedures. Following these, are chapters which provide concise descriptions of each SmartLOGO primitive and provide many sample programs (procedures).

There are several ways to use this manual. **Inexperienced programmers should go through the introductory chapter, read the chapters on turtle graphics and then do some programming.** Experienced programmers can find out what a specific primitive does by looking it up in the index or by checking Appendix F or the *Quick Reference Guide*. Browse through the book and you'll find many interesting ideas that can help you create your own programs.

The appendices include technical information such as: messages that appear on the screen, information on files included on the SmartLOGO digital data pack, handy procedures, and a summary of SmartLOGO primitives.

Throughout this manual, **green** text is used to represent what you type on the computer. Black text is used to represent what the computer displays. The words in *italics* are the inputs to primitives.

1



Logo is a language for computers. Compared with languages like English or French, Logo has a very small number of words and rules, but you can add new words. In fact this is what programming in SmartLOGO* is all about: using what exists to make new things, and then using the new things to make more new things.

The initial vocabulary words are called Logo *primitives*. These words deal with different kinds of computer functions, from adding and subtracting numbers, to drawing turtle graphics and manipulating words and lists of words.

A Logo program is a collection of procedures. Procedures are either *primitives* (pre-defined by SmartLOGO) or *procedures* defined by you. All the procedures you define (user-defined procedures) are built out of SmartLOGO primitives. For example, a program to draw a house may contain these procedures: HOUSE, BOX,

*SmartLOGO is the Logo language that was created especially for the ADAM™ computer. The word Logo is used to refer to general Logo features. SmartLOGO refers specifically to the Logo you will be using.

TRI, RIGHT, FORWARD and REPEAT. Of these, the last three are *primitives*. The first three are user-defined procedures.

```
TO HOUSE
BOX
FORWARD 50
RIGHT 30
TRI 50
END
```

```
TO BOX
REPEAT 4 [FORWARD 50 RIGHT 90]
END
```

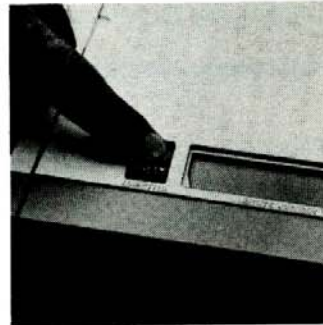
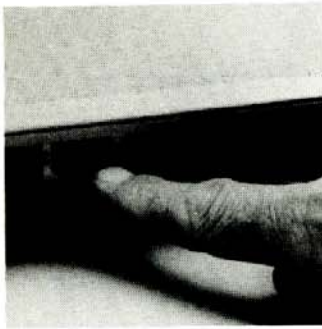
```
TO TRI :SIZE
REPEAT 3 [FORWARD :SIZE RIGHT 120]
END
```

Before going on, make sure you have read *Turtle Talk* and have viewed the *Exploring SmartLOGO* tutorial documentation. Working with SmartLOGO on your ADAM™ will help you understand the ideas in the following chapters.

What you need

In order to use SmartLOGO, you need:

- An ADAM™ computer.
- A video monitor or a television set. You can use either a black and white or a color set. If you use a black and white set, the colors described here will appear as shades of gray.
- A SmartLOGO package which contains two books: *Turtle Talk* and the *SmartLOGO Reference Manual*, and the SmartLOGO digital data pack.



1. Turn the computer **ON**.
2. Place the SmartLOGO digital data pack into the drive.
3. Press the computer **RESET** button. Wait for several seconds.

Remember to turn the machine on before putting the SmartLOGO digital data pack in the drive. If your ADAM™ is already on, just put the SmartLOGO digital data pack in and press the computer **RESET** button.

After a moment you'll see on the screen:

```
COPYRIGHT 1984
LOGO COMPUTER SYSTEMS INC.
WELCOME TO SMARTLOGO
TUTORIALS? YES OR NO NO
```

? ■

↑ This question may be removed.
See Appendix C, Startup. For the
moment type NO.

The ? (question mark) is called a *prompt*. When ? is on the screen, SmartLOGO is waiting for you to type something. The flashing ■ is the *cursor*. It moves as you type, showing you where the next character you type will appear.

The Keyboard

The ADAM™ Home Computer keyboard is set up like a typewriter.

Character Keys

Character keys include letters of the alphabet, numbers and punctuation marks. For example: **A**, **B**, **C**, **7**, **:**, **\$**.

RETURN Key

The **RETURN** key tells SmartLOGO: "Now do what I just typed." Press it when you've typed something you want SmartLOGO to do. In the Editor, pressing the **RETURN** key moves the cursor and the rest of the line to the next line.

SPACEBAR

The **SPACEBAR** prints an invisible but important character called space. SmartLOGO uses spaces to separate words. For example, SmartLOGO would interpret THISISAWORD as a single word and would interpret THIS IS A WORD as four words.

SHIFT Key

Holding down the **SHIFT** key, while pressing certain character keys, changes that particular character key's meaning in SmartLOGO. For example, if you hold the **SHIFT** key down and press 8, SmartLOGO will print * on the screen.

To make a shift character, always press the **SHIFT** key first and *hold* it down *while* typing the other key.

Arrow Keys



will move the *cursor* one space to the left.



will move the *cursor* one space to the right.



will move the *cursor* one line up.



will move the *cursor* one line down.

The *arrow keys* are useful editing keys. They move the *cursor* in the direction in which they point without affecting the text already there. (Note: The up and down arrows only work in the SmartLOGO Editors.) Once the *cursor* is positioned, you can insert or delete characters. To insert text, simply position the *cursor* and begin typing.

ESCAPE/WP Key

Pressing the **ESCAPE/WP** key signals SmartLOGO to stop whatever it is doing. When you press it, SmartLOGO types

```
STOPPED!!!  
? ■
```

then lets you type the next instruction.

Smart Key **V**

Pauses keyboard and screen operations. To prevent printed information from scrolling off the screen, press the Smart Key **V** once. Press it again to continue the scrolling.

Smart Key **VI**

The Smart Key **VI** is used to exit the SmartLOGO Editors. This key is discussed, along with other special editing keys, in Chapter 3, page 37, and Chapter 6, page 71.

BACKSPACE Key

Erases the character to the left of the cursor.

DELETE Key

Erases the character under the cursor.

INSERT Key

Opens up a new line at the present cursor position. This is most useful in the Editor, when you wish to insert a new line into an existing procedure.

CLEAR Key

Deletes all the characters from the present cursor position to the end of the current line. SmartLOGO holds this text in the delete buffer.

MOVE/COPY Key

Copies the last typed line. The **MOVE/COPY** key inserts a copy of the text that is in the delete buffer, at the current cursor position. The delete buffer retains the line of text until new text is put into the delete buffer. The delete buffer can hold 256 characters.

Special Characters

A **quotation mark**, `"`, used immediately before a word, indicates that it is being used as itself not as the name of a procedure or the value of a variable. We call this a *literal word*. Note: numbers are literal words that don't need a quotation mark.

Examples:

```
?PR "HELLO  
HELLO
```

```
?PR 5  
5
```

A **colon**, `:`, used immediately before a word, indicates that the word is to be taken as the name of a variable and produces the value of that variable.

Example:

```
?MAKE "PURPLE 13  
?PR :PURPLE  
13  
?SETC :PURPLE
```

Brackets, `[]`, are used to surround a list. They also group the members or items of a list together.

Example:

```
?MAKE "COLORS [RED BLUE]  
?SHOW :COLORS  
[RED BLUE]
```

Parentheses, (), are used to group things in ways that SmartLOGO ordinarily would not, or to change the order in which mathematical operations are performed. They also let you vary the number of inputs for certain procedures. For example, PRINT usually takes only one input but can take more if the primitive and its inputs are enclosed in parentheses.

Examples:

```
?PR 3*5+2
```

```
17
```

```
?PR 3*(5+2)
```

```
21
```

```
?MAKE "COLOR "BLUE
```

```
? (PRINT [ I HAVE A ] :COLOR "C →  
AR)
```

```
I HAVE A BLUE CAR
```

A **backslash, **, tells SmartLOGO to interpret the character that follows it literally *as a character*, rather than keeping some special meaning it might have. For instance, suppose you wanted to use 3[A]B as a single *word*. You need to type 3\[A]B in order to avoid SmartLOGO's usual interpretation of the contents of the brackets as being a list. You have to put a backslash before [, (,],), +, -, *, /, =, <, >, and \ itself. If you want to print a blank space, the \ will let you do this.

Example:

```
?PR [985-4482]
```

```
985 - 4482
```

```
?PR [985\ -4482]
```

```
985-4482
```

How Primitives are Described

The rest of this manual, except Chapter 2, Logo Grammar, consists of a listing and description of each SmartLOGO primitive.

In bold face at the top of each description, you will find the name of the primitive and its short form if one exists. Also indicated on that line is whether the primitive is a command or an operation or an infix operation. The difference between a command and an operation is described in Chapter 2, Logo Grammar. An infix operation is one that is placed between its inputs (for example, $4 + 5$). All other primitives are written in front of their inputs.

Below this, the name of the primitive, followed by the *type* of each input is shown. This line is called the format. You are to supply all inputs (shown in italics). A description of the types of inputs is in this chapter.

This is followed by the definition, general information about the primitive and illustrations of how to use the primitive.

How We Describe Formats

If a primitive has more than one format, we write one below the other, with the simplest or most commonly used on the top line. You will see that, with some primitives (such as PRINT), an optional format is surrounded by parentheses.

This indicates that the primitive will accept as many inputs as you wish but when using more than the indicated number of inputs with such a primitive, you must always put a left parenthesis before its name and a right parenthesis after the last input.

When we describe the kind of input that a primitive requires, we are *not* speaking about the way the input is written when you define the procedure; these rules are described in Chapter 2. If you look up MAKE, you will see that it must have the following form:

MAKE *name object*

This uses two input words: *name* and *object*. *Name* means that the first input must be a word (we call a word a name if it is used to identify a procedure, a variable or a property) and *object* is an abbreviation for a Logo object (a word, a number or a list).

On the following pages are the words that are used to describe the inputs to SmartLOGO primitives. Beside each word is a description of what the real input must be. These definitions apply only to the inputs of SmartLOGO primitives.

Input Words

Type of Input	Real Input
<i>address</i>	An integer from 0 through 65535.
<i>byte</i>	An integer from 0 through 255.
<i>character</i>	A letter of the alphabet, number, or punctuation mark.
<i>colornumber</i>	An integer from 0 through 15. SmartLOGO accepts bigger numbers and divides them by 16 using the remainder as the input number (i.e. $16 = 0$, $17 = 1$, $100 = 4$).

<i>condnumber</i>	An integer from 0 through 5.
<i>degrees</i>	Degrees of an angle. A real number between - 32767 and 32767.
<i>duration</i>	An integer from 0 through 255.
<i>filename</i>	A single <i>word</i> of 10 characters or less, used to name a file.
<i>freq</i>	An integer from 128 through 9999.
<i>instructionlist</i>	A <i>list</i> of procedures that SmartLOGO can execute.
<i>joystick</i>	An integer, 0 or 1.
<i>line</i>	An integer from 0 through 23.
<i>list</i>	Information enclosed in [] brackets.
<i>name</i>	A word naming a procedure, a variable or a property.
<i>namelist</i>	A <i>list</i> of words that name procedures or variables or properties.
<i>newname</i>	A word used for naming a procedure.
<i>newshapenumber</i>	An integer from 0 through 59.
<i>number, a, b, x, y</i>	A number. (The type of number will be specified.)
<i>object</i>	A Logo object (a <i>word</i> , a <i>list</i> or a number).
<i>paddlenumber</i>	An integer, 0 or 1.
<i>position</i>	A <i>list</i> of two numbers giving the coordinates of the turtle or the cursor.
<i>pred</i>	A predicate, which is an operation that outputs either the word TRUE or the word FALSE.

<i>prop</i>	A word naming a property.
<i>ratio</i>	A number from – 32767 to 32767.
<i>shapenumber</i>	An integer from 0 through 59.
<i>shapespec</i>	A <i>list</i> of 32 numbers representing the shape grid.
<i>speed</i>	A number from – 128 through 128.
<i>startvol</i>	An integer from 0 through 15.
<i>steplength</i>	An integer from 0 through 15.
<i>steps</i>	An integer from 0 through 15.
<i>stepvol</i>	An integer from – 7 through 7.
<i>turtlenumber</i>	An integer from 0 through 29.
<i>turtlenumberlist</i>	A <i>list</i> of integers from 0 through 29.
<i>type</i>	An integer from 0 through 7.
<i>voice</i>	An integer from 0 through 2.
<i>volume</i>	An integer from 0 through 15.
<i>word</i>	A sequence of characters (not including spaces).

Special SmartLOGO Considerations

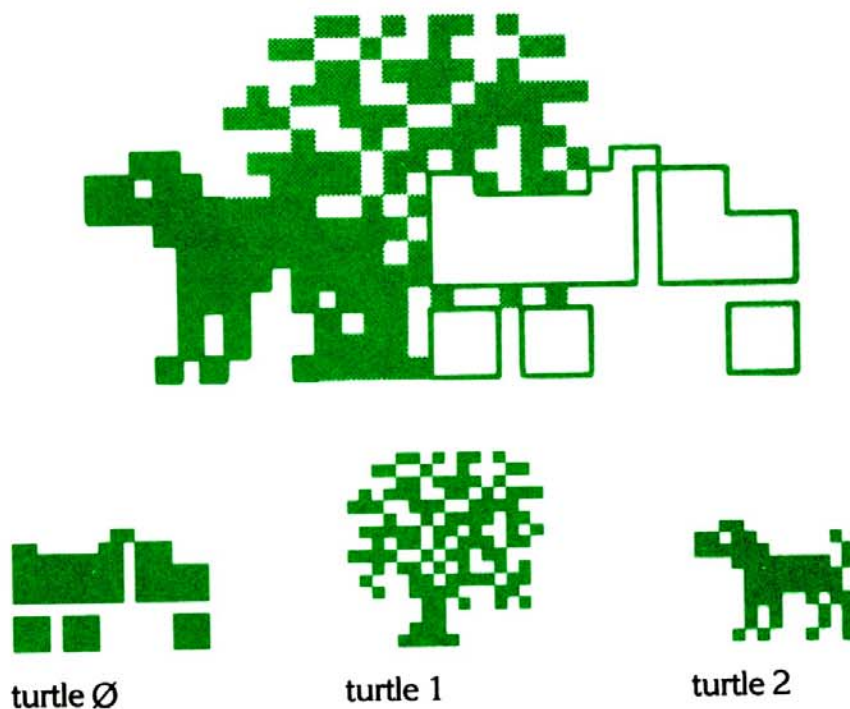
Order of Precedence for Turtles

ADAM™'s SmartLOGO has 30 turtles and there is an order of precedence among them. Lower numbered turtles always take precedence over higher numbered turtles when

there is a question of who should be displayed on the screen. If two or more turtles overlap, the lower numbered turtle is the one you see.

To understand this, think of 30 transparent planes, numbered 0 through 29, making up your display screen. Each turtle moves on the plane having its number — turtle 7 moves on plane 7 and so on. Plane 0 is at the front of the screen, plane 14 in the middle and plane 29 at the rear of the screen. So, the visible turtle on plane 5 will appear in front of a turtle on plane 6 (or any other higher numbered plane), but the visible turtle on plane 3 will appear in front of turtle 5 when they overlap.

This feature can be used to enhance your dynamic graphics. A truck (a turtle wearing a truck shape) can move in front of a tree (a turtle with a tree shape) if the truck-turtle has a lower number than the tree-turtle. Similarly, a dog can pass behind the tree if the turtle using the dog shape has a higher number than the turtle using the tree shape.



The Four-Turtles-in-a-Line Rule

There is an important rule that you should be aware of. It is called the four-turtles-in-a-line rule. If there are more than four turtles on the same horizontal line, the ones with the highest numbers will disappear. Only four turtles can be visible on one horizontal line at a time. Even if the four turtles are hidden, another turtle with a higher number will disappear. Turtles that disappear for this reason don't go away, they become visible again when one of the other turtles moves.

If you find turtles appearing and disappearing in strange ways, check to see if you have four or more turtles on a horizontal line.

When SmartLOGO starts, all 30 turtles are located at the center of the screen. If you want turtle 4 to be visible, a turtle will have to move — either the one you want to see or one of the lower numbered turtles.

Try the following example to see the effects of the four-turtles-in-a-line rule:

```
?TELL ALL
?CS HT PU
?TELL [2 3 4 5]
?ST
?EACH [SETCOLOR 7 FD 50 SETX →
      20 * WHO]
?TELL 0
?ST SETSP 25
?SETSP 0
?SETY 50
?TELL 5
?SETSP 25
```

When you have finished experimenting, the following commands clear the screen and leave only the first turtle (turtle 0) visible.

```
? TELL ALL
? CS
? HT
? SETSH 36
? TELL 0
? ST
```

Text and Graphics on the Screen

Often if the screen is full of graphics, it may be hard to read the text. Use `CLEARGRAPHICS (CG)` so the graphics will be cleared, but the text will remain unaltered. If the text is unnecessary, `CLEARTEXT (CT)` leaves the graphics but clears the text.

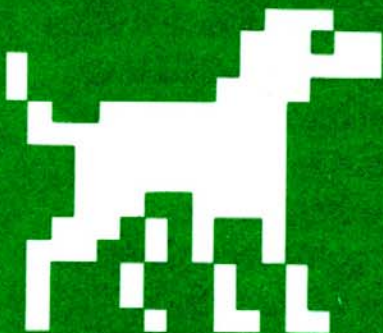
Graphics and text are separate. Clearing one does not affect the other.

`CLEARSCREEN (CS)` clears both graphics and text, which gives you a totally clear screen with the turtle back home. `CLEAN` does the same as `CLEARSCREEN` to the graphics screen, except that the turtle's position and speed are unchanged. `CLEARSCREEN`, `CLEARGRAPHICS`, and `CLEAN` all clean the `FILLED`, `STAMPed` and `SHADEd` areas, but do not change the background color. Note that only those turtles in the current `WHO` list will be affected by `CLEARSCREEN` or `CLEARGRAPHICS`. `TELL ALL CLEARSCREEN` will always clean the screen, stop all the turtles and send them to the center, if there are no demons (use `ERDS` to erase demons).

`SETBG` changes the background color (see Chapter 5 for a listing of colors). The text is always white (15). If the background is set to the same color as text (15), you will not be able to read the text. Use the primitive `CHANGE.COLOR` to make the text readable. For instance `CHANGE.COLOR 15 1` changes everything in color 15 (except the background) to color 1 (black). Then you can read the text.

It is important to keep these facts and primitives in mind so that you know how to clear graphics, text or both.

2.



Logo is a powerful and flexible programming language made up of *building blocks* called *procedures*. Some procedures are already built into the SmartLOGO system; these are called *primitives*. Others are defined by you. Other than the fact that primitives are built in, there is no difference between primitives and the procedures you define.

Procedures can construct, modify and run other procedures. They obey the rules of Logo grammar. The following sections briefly describe these rules.

Procedures

The name of a procedure can be anything you like except a name that SmartLOGO already uses for a primitive. A meaningful name is best. A name can be as long you want, but it must be only one word (no spaces). The word can contain letters, numbers or symbols.

Here is a definition of a procedure called
WELCOME:

```
TO WELCOME           title line
PRINT "HI
END
```

The title line always begins with TO followed by the name of the procedure. The last line contains only the word END. For WELCOME, the main body is a request to run the primitive PRINT.

There are two ways of defining a procedure;

- by using the SmartLOGO Editor, with TO or EDIT.
- by using the primitive DEFINE.

Once a procedure is defined, one way of executing it is to type its name at top level (top level, represented by the question mark (?), is when commands are immediately executed):

```
?WELCOME           procedure call
HI                 result
```

Another way is to call the procedure inside the definition of another procedure. Suppose WARMWELCOME is defined like this:

```
TO WARMWELCOME
WELCOME
WELCOME
WELCOME
WELCOME
WELCOME
END
```

When it's called, WARMWELCOME executes WELCOME 5 times.

```
?WARMWELCOME
```

```
HI  
HI  
HI  
HI  
HI
```

WARMWELCOME is the *superprocedure* that contains the *subprocedure* WELCOME. Using superprocedures and subprocedures, you can build structures of great complexity. These structures contain levels. WARMWELCOME is the highest level, and its subprocedure is in a level below. These levels make up a program hierarchy.

A procedure can also be a subprocedure of itself. This is called *recursion*. You'll find many examples of this powerful Logo feature throughout this manual.

If you ask SmartLOGO to run an undefined procedure, an error message appears.

```
?TALK
```

```
I DON'T KNOW HOW TO TALK
```

Punctuation and Inputs to Procedures

SmartLOGO interprets every word as a request to run a procedure. You must use special characters to indicate when this is not the case.

A word beginning with a quotation mark — for example, "HI — tells SmartLOGO that the word must be treated literally, not as a procedure call. Here, "HI is an input to the procedure PRINT.

```
?PRINT "HI  
HI
```

Numbers are like literal words, but they don't need quotation marks.

```
?PRINT 5
5
```

A sequence of words surrounded by square brackets indicates a *list*. Lists can be inputs to procedures.

```
?PRINT [ARE WE HAVING FUN?]
ARE WE HAVING FUN?
```

The list [ARE WE HAVING FUN?] is a list of SmartLOGO words; SmartLOGO does not try to execute them. The following example illustrates this more clearly.

```
?PRINT [2+2]
2+2
```

Without the brackets, SmartLOGO will attempt to execute the sequence of words.

```
?PRINT 2+2
4
```

or

```
?PRINT ARE WE HAVING FUN?
I DON'T KNOW HOW TO ARE
```

Your procedures can also have inputs. For example:

```
TO GREET :PERSON title line
PR "HI
PR :PERSON
PR [HAVE A NICE DAY]
END
```

A word beginning with a colon (:) tells SmartLOGO that the word is a variable. Variables that hold the inputs to procedures are written on the title line after the name of the procedure. PERSON is a variable whose value is determined when GREET is called. The main body of GREET contains three calls of the procedure PRINT (PR is the short form of PRINT). The second of these calls uses the current value of PERSON.

Here's an example of a request to execute GREET at top level.

```
?GREET "SHARNEE
HI
SHARNEE
HAVE A NICE DAY
```

In this case, the input is the word SHARNEE; SmartLOGO makes this the value of PERSON when it executes GREET.

Commands and Operations

There are two kinds of procedures in SmartLOGO: operations and commands. Operations (referred to as reporters in *Turtle Talk*) output a value to another procedure; commands (such as PRINT) do not.

The primitive SUM is an operation that outputs the sum of two numeric inputs. In this example, the output of SUM is sent to the primitive command PRINT:

```
?PRINT SUM 31 28
59
```

Since an operation can be used only as an input to another procedure, every SmartLOGO line must begin with a command. Otherwise, you get an error message. For example:

```
?SUM 31 28
YOU DON'T SAY WHAT TO DO WITH
H 59
```

Your procedures can be commands or operations. The procedure GREET is a command. To construct operations, you must use the primitive OUTPUT. The procedure FLIP, for example, is an operation:

```
TO FLIP
IF 0 = RANDOM 2 [OUTPUT "HEADS]
OUTPUT "TAILS
END
```

FLIP outputs the literal word HEADS if RANDOM 2 outputs 0, or TAILS if RANDOM 2 outputs 1. You can pass the output from FLIP to PRINT:

```
?PR FLIP
HEADS
```

Variables

You can think of a *variable* as a container with a name on the outside and an *object* (a word, list, or number) inside. The object is the value of the variable. A colon in front of a word tells SmartLOGO it is a variable name and makes its current value available to a procedure. For example:

```
?PRINT :JOHN
```

tells SmartLOGO to look for a container named JOHN. If it finds one, it looks inside the container and makes whatever

it finds available to PRINT. PRINT then displays the contents of JOHN on the screen.

If no variable JOHN exists, SmartLOGO prints the error message:

```
JOHN HAS NO VALUE
```

You can assign a value to a variable in two ways:

- by defining a procedure with inputs (see the previous section “Punctuation and Inputs to Procedures”) and then calling the procedure with specified values.
- by using one of the naming primitives MAKE and NAME.

MAKE requires two inputs: a word and a value.

```
?MAKE "INTEGER 25
?PRINT :INTEGER
25
```

In this case, the value is a number (25). However, it can be a word or a list as well. Consider this example:

```
?MAKE "NUMBER "INTEGER
```

Here, MAKE has two quoted words as inputs. It puts the word INTEGER inside the container NUMBER. The contents of the variable name INTEGER from the previous example are undisturbed. So,

```
?PRINT :NUMBER
INTEGER
?PRINT :INTEGER
25
```

NAME has the same function as MAKE, but the order of inputs is reversed.

Global and Local Variables

Variables created with `MAKE` or `NAME` remain in the workspace until erased. These variables are called *global variables*. There are also variables that remain in the workspace only as long as a procedure is being executed. These variables are called *local variables*. Variables that are defined as inputs to procedures are local variables.

The procedure `GREET` can be modified to print the date.

```
TO GREET :PERSON
PR :DATE
PR "HI
PR :PERSON
PR [HAVE A NICE DAY]
END
```

`DATE` does not appear on the title line of `GREET`, so it is a global variable. You can define the value of `DATE` at top level.

```
?MAKE "DATE [MARCH 14 1984]
?GREET "BRIAN
MARCH 14 1984
HI
BRIAN
HAVE A NICE DAY
```

The variable `PERSON` is not global. After `GREET` stops executing, `PERSON` no longer has any value. (But `DATE` is still in the workspace.)

You could also use `MAKE` to define `DATE` inside the procedure `GREET`. It would still remain as a global variable after `GREET` executes.

Understanding a Logo Line

A SmartLOGO line can be longer than the line you see on the screen. It can contain 2048 characters. For example:

```
?MAKE "MANYNAMES [MIKE BARBA →  
RA ERIC JUDY SHARNEE EFFIE]
```

The arrow (→) indicates that the next screen line is a continuation of the previous screen line. You end a SmartLOGO line by pressing **RETURN**.

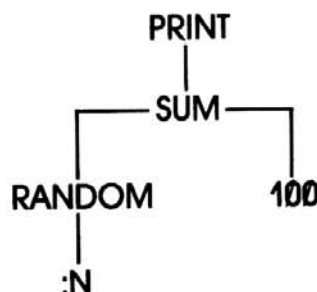
Here are some guidelines to help you interpret a complex Logo line.

- The first word of a Logo line must always be a command.
- An operation is always the input to another procedure.
- Every input to a procedure must be accounted for.
- When the inputs to a command have been accounted for, the next procedure must be another command.

Here is an example of a complex Logo line:

```
?PRINT SUM RANDOM :N 100
```

PRINT is a command with one input, in this case the output of SUM. SUM requires two inputs. The first is the output of RANDOM, which itself requires one input (the current value of N). The second input to SUM is 100.



If N is assigned the value 10,

```
?MAKE "N 10
```

then the line will print a number between 100 and 109:

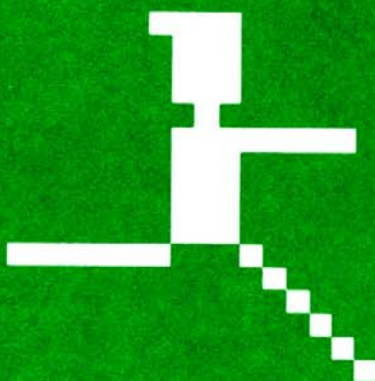
```
?PRINT SUM RANDOM :N 100  
101
```

C

C

C

3



This chapter deals with the SmartLOGO Editor and those primitives which allow a user to create new procedures — to write a unique SmartLOGO vocabulary.

The SmartLOGO Editor is used to define procedures, but other primitives allow for the defining of procedures within other procedures.

The SmartLOGO Editor is used to create and edit procedures or variables, which are then entered into the workspace. A procedure or variable erased in the Editor is not erased from the workspace. An entire file can be edited in the Editor. The Editor is called into action with any of the following primitives: TO, EDIT, EDNS or EDFILE. See Chapter 13 for more information on variables and EDNS. See Chapter 19 for more information on files and EDFILE.

SmartLOGO also has a Shape Editor which is used to create and edit shapes which the turtle can wear. The operation of the Shape Editor is described in Chapter 6.

SmartLOGO Editor

How the Editor Works

The word **TO** calls the Editor. For example:

```
?TO GREET
TO GREET :NAME
PR "HI
PR :NAME
PR [HAVE A NICE DAY]
END
```

The prompt "?" disappears when you are in the Editor.

TO is the command which starts the Editor. Its input is the name of the procedure to be defined. If a procedure is already defined, its definition is brought into the Editor where you can modify it. If a procedure name is undefined, the Editor shows only the title line and **END**.

```
?TO SQUARE
TO SQUARE ■
END
```

The input to **EDIT** (or **ED** for short) can be a single name or a list of names. If the input is a single name it must have a quotation mark in front of it.

```
?ED "POLY
TO POLY :SIDE :ANGLE
FD :SIDE
RT :ANGLE
POLY :SIDE :ANGLE
END
```

When a list of procedure names is used as the input to **EDIT**, all the procedures in the list will be brought into the Editor.

```
?ED [HOUSE ROOF DOOR]
```

TO and EDIT can be used without an input. In this case, you will enter the Editor with no procedure name. You can start defining a procedure by typing a procedure name.

There is no prompt character, but the *cursor* shows you where you are typing.

You can move the *cursor* anywhere in the text using the arrow keys. You can also delete characters using the appropriate keys described in this section.

Pressing the **RETURN** key marks the end of a SmartLOGO line. A SmartLOGO line can contain 2048 characters. When you press the **RETURN** key, the cursor and any text that comes after it, moves to the beginning of the next line and is ready for you to continue typing.

You can have more characters in a line of text than fit across the screen. Simply continue typing when you get to the end of the screen line. An arrow (→) will appear at the end of the screen line and the *cursor* will move to the next screen line. SmartLOGO does the same thing outside of the Editor.

This is how a long line would appear on the screen:

```
TO PRINTMESSAGE :PERSON  
PRINT SE :PERSON [ , I AM GOI→  
NG TO TYPE A VERY LONG MESSA→  
GE FOR YOU]  
END
```













The Editor has a line buffer called the delete buffer. The delete buffer can hold 256 characters. The **CLEAR** key deletes a line of text and puts it in this buffer. The **MOVE/COPY** key reinserts this line of text later at the place marked by the cursor. The text remains in the buffer until the **CLEAR** key is pressed again.

Editing Actions





When you are in the Editor, you can use the following editing keys:

*The star represents editing keys that work both inside and outside the SmartLOGO Editor. They can be used at top level to make changes on the line presently being typed.

Cursor Motion

- | | |
|---|--|
| *  | Moves the cursor right one space. |
| *  | Moves the cursor left one space. |
|  | Moves the cursor down to the next line. |
|  | Moves the cursor up to the previous line. |
| *   | Moves the cursor to the beginning of the current line. |
| *   | Moves the cursor to the end of the current line. |
|   | Moves the cursor to the beginning of the edit buffer. |
|   | Moves the cursor to the end of the edit buffer. |

Inserting and Deleting

- | | |
|---|--|
|  |  creates a new line by moving the cursor and the rest of the text line to the beginning of the new screen line. |
|  | Opens a new line at the position of the cursor but does not move the cursor. |
| *  | Erases the character to the left of the cursor. |

-
- | | |
|--------------------|--|
| * DELETE | Erases the character under the cursor. |
| * CLEAR | Deletes text from the cursor position to the end of the current line. This text is placed in the delete buffer. The delete buffer can hold 256 characters. |
| * MOVE/COPY | Inserts a copy of the text that is in the delete buffer at the current cursor position. |

Exiting the Editor

Smart Key **VI** Smart Key **VI** is the standard way to exit the Editor. It saves all the changes that were made.

The Editor is a large buffer which holds a block of text as you work on it. When you exit the Editor by pressing Smart Key **VI**, SmartLOGO reads each line in the edit buffer as though you had typed it outside the Editor. If there are SmartLOGO instructions in the edit buffer that are not contained in the procedure definition (within **TO ... END**), SmartLOGO carries them out just as if you had typed them in at top level.

You can define more than one procedure while in the Editor, as long as each procedure is terminated by **END**. If you forgot to type **END** at the end of the last definition, SmartLOGO inserts **END** for you.

ESCAPE/WP

Cancels editing. Use it if you don't like the changes you are making, or if you decide not to make changes. If you were defining a procedure, the procedure definition will be the same as before you started editing.

EDIT, ED

command

EDIT

EDIT *name*

EDIT *namelist*

Starts up the SmartLOGO Editor. If an input is given, the Editor starts up with the definition(s) of the given procedure or procedures in the edit buffer. The input to EDIT can be a list of procedure names instead of a single name. In this case, all the procedure definitions will be brought into the Editor.

END

special word

END

END is a special word that tells SmartLOGO that you are finished defining the procedure. It must be on a line by itself. END must be used to separate procedures when defining several procedures in the SmartLOGO Editor.

TO

command

TO *name*

Starts up the Editor. If the procedure *name* has not been previously defined, the edit buffer contains only the title line: TO *name* and END. If no input is given, the edit buffer has the title line TO with no name and END.

Press the Smart Key **VI** to complete the definition and exit the Editor. SmartLOGO reads every line from the edit buffer as though you had typed it outside the Editor. The lines in the procedure are not executed.

Creating Procedures Outside the Editor

COPYDEF

command

COPYDEF *name newname*

Copies the instructions that make up *name* into *newname*. *Newname* may be defined or undefined. If defined, its contents will be replaced.

Example:

The definition of HI is replaced by COPYDEF.

```
TO HI
PR [YOU SAY HELLO?]
END
```

```
TO BYE
PR [I SAY GOODBYE]
END
```

```
?HI
YOU SAY HELLO?
?COPYDEF "BYE "HI
?HI
I SAY GOODBYE
```

DEFINE

command

DEFINE *name list*

Makes *list* the definition of the procedure *name*. The first element of *list* is a list of the inputs to the procedure. If the

procedure *name* has no inputs, the first element of *list* must be an empty list ([]). Each following element is a list of instructions, consisting of one line of the procedure definition. *List* does not contain END, since END is not part of the procedure definition. The second input to DEFINE has the same form as the output of TEXT. DEFINE is used to make procedures or change procedures without using the Editor. See TEXT.

Example:

```
?DEFINE "SQUARE [[:SIDE] [REPEAT 4 [FD :SIDE RT 90]]]
```

defines the same procedure as

```
TO SQUARE :SIDE  
  REPEAT 4 [FD :SIDE RT 90]  
END
```

TEXT

operation

TEXT *name*

Outputs the list of instructions that define the procedure *name*. TEXT outputs the same list format that DEFINE accepts as its input. TEXT and DEFINE are a useful pair for copying or modifying procedures within programs.

Example:

```
?PRINT TEXT "SQUARE  
[:SIDE] [REPEAT 4 [FD :SIDE RT 90]]  
?DEFINE "BOX TEXT "SQUARE  
?PR TEXT "BOX  
[:SIDE] [REPEAT 4 [FD :SIDE RT 90]]
```

DEFINEDP and PRIMITIVEP allow you to check if a procedure or primitive is defined.

DEFINEDP

operation

DEFINEDP *name*

Outputs TRUE if *name* is a defined procedure currently in the workspace, otherwise FALSE.

PRIMITIVEP

operation

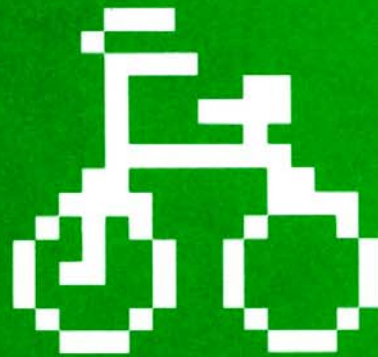
PRIMITIVEP *name*

Outputs TRUE if *name* is a primitive, otherwise FALSE.

Example:

```
?PR PRIMITIVEP "FORWARD  
TRUE
```

4.



Each of the 30 SmartLOGO turtles has a pen with which it draws. The pen's position is indicated by a small dot in the turtle's back.

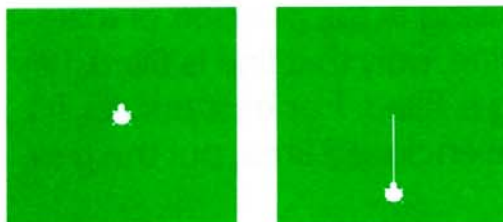
This chapter describes instructions related to the creation of graphic designs on the screen. The turtle can be moved in relation to where it is or in relation to the screen coordinates. It's important to be aware of the state of the turtle's pen when using the following primitives. Even when a turtle is hidden, its pen will draw, fill, and even stamp the invisible shape on the screen. For information on the colors in which the turtle's pen can draw, see Chapter 5.

BACK, BK

command

BACK *number*

Moves the turtle *number* steps back. Its heading does not change.



DISTANCE

operation

DISTANCE position

Outputs the distance in turtle steps between the turtle and *position*.

Examples:

```
?CS
?FD 50
?PR DISTANCE [0 0]
50
```

The following example uses one turtle's position as the input to DISTANCE. The output is the distance between that turtle and the current turtle.

```
?TELL 0 BK 40
?ASK 1 [ST RT 45 FD 50]
?PR DISTANCE ASK 1 [POS]
83.237169
```

DOT

command

DOT position

Puts a dot in the turtle's pen color at the specified position. The turtle does not move.

Example:

DOT [100 0] puts a dot halfway down the right edge of the screen.

FILL

command

FILL

Fills the screen, or an enclosed area of the screen, in the turtle's pen color. FILL starts drawing at the position of the pen; if the pen is over a drawn line, only that line is filled. Of course, if the pen is up, nothing is filled. For best results, lift the pen, put the turtle inside the enclosed area, put the pen down and fill.

Examples:

The following instructions draw a square, move the turtle inside the square with its pen up, put the pen down and then fill the square. The pen color is changed and the square is filled again.

```
?REPEAT 4 [FD 50 RT 90]  
?PU  
?RT 45 FD 10  
?SETPC 13 PD  
?FILL  
?SETPC 15  
?FILL
```

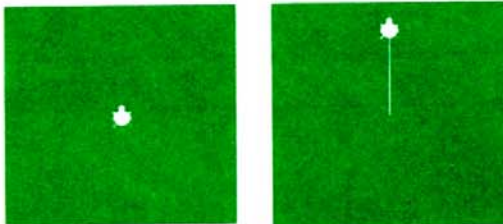
FORWARD, FD

command

FORWARD *number*

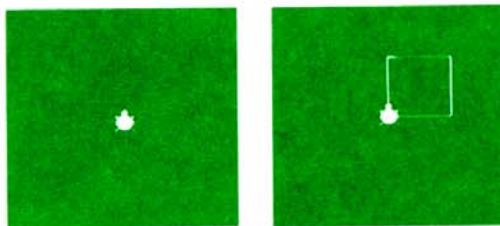
Moves the turtle forward *number* steps in the direction in which it is heading.

Examples:



FD 70

```
TO SQUARE :SIDE  
REPEAT 4 [FD :SIDE RT 90]  
END
```



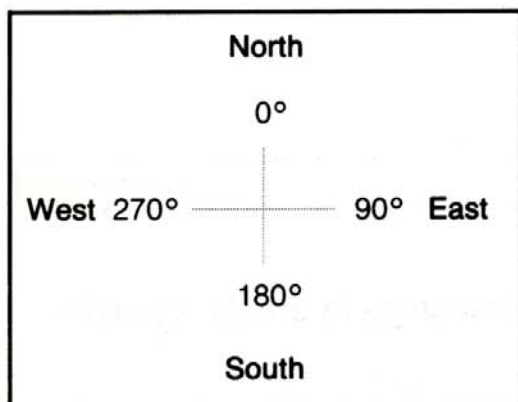
SQUARE 30

HEADING

operation

HEADING

Outputs the turtle's heading, a number greater than or equal to 0 and less than 360. SmartLOGO follows the compass system in which north (up) is a heading of 0 degrees, east (right) 90 degrees, south (down) 180 degrees and west (left) 270 degrees. When you start SmartLOGO, or use CLEARSCREEN (CS) or CLEARGRAPHICS (CG) or HOME the turtle has a heading of 0. See SETHEADING.

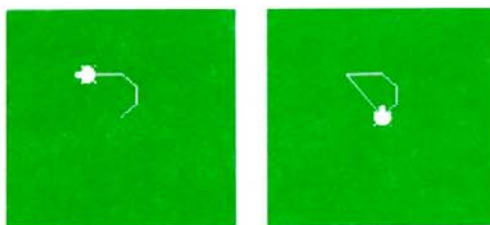


HOME

command

HOME

Moves the turtle to the center of the screen and sets its heading and speed to 0. This command is equivalent to SETSPEED 0 SETPOS [0 0] SETHEADING 0. If the turtle's pen is down, the turtle draws a line from its current position to HOME.



HOME

LEFT, LT

command

LEFT *degrees*

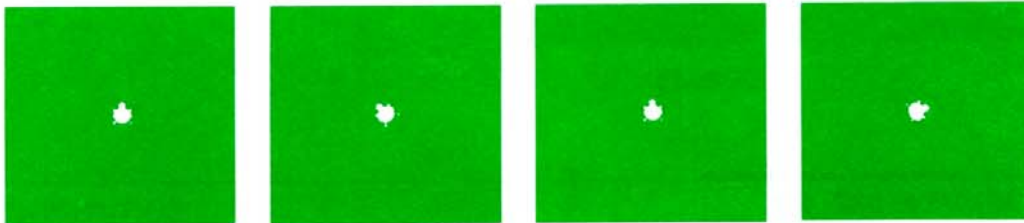
Turns the turtle left (counterclockwise) the specified number of *degrees*. See RIGHT.

Examples:

LT 45 turns the turtle 45 degrees left.

LT 380 turns the turtle 380 degrees left, which is equivalent to 20 degrees left.

LT -70 turns the turtle 70 degrees right.



PEN

command

PEN

Outputs the state of the turtle's pen in a two-element list.

The first member of the list is the pen's drawing state (PENDOWN, PENUP, PENERASE or PENREVERSE), the second is the pen's color number.

Example:

```
? PR PEN  
PENDOWN 15
```

PENDOWN, PD

command

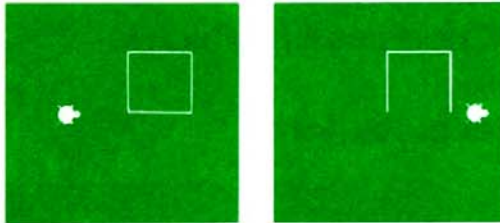
PENDOWN

Puts the turtle's pen down. When the turtle moves, it draws a line in the current pen color. Every turtle begins with its pen down.

PENERASE, PEcommand

PENERASE

Puts the turtle's eraser down. When the turtle moves, it will erase any previously drawn lines it passes over. To lift the eraser use PENDOWN or PENUP.



PE FD 85

PENREVERSE, PXcommand

PENREVERSE

Puts the "reversing pen" down. When the turtle moves, it draws where there aren't lines and erases where there are lines. To lift the reversing pen use PENDOWN or PENUP.

Example:

The CLOCK procedure uses PENREVERSE to erase an existing line and draw a new one, creating the illusion that the line is moving, like the sweep hand on a clock. It uses WHEN 0, the timing demon, which executes the list of instructions once every second.

```
TO CLOCK
CS HT
PX FD 50
WHEN 0 [BK 50 RT 6 FD 50]
END

?CLOCK
```

To cancel the demon, use the ERDS (ERase DemonS) command. PENDOWN puts the drawing pen down.

? ERDS PENDOWN

PENUP, PU

command

PENUP

Lifts the pen up. When the turtle moves, it does not draw lines.

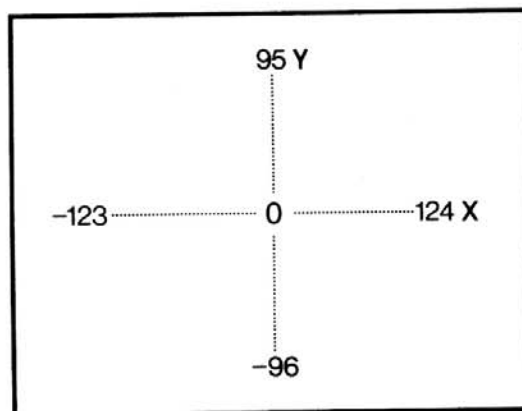
POS

operation

POS

Stands for POSition. Outputs the coordinates of the current position of the turtle in the form of a list $[x\ y]$. When you start SmartLOGO, the turtle is at $[0\ 0]$, the center of the turtle field, also called HOME. See SETPOS for setting the turtle's position.

The screen boundaries are as follows:



RIGHT, RT

command

RIGHT *degrees*

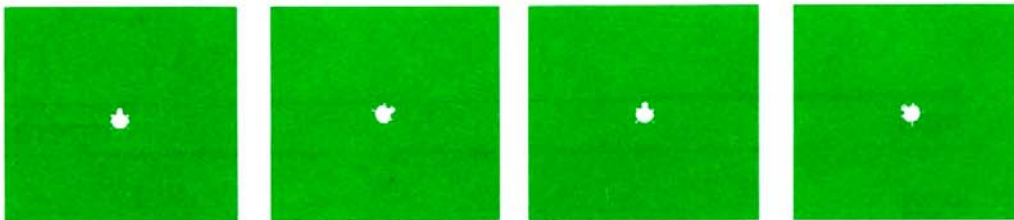
Turns the turtle right (clockwise) the specified number of *degrees*.

Examples:

RT 60 turns the turtle 60 degrees right.

RT 390 turns the turtle 390 degrees right, which is equivalent to 30 degrees right.

RT -50 turns the turtle 50 degrees left.



SETHEADING, SETH

command

SETHEADING *degrees*

Turns the turtle so that it is heading in the direction specified by *degrees*. The turtle's position is not changed. Note that RIGHT and LEFT produce turns relative to the turtle's heading, but SETHEADING sets an absolute heading without reference to its prior heading. Positive numbers are clockwise, negative numbers are counterclockwise from north. See HEADING.

Examples:

?SETH 45

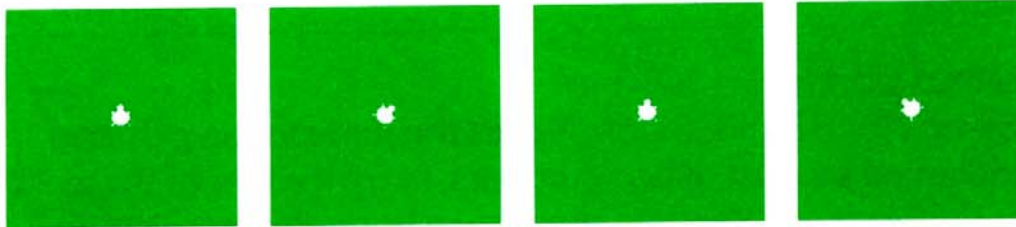
Heads the turtle northeast.

?SETH -45

Heads the turtle northwest.

?PR HEADING

315



SETH 45

SETH -45

SETPEN

command

SETPEN *list*

Sets the turtle's pen to the two elements in *list*. The first element of *list* is the pen's drawing state (PENDOWN, PENUP, PENERASE or PENREVERSE) and the second element is the pen's color. *List* matches the list output by PEN.

Examples:

The following instructions put turtle 0's pen down, set its color to light blue, and then check the pen state with PEN.

```
? TELL 0
? SETPEN [PD 7]
? PR PEN
PENDOWN 7
```

These instructions set turtle 1's pen state to whatever turtle 0's pen state is:

```
? TELL 1
? SETPEN ASK 0 [PEN]
```

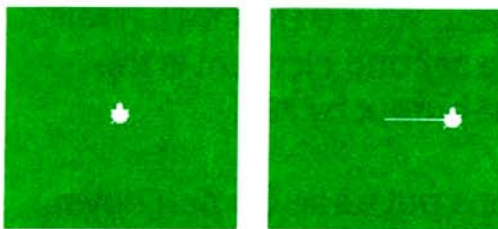
SETPOScommand

SETPOS position

Stands for SET POSition. Moves the turtle to the *position* indicated by a list of two numbers, $[x\ y]$. If either of these numbers is greater than the screen boundaries (see POS) the turtle will wrap around the screen. If the turtle's pen is down, the turtle leaves a trace between its original and new positions.

Example:

SETPOS [100 0] moves the turtle to a point half way down the right edge of the screen.



SETPOS [100 0]

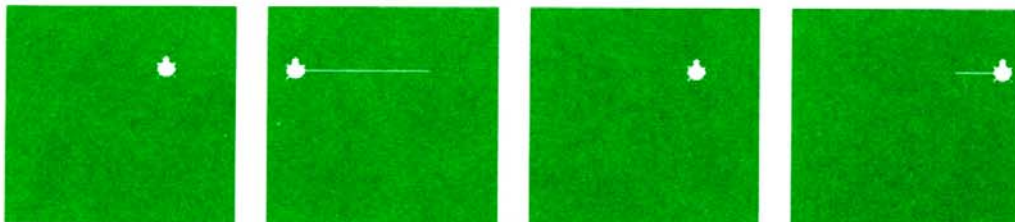
SETXcommand

SETX x

Puts the turtle at a point with x-coordinate x (y-coordinate is unchanged). If the turtle's pen is down, it will leave a horizontal trace.

Examples:

SETX -100 moves the turtle horizontally to the left edge of the screen.



SETX -100

SETX 2 * XCOR

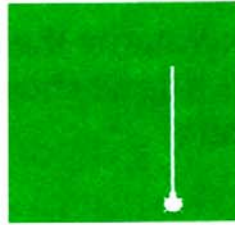
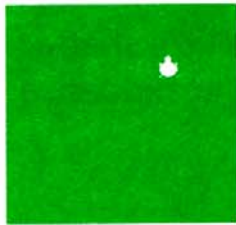
SETYcommand

SETY y

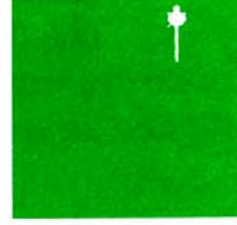
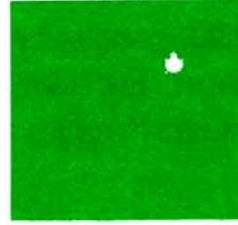
Puts the turtle at a point with y-coordinate y (x-coordinate is unchanged).

Examples:

SETY -85 moves the turtle vertically near the lower edge of the screen.



SETY -85



SETY 2 * YCOR

SHADEcommand

SHADE

Fills the screen, or an enclosed area of the screen, with stamped copies of the turtle's shape. The shading is done by the current pen, in its pen color and its drawing state. If the pen is up, the area will not be shaded. As with FILL, the area that is shaded is the area in which the turtle's pen is located. If the pen is over a drawn line, nothing is shaded. Results are best when the pen is lifted (with PENUP) to enter a closed area, then put down to SHADE.

Example:

The following instructions draw a circle, move the turtle inside, put the pen down and shade the circle with copies of the turtle's shape.

```
? REPEAT 36 [FD 10 RT 10]  
? PU  
? RT 45 FD 10  
? PD  
? SHADE
```

The FOLLOW procedure has turtle 0 moving around the screen, and points turtle 1 TOWARDS its position, once a second.

```
TO FOLLOW
TELL 1 ST
TELL 0 PU ST
RT 80 SETSP 10
WHEN 0 [ASK 1 [SETH TOWARDS →
ASK 0 [POS]]]
END

? FOLLOW
```

To erase the WHEN 0 once-a-second demon, type:

```
? ERDS
```

XCOR	operation
-------------	-----------

XCOR

Outputs the x-coordinate of the current position of the turtle. See YCOR.

YCOR	operation
-------------	-----------

YCOR

Outputs the y-coordinate of the current position of the turtle.

Examples:

```
? CS
? PR YCOR
0

? FD 85
? PR YCOR
85
```


5



The sixteen colors in the table below are used for the color of the turtles, their pens, and the background. When SmartLOGO starts up, the background is light blue (color number 5) and the turtles and their pens are white (color number 15).

Changing the colors can result in stunning visual effects — but you should be aware that if a turtle or a pen is transparent or the same color as the background, it will seem invisible. A full set of operations exists to report on the pencolor, turtle color and background color.

Examples:

If you have light green turtle drawings on the screen, you can change the light green graphics to black:

```
?CHANGE.COLOR 3 1
```

Now change all drawings which are black to gray:

```
?CHANGE.COLOR 1 14
```

COLOR

operation

COLOR

Outputs a number representing the turtle's color. This is an integer (color number) from 0 through 15. See SETCOLOR for changing a turtle's color.

Example:

```
?PR COLOR  
15
```

COLOR.OVER

operation

COLOR.OVER

Outputs the number of the color under the turtle's pen (indicated by the clear spot in the turtle's back). If the turtle's pen is down, this may be the color of the turtle's own lines rather than the color the turtle seems to be over. COLOR.OVER detects drawn lines and background colors, but not other turtles.

Example:

The following procedures sets up a race between 3 turtles. The "finish line" is a square stamped in purple. COLOR.OVER is used to determine when a turtle gets to the purple square.

Table of Colors

Number	Color
0	transparent
1	black
2	medium green
3	light green
4	dark blue
5	light blue
6	red
7	cyan
8	medium red
9	light red
10	yellow
11	light yellow
12	dark green
13	magenta
14	gray
15	white

BACKGROUND, BG

operation

BACKGROUND

Outputs a number representing the color of the background. When SmartLOGO starts, BACKGROUND is color number 5 (light blue). See SETBG for setting the background color. Color 0, the transparent color, is black when used for a background.

CHANGE.COLOR

command

CHANGE.COLOR *colornumber colornumber*

Changes all turtle graphics from the first *colornumber* to the second *colornumber*. It does not affect the background color or the colors of the turtles themselves. Only lines which have previously been drawn are affected.

SETBGcommand

SETBG *colornumber*

Stands for SET BackGround. Sets the background color to the color represented by *colornumber*. If the number is greater than 15, it is set to the remainder of that number divided by 16.

Example:

The following procedure cycles through all the possible background colors.

```
TO CHANGEBG
SETBG 1+BG
WAIT 30
CHANGEBG
END

?CHANGEBG
```

To stop this procedure, press ESCAPE/WP.

SETCOLOR, SETCcommand

SETCOLOR *colornumber*

Sets the color of the turtle to *colornumber*. There are 16 different colors (0 to 15). If the number is greater than 15, the color is set to the remainder of *colornumber* divided by 16.

Example:

The following instructions set each turtle's color to its own number, that is, turtle 1 turns black (color 1) and turtle 20 turns dark blue (the remainder of 20/16 is 4).

```
?TELL ALL CS ST PU
?EACH [SETH 12*WHO]
?EACH [SETC WHO]
?SETSP 10
```

Some turtles seem to be invisible; turtles 0 and 16 are the transparent color, others are the same color as the background.

SETPENCOLOR *colornumber*

Sets the color of the pen to *colornumber*. *Colornumber* is any integer from 0 to 15. SmartLOGO accepts numbers above 15, and sets the pencolor to the remainder of that number divided by 16.

Examples:

```
?SETPC 6
```

```
?PR PC
```

```
6
```

```
?SETPC 22
```

```
?PR PC
```

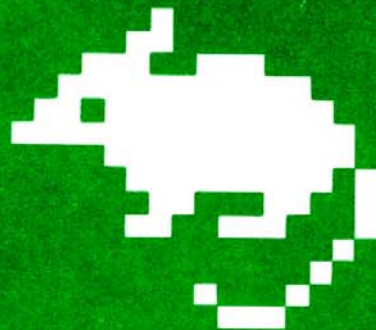
```
6
```

C

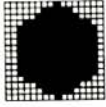

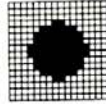
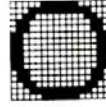

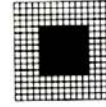
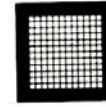
C


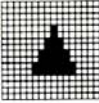
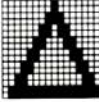
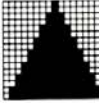


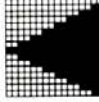
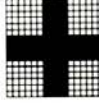
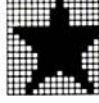
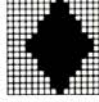
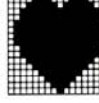
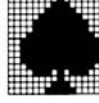
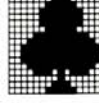
C

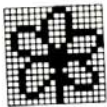
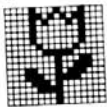
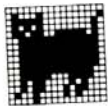

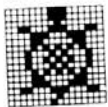
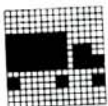
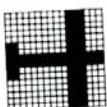

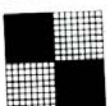
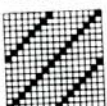
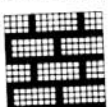
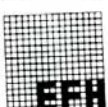

6.



This list shows the 60 available turtle shapes:

Number	Shape	Name
0		hexagon
1		octagon
2		small circle
3		empty circle
4		filled circle
5		small square
6		empty square

Number	Shape	Name
7		filled square
8		small triangle
9		empty triangle
10		filled triangle
11		filled triangle
12		filled triangle
13		filled triangle
14		cross
15		star
16		diamond (cards)
17		heart (cards)
18		spade (cards)
19		club (cards)

Number	Shape	Name
20		flower
21		flower
22		cat
23		dog
24		turtle
25		truck
26		airplane
27		rocket
28		checkered square
29		stripes
30		brick wall
31		EFH
32-35		blank shapes

Number	Shape	Number	Shape
--------	-------	--------	-------

turtle shapes



You can change a turtle's shape to any one of these predefined shapes. When you start SmartLOGO, every turtle's shape is number 36. As it changes heading the shape also changes taking shape 36 through 59.

Any or all of the 60 shapes can be changed using the SmartLOGO Shape Editor. There are four blank shapes (32 to 35) that you can use to define your own shapes, but any of the shapes can be redefined. If you set a turtle's shape to one of the first 36 shapes (0 through 35), the turtle will retain that shape even when the direction of its motion — its heading — changes.

Shapes 36 through 59 are predefined as the standard turtle shape in its various orientations. If the heading is zero, the turtle uses shape 36. If the heading is 90, the turtle uses shape 42 and so on.

This entire set of standard turtle shapes (36 through 59) can be changed into an object having various headings. You could edit shape 36 to be a rocket headed straight up. You would then edit shape 42 so that the rocket would head 90 degrees toward the right, shape 54, 90 degrees to the left and so on. This new set of shapes would replace the turtle shapes so that when you change the turtle's heading it will automatically change the shape to aim in that direction.

To give the turtle a new shape use the command `SETSHAPE`. In order to find out what shape the turtle is using, type `PR SHAPE`.

SmartLOGO Shape Editor

You can create as many shapes as you want using the SmartLOGO Shape Editor. There are 60 possible shapes available at one time. Shapes 36 to 59 are the original turtle shapes. There are also 32 predefined shapes and 4 blank shapes.

ES is the command to start the SmartLOGO Shape Editor. Its input is the shapenumber (0 through 59). These shapes start out with the predefined shapes every time SmartLOGO starts up. ES brings the specified shape into the Editor.

? ES 1

The Shape Editor is a grid made up of 16 boxes by 16 boxes. When you first enter the Editor the cursor, the flashing white square is in the top left box.

You can move the cursor anywhere in the shape. You are able to pass over the boxes without changing them, and can create or change a shape by filling in the boxes or erasing them using the **HOME** key or the **CONTROL**—arrow key combinations. The filled boxes (the black areas) make up the shape.

Moving the Cursor and Changing the Shape

Use the arrow keys to move the cursor around without changing the shape. To change what is under the cursor, press the **HOME** key; a blank spot (white) will become filled (black) and a filled spot will become blank. This is how you define your shape. Remember to position the cursor

before pressing the **HOME** key. These are the keys that can be used in the Shape Editor:

→	Moves the cursor right one space.
←	Moves the cursor left one space.
↑	Moves the cursor up one space.
↓	Moves the cursor down one space.
HOME	Fills an empty space or empties a filled space.
CONTROL →	Fills or empties, then moves right.
CONTROL ←	Fills or empties, then moves left.
CONTROL ↑	Fills or empties, then moves up.
CONTROL ↓	Fills or empties, then moves down.
CLEAR	Empties the shape, leaving a blank shape.
MOVE/COPY	Replaces the present edited shape with the shape that was originally put into the Editor.

Leaving the Shape Editor

To leave the Shape Editor, press either Smart Key **VI** or **ESCAPE/WP**.

Smart Key **VI** exits the Shape Editor saving the changes you have made.

ESCAPE/WP quits the Shape Editor without saving any changes.

COPYSH**command**

COPYSH shapenumber newshapenumber

Copies the shape *shapenumber* into *newshapenumber*.
Shapenumber does not lose its shape.

Example:

?COPYSH 25 0

Shape 0 is now the truck shape.

EDITSHAPE, ES**command**

EDITSHAPE shapenumber

Starts up the SmartLOGO Shape Editor which allows you to make up your own shapes. ES brings the shape corresponding to *shapenumber* into the Editor, *shapenumber* being an integer from 0 to 59.

GETSH**operation**

GETSH shapenumber

Outputs a list of 32 numbers representing the grid of *shapenumber* (an integer from 0 through 59). Normally, one need not be concerned with the representation of the shape as a list of numbers. All that is important is that GETSH and PUTSH operate as a pair of instructions. They can be used to store and retrieve a shape as a list of numbers which can be saved in a file on tape.

When SmartLOGO starts up, the 60 shapes are preset. Any new shapes created using the Shape Editor will be lost when the computer is turned off, unless they have been created as variables with GETSH. They can then be saved on tape with the rest of the workspace.

Example:

?MAKE "ROCKET GETSH 27

?PR :ROCKET

```
0 1 1 1 1 1 7 15 15 31 31 25→
 25 17 17 16 128 192 192 192→
 192 192 240 248 248 252 252
 204 204 196 196 4
```

Your workspace can now be saved, and the name ROCKET with its value will be saved. Upon re loading the file, the name and its value would enter the workspace. Then, the command:

?PUTSH 27 :ROCKET

would load the list of numbers back into shape 27.

For those who are truly interested, here is how the number list is generated:

column position column value	A								B							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
	128	64	32	16	8	4	2	1	128	64	32	16	8	4	2	1
0																
1																
2																
3																
4																
5																
6																
7																
8																
9																
10																
11																
12																
13																
14																
15																

There are two half-grids of 8 columns by 16 rows, placed side by side to make a 16×16 grid. Each column has a number, and a value associated with it (see diagram).

The value is the number as a power of 2; 2 to the power 0 is 1, 2 to the power 2 is 4, and 2 to the power 7 is 128. If filled, each square in a row assumes the value of the column it is in, if empty its value is 0.

For each row of each half-grid, the values of each square are added. There is a different total for each combination of filled and empty squares: if only 0 and 1 were filled, the sum would be 3 (1 + 2, all others are 0); if only 5 and 7, the total would be 160 (32 + 128). An empty row gives a sum of 0.

These sums are the numbers output by GETSH. They are stored in the list according to row number. Row 0's A sum is the 1st element in the list, and its B sum is the 17th. Row 1's A sum is the 2nd element, and its B sum is the 18th, and so on through the 16 rows. The 16 rows are thus represented by 32 numbers. See the previous example.

HT	command
-----------	---------

HT

Stands for Hide Turtle. Makes the turtle invisible.

PUTSH	command
--------------	---------

PUTSH *shapenumber shapespec*

Gives *shapenumber* the specified *shapespec* as its shape. The output of GETSH can be the input of *shapespec* in PUTSH. PUTSH also allows you to define shapes within a program, as an alternative to using the Shape Editor. See GETSH.

SETSHAPE, SETSH	command
------------------------	---------

SETSHAPE *shapenumber*

Sets the shape of the current turtle to the shape specified by *shapenumber*.

Shapes are numbered 0 through 59, with shapes 36 through 59 being the standard turtle shape in its various headings. See the shape list at the beginning of this chapter.

Examples:

To make the turtle look like a rocket:

```
?SETSH 27
```

And to return it to its turtle shape:

```
?SETSH 36
```

SHAPE

operation

SHAPE

Outputs the number representing the shape of the turtle. Do not confuse a turtle's shape number with its turtle number. When SmartLOGO starts, SHAPE is 36. See SETSHAPE.

Examples:

```
?TELL 3  
?SETSH 25  
?PR SHAPE  
25  
?PR WHO  
3
```

To find the shapes of several turtles:

```
?TELL [0 1 2 3 4]  
?EACH [PR SHAPE]
```

SHOWNP

operation

SHOWNP

Outputs TRUE if the turtle is not hidden, otherwise FALSE. The turtle may not be visible, and yet be "showing"; it may be

the same color as the background, it may be using the transparent color or it may be off the screen in WINDOW mode.

SNAP**command**

SNAP

Copies the pattern of lines under the turtle, into its current shape number. The result is that the turtle's shape becomes a copy of the pattern it's over. Any other turtles using the same shape number will also take the new shape.

Example:

```
?SETSH 1
?CS PD
?RT 45
?REPEAT 4 [FD 10 BK 10 RT 90 →
]
?SNAP
?CS
```

Note that if you SNAP while the turtle is wearing the standard turtle shape (from 36 to 59) that shape will be replaced until you restart SmartLOGO.

ST**command**

ST

Stands for Show Turtle. Makes the turtle visible. See SHOWNP for some exceptions.

Examples:

```
?HT
?ST
?HT
?SETSH 21
?ST
```

○

○

○

7



This chapter describes the SmartLOGO primitives which give the turtles motion, making animated scenes of all kinds possible.

A turtle moving at a speed is different from a turtle which is obeying a FORWARD command. When a turtle goes FORWARD, the computer devotes itself to the action. The flashing cursor goes away until the turtle has stopped drawing.

Speed is different — when all 30 turtles are in motion following a SETSPEED command, the SmartLOGO prompt and cursor are there, indicating that the computer is ready to accept further instructions.

FREEZEcommand

FREEZE

Halts all turtles and prevents any further movement, with **SPEED**, until a **THAW** command is given. **THAW** restores all former speeds, unless some have been changed in the meantime. **FREEZE** acts on all turtles, regardless of whether they are active.

Example:

```
? TELL [0 1 2 3]
? ST
? SETSP 1
? TELL 1
? RT 90
? FREEZE
```

All turtles stop, even though turtle 1 is the only current turtle.

```
? SETSP 50
? THAW
```

They all begin moving again, turtle 1 at a higher speed.

SETSPEED, SETSPcommand

SETSPEED *speed*

Sets the current turtle's speed to *speed* (without altering its heading). If *speed* is greater than 0, the turtle moves forward. If *speed* is less than 0, the turtle moves backwards. If *speed* is equal to 0, the turtle stops moving. It is an error if *speed* is greater than 128, or less than -128. Note that **SETSP**'s input does not need to be an integer.

Example:

The FLY.AND.DRIVE procedure sets two turtles moving on the screen, one wearing an airplane shape and the other wearing a truck shape.

```
TO FLY.AND.DRIVE
  TELL 0
  SETSH 26 SETC 9
  SETH 90 PU SETY 60
  ST SETSP 10
  TELL 1
  SETSH 25 SETC 14
  SETH 90
  PU ST SETSP 5
  END
```

SETXVEL

command

SETXVEL *speed*

Sets the horizontal component of the current turtle's velocity to *speed* without changing its vertical component.

SETXVEL, like its counterpart SETYVEL, can affect the turtle's speed and heading.

Example:

BOUNCE causes turtle 0 to bounce back and forth between turtles 1 and 2.

```
TO BOUNCE
  TELL 1 RT 90 FD 50 ST
  TELL 2 LT 90 FD 50 ST
  ON.TOUCH 0 1 [SETXVEL -20]
  ON.TOUCH 0 2 [SETXVEL 20]
  TELL 0 ST RT 90 SETSP 20
  END
```

? BOUNCE

To cancel the ON.TOUCH demons, type ERDS.

SETYVEL	command
----------------	---------

SETYVEL *speed*

Sets the vertical component of the current turtle's velocity to *speed* without changing its horizontal component. See SETXVEL.

SPEED	operation
--------------	-----------

SPEED

Outputs the turtle's speed. Note that speed is defined as , turtle steps per 16/60th's of a second.

THAW	command
-------------	---------

THAW

"Melts" a FREEZE and restores all the turtles' speeds to the last speed indicated. See FREEZE.

XVEL	operation
-------------	-----------

XVEL

Outputs the horizontal component of the current turtle's velocity. SETSPEED and SETHEADING can affect XVEL as well as SETXVEL.

YVEL	operation
-------------	-----------

YVEL

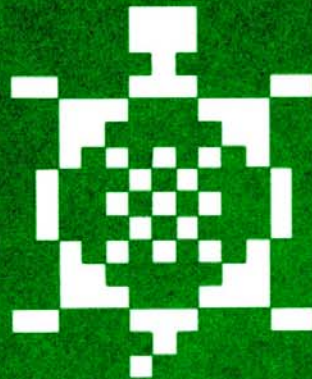
Outputs the vertical component of the current turtle's velocity. SETSPEED and SETHEADING can affect YVEL as well as SETYVEL.

1

2

3

8



Controlling 30 Turtles

Chapter 8

Any of the SmartLOGO instructions that move a turtle around, make it draw lines or change its color or shape, can be sent to any of the 30 turtles, individually or in groups.

When SmartLOGO starts up, only turtle 0 follows your instructions. The others are there, but they're hidden, and they aren't listening to your instructions.

The TELL command changes all that. TELL is used to select those turtles which will listen. Any instructions which follow a TELL command are sent to those turtles selected. An **operation**, used when several turtles are listening, only outputs information about the first turtle of the selected turtles (see EACH to get information about all selected turtles). Turtles that have not been selected will completely ignore instructions, even if they're still moving on the screen, unless TELL or ASK addresses them. For example:

```
? TELL 0  
? ST SETSP 10  
? TELL 1  
? ST SETSP 10
```

Turtle 0 and turtle 1 are moving on the screen, but only turtle 1 will follow instructions. We say that turtle 1 is the "current" or the "active" turtle. To discover which turtle or turtles are active, use the instruction PRINT WHO.

If you want turtle 0 to turn, or change its pencolor, or hide, or whatever, you must include its number in an ASK or TELL command.

```
?RT 90
?SETC 6
?SETSP 0
?TELL [0 1]
?RT 90
?SETSP 20
?CS
```

Only turtle 1 follows these instructions.

Both 0 and 1 follow these instructions.

The instructions described in this chapter allow you to control one, several, or all turtles at a time in a variety of ways. For primitives which detect collisions between turtles, see Chapter 15, Flow of Control and Conditionals.

ALL	operation
------------	-----------

ALL

Outputs the list of turtles numbered 0 through 29.

Examples:

```
?TELL ALL
?PR WHO
0 1 2 3 4 5 6 7 8 9 10 11 12 →
13 14 15 16 17 18 19 20 21 →
22 23 24 25 26 27 28 29

TO BURST
TELL ALL
CS PU
EACH [RT 12 * WHO]
ST SETSP 10
END
```

All thirty turtles burst out from the center and move across the screen.

ASK turtlenumber instructionlist

ASK turtlenumberlist instructionlist

Temporarily addresses the specified turtle or turtles giving them the instructions in *instructionlist*. This does not change the list of active turtles (which was set by the last TELL command). *Instructionlist* may be any command or operation; if it is an operation, ASK outputs whatever the operation outputs. FREEZE and THAW apply to all turtles, even if they are not in the *turtlenumberlist*.

Examples:

The following set of instructions selects the first four turtles, then points them in four different directions and gives them a speed. ASK is used to change the color of two of the moving turtles — without affecting the others — and to make another turtle show. PR WHO shows that none of this has affected the list of current turtles, which all obey the last SETC command.

```
?TELL [0 1 2 3]
?ST PU
?EACH [SETH 90 * WHO]
?SETSP 5
?ASK 1 [SETC 3]
?ASK 3 [SETC 7]
?ASK 4 [ST SETC 1]
?PR WHO
0 1 2 3
?SETC 13
```

Addressing a specific turtle from the WHO list can be done using the word and list primitives.

```
?ASK FIRST WHO [SETSH 6]
?ASK ITEM 3 WHO [ST FD 50]
```


EACH *instructionlist*

Makes each turtle in the current turtle list run the instructions in *instructionlist*, in sequence. If there is more than one active turtle, the first turtle executes all the instructions in *instructionlist* before the second turtle does anything. FREEZE and THAW apply to all turtles, even if they are not in the *turtlenumberlist*.

Examples:

The following instructions make four turtles line up 20 turtle steps apart and set their colors to their corresponding numbers + 1. WHO outputs the number corresponding to each turtle. Thus, turtle 0 will do SETX 0 and SETCOLOR 0+1, turtle 1 SETX 20 and SETCOLOR 1+1, and so on.

```
?TELL [0 1 2 3]
?ST
?HOME
?EACH [SETX WHO * 20]
?EACH [SETCOLOR WHO + 1]
```

In order to find out information about all the current turtles use EACH with the operation.

```
?EACH [PR COLOR]
1
2
3
4
```

The following instructions make the turtles draw squares at once, then one after another:

```
?CS PD
?REPEAT 4 [FD 15 RT 90]
?CLEAN
?EACH [REPEAT 4 [FD 15 RT 90 →
]]
```

TELL *turtlenumber*

TELL *turtlenumberlist*

TELL sets the current turtle or turtles to those specified in *turtlenumber* or *turtlenumberlist*. When SmartLOGO starts up, the current turtle is turtle 0. See WHO.

Examples:

```
?TELL 0 ST
?FD 50
?TELL 1 ST
?RT 90 FD 50
?TELL 2 ST
?LT 90 FD 50
?TELL [0 1]
?SETC 1
?TELL [1 2]
?SETC 9
?TELL [0 1 2]
?SETSP 10

?TELL ALL
?CS ST PD
?EACH [SETH 12 * WHO]
?FD 90 BK 90
```

To return SmartLOGO to its startup turtle state:

```
?TELL ALL
?CS HT SETSH 36
?SETC 15 SETPC 15 PD
?TELL 0
?ST
```

WHO

Outputs the numbers of the current turtles. The output is either an integer or a list of integers, from 0 through 29.

Examples:

```
?TELL 1
?PR WHO
1
?TELL [0 1]
?PR WHO
0 1
?TELL ALL
?PR WHO
0 1 2 3 4 5 6 7 8 9 10 11 12 →
13 14 15 16 17 18 19 20 21 →
22 23 24 25 26 27 28 29
```

A useful trick is to use WHO with EACH. As EACH runs the instructionlist, WHO outputs the number of each turtle separately, so each turtle will be given a different number.

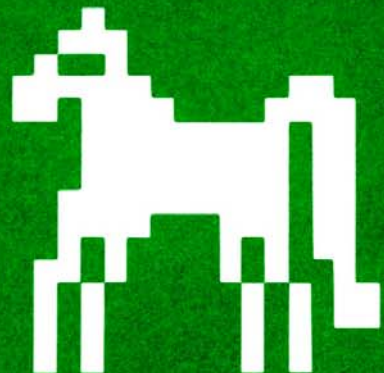
```
?TELL ALL
?ST PU
?EACH [SETH 12 * WHO]
?EACH [SETSP WHO]
?ASK 10 [PR SPEED]
10
?ASK 25 [PR SPEED]
25
```

C

C

C

9



Screen Commands

Chapter 9

When you start up SmartLOGO, the entire screen is available for text and graphics. Graphics always occupy the full screen, but you can set the text display so that it only uses the bottom of the screen. Turtle graphics commands are listed in Chapters 4 through 7. Text commands are listed in Chapter 10.

Clearing the Screen

CLEAN	command
--------------	---------

CLEAN

Erases all drawn lines as well as filled, stamped, or shaded areas, without changing the turtle state or the text displayed.

Example:

```
?RT 7 PD  
?SETSP 20  
?CLEAN
```

CLEARGRAPHICS, CG**command**

CLEARGRAPHICS

Erases all graphics from the screen, and returns all current turtles to the center of the screen pointing upwards. The text is unaffected.

Example:

```
?RT 7 PD
?SETSP 20
?CG
```

CLEARSCREEN, CS**command**

CLEARSCREEN

Erases all text and graphics from the screen, and stops all current turtles, returning them to the center of the screen, pointing upwards.

Example:

```
?RT 7 PD
?SETSP 20
?CS
```

CLEARTEXT, CT**command**

CLEARTEXT

Erases all text from the current top text line down. Leaves all graphics intact.

Example:

```
?SETTEXT 0
?CT
```

Reducing the Text Screen

SETTEXT

command

SETTEXT *line*

Sets the topmost line on which text will be printed. *Line* must be a number from 0 to 23, 0 being the top screen line and 23 the bottom screen line. When SmartLOGO starts up, text is displayed on the full screen, from line 0. Printing above a preset top line is possible with SETCURSOR (see Chapter 10), but text printed in this manner will not “scroll” with the other lines and will not be erased with CT. It is treated as graphics.

Example:

```
? SETTEXT 18
```

The text will now only come up to line 18, and will then “scroll” off the screen.

```
? SETCURSOR [13 12] PR "HELLO  
? CT
```

“HELLO” will be printed in the center of the screen, and will remain despite the CT command. To clear it:

```
? CS
```

To use the full screen for text type:

```
? SETTEXT 0
```

SETWIDTH

command

SETWIDTH *number*

Sets the width of text lines on the screen. Some televisions and monitors do not display the full width of text output by the ADAM™. SETWIDTH allows the user to match the

TV/monitor to the computer. *Number* is an integer from 1 through 6. SETWIDTH 1 sets the left column to column 1, (column 0 is for the prompt), giving a line length of 30 characters. SETWIDTH 2 sets the left column to column 2, giving the “standard” line length of 29 characters. A line length of 29 characters can contain 28 typed characters, and the arrow (→). The **RETURN** key must be pressed an extra time after the SETWIDTH command.

? SETWIDTH 4

Text has been cleared.

?

Press **RETURN**.

? SETWIDTH 2

Text has been cleared.

? ?

Press **RETURN**.

?

Back to standard line length.

The Screen's Boundaries

IN.WINDOWP

command

IN.WINDOWP

Outputs TRUE if the turtle is within the visible portion of the screen, otherwise FALSE. Always outputs TRUE in WRAP mode.

WINDOW

command

WINDOW

Makes the turtle field unbounded; what you see is a portion of the turtle field as if you were looking through a small window around the center of the screen. When the turtle moves beyond the visible bounds of the screen, it continues to move but can't be seen.

The entire turtle field is 32767 steps high and 32575 steps wide. When SmartLOGO starts up, the screen is in WRAP mode. See WRAP.

Example:

LOST.IN.SPACE sends a rocket-shape turtle off the screen and once a second it reports its distance from the center.

```
TO LOST.IN.SPACE
WINDOW
SETSH 27 SETSP 10
WHEN 0 [PR DISTANCE [0 0]]
END

?LOST.IN.SPACE
```

To stop the once-a-second WHEN demon, erase the demon with ERDS.

WRAP

command

WRAP

Makes the turtle field wrap around the edges of the screen; if the turtle moves beyond one edge of the screen it appears and continues from the opposite edge.

Following a WRAP command, the turtle never leaves the visible bounds of the screen; when it tries to, it "wraps around". Thus, the turtle can move FORWARD (or BACK) an infinite amount of times without hitting the limits of the turtle field.

When you give the WRAP command, the screen is cleared. See WINDOW.

Example:

```
?WRAP
?RT 5
?FD 500
?PRINT POS
43.57789 -77.902612
```

The Screen Ratio

SCRUNCH

command

SCRUNCH

Outputs the current scrunch setting, a single number representing the size of a vertical turtle step relative to a horizontal turtle step. When SmartLOGO starts, the scrunch is set to 1.

Example:

```
? PR SCRUNCH
1
```

SETSCRUNCH, SETSCR

command

SETSCRUNCH *ratio*

Sets the aspect ratio (the ratio of the size of a vertical turtle step to the size of a horizontal one) to *ratio*. If *ratio* is a negative number, FORWARD and BACK both work opposite to their usual way.

SETSCR 2 makes each vertical turtle step twice the length of a horizontal one.

Example:

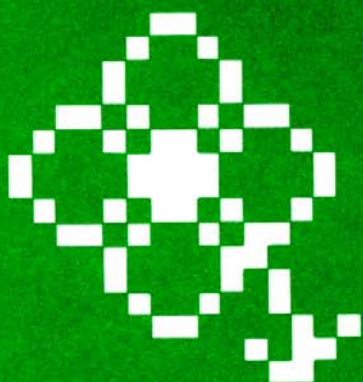
```
? PR SCRUNCH
1
? REPEAT 36 [FD 5 RT 10]
? SETSCR 2
? REPEAT 36 [FD 5 RT 10]
? SETSCR 1
```

1

2

3

10.



This chapter describes those primitives which control the printing of text on the screen.

The ADAMTM computer displays 24 lines of text on the screen, with 29 characters on each line. The first position of each Logo line is for the prompt. The last position of each screen line is for the arrow (→), which shows that the Logo line continues on the next screen line. See SETWIDTH in Chapter 9 to change the number of characters on a screen line.

The cursor is similar to the turtle — you can print characters anywhere on the screen by putting the cursor at the desired place.

For those commands which print out procedures, variables, and other workspace details (POTS, PONS, POALL etc.) see Chapter 18, Workspace Management.

CURSOR

operation

CURSOR

Outputs a list of two numbers giving the current position of the cursor. The first number gives the column and the second, the row or line in which the cursor is located. See SETCURSOR and the example given there.

PRINT *object*

(**PRINT** *object object...*)

Prints its input(s) on the screen, followed by a line feed. The outermost brackets of lists are not printed. Compare with **TYPE** and **SHOW**.

Examples:

```
? PRINT "A
```

```
A
```

```
? PRINT "A PRINT [A B C]
```

```
A
```

```
A B C
```

```
? (PRINT "A [A B C])
```

```
A A B C
```

```
? PRINT [ ]
```

```
?
```

```
TO REPRINT :MESSAGE :HOWMANY
```

```
IF :HOWMANY < 1 [STOP]
```

```
PR :MESSAGE
```

```
REPRINT :MESSAGE :HOWMANY - →
```

```
1
```

```
END
```

```
? REPRINT [TODAY IS FRIDAY!] →
```

```
4
```

```
TODAY IS FRIDAY!
```

```
TODAY IS FRIDAY!
```

```
TODAY IS FRIDAY!
```

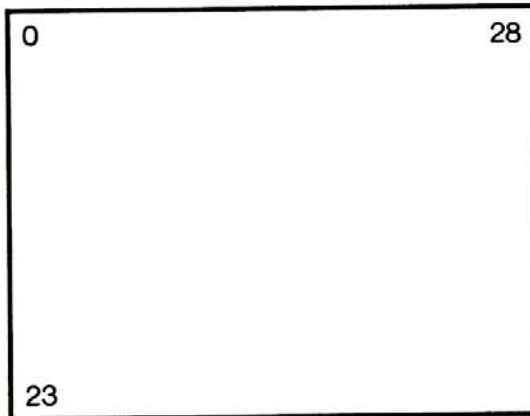
```
TODAY IS FRIDAY!
```

SETCURSOR

command

SETCURSOR *position*

Sets the cursor to *position*. The first element of *position* is the column number; the second, the line number. Columns on the screen are numbered from 0 to 28, lines from 0 to 23.



The column number must be from 0 to 28, the line number from 0 to 23. Non-integer inputs will be rounded to integer values. For example, SETCURSOR [1.7 7.2] will place the cursor at column 2, line 7.

Example:

?SETCURSOR [27 12]

Puts the cursor half-way down the right edge of the screen.

SHOW

command

SHOW *object*

Prints *object* on the screen, followed by a line feed. If *object* is a list, it is printed with brackets around it.

Compare with TYPE and PRINT.

Examples:

```
? SHOW "A
A
? SHOW "A SHOW [A B C]
A
[A B C]
? TYPE "A TYPE [A B C]
AA B C? PRINT "A PRINT [A B C]→
]
A
A B C
```

TYPE

command

TYPE object

(TYPE object object...)

Prints its input(s) on the screen, not followed by a line feed (the cursor does *not* move to the next line). The outermost brackets of a list are not printed. Compare with PRINT and SHOW.

Examples:

```
? TYPE "A
A? TYPE "A TYPE [A B C]
AA B C? (TYPE "A [A B C])
AA B C?
```

The procedure PROMPT types a message followed by a space:

```

TO PROMPT :MESSAGE
TYPE :MESSAGE
TYPE "\           Backslash followed by a space.
END

TO MOVE
PROMPT [HOW MANY STEPS?]
FD FIRST RL
MOVE
END

?MOVE
HOW MANY STEPS? 50
HOW MANY STEPS? 37
HOW MANY STEPS? 2
HOW MANY STEPS? 108

```

Press **ESCAPE/WP** to stop this procedure. The variety of ways that you can use text and characters in SmartLOGO is almost as interesting as turtle graphics itself. You can use uppercase and lowercase letters; you can change the background color on which the characters are printed.

Special Character Commands

**** (backslash)

Tells SmartLOGO to treat the following character as a regular alpha-numeric character. The backslash, (****), is used (usually with **PRINT** and **TYPE**) when you want one of SmartLOGO's special characters to be printed as a normal character in text. It is also useful when spaces are needed.

Compare

```

?PR [(514) 444-1212]
( 514 ) 444 - 1212

?PR [\ (514\ ) 444\ -1212]
(514) 444-1212

```


11.

11.

This Chapter describes SmartLOGO's primitives for generating music and sound effects. The ADAM™ computer has four sound channels. Three of them can generate "pitched" sounds, which can be used to create melodies. The fourth is the noise channel which can be used to create beeps, buzzes and sounds like wind, snare drum, and explosion effects.

The TOOT primitive activates any of the three pitched channels. The NOISE primitive activates the noise channel. NOISE requires some experience to use; you may prefer to use the procedures given as examples as a starting point for experimenting.

Music: The TOOT Primitive

TOOT	command
------	---------

TOOT *voice freq volume duration*

Turns on sound channel *voice* (0, 1, or 2) which sends a tone at *freq* (in cycles per second, or Hertz), at a loudness of *volume* (a number from 0 through 15) for a time of *duration*. A *volume* of 15 is loudest; 0 is silent. *Duration* is timed in 1/60ths of a second from 0 through 255. *Freq* is pitch; its input can be a number from 128 through 9999. (See the table of frequencies below.)

All three voices can play at the same time, but a second TOOT command to the same voice will not be started until the first TOOT is finished.

Examples:

440 Hertz is A below middle C on a piano, the "tuning note".

```
? TOOT 0 440 15 40
```

The CHORD procedure plays a three-note chord

```
TO CHORD :FREQ0 :FREQ1 :FREQ→  
2  
TOOT 0 :FREQ0 15 60  
TOOT 1 :FREQ1 15 60  
TOOT 2 :FREQ2 15 60  
END
```

```
? CHORD 130.81 164.81 196
```

will play C-major triad for one second.

Using the table of frequencies below, the names of notes may be used for operations which output the frequency for the note, for example:

```
TO C
OP 130.81
END
```

```
TO E
OP 164.81
END
```

```
TO G
OP 196
END
```

```
?CHORD C E G
```

will now play the same C-major triad.

The following table correlates notes with their frequencies for two octaves on either side of middle C.

Table of Frequencies

Note	Frequency	Note	Frequency
C	130.810	C*	523.250
D	146.830	D	587.330
E	164.810	E	659.260
F	174.610	F	698.460
G	196.000	G	783.990
A	220.000	A	880.000
B	246.940	B	987.770
C	261.630	C	1046.500
D	293.660	D	1174.700
E	329.630	E	1318.500
F	349.230	F	1396.900
G	392.000	G	1568.000
A	440.000	A	1760.000
B	493.880	B	1975.500

*middle C.

Higher (or lower) notes may be approximated by doubling (or halving) the frequency of the corresponding note in the previous (or next) octave.

To create periods of silence, use a volume of 0.

Sounds Effects

NOISE

command

NOISE type startvol stepvol steps steplength

Turns on the noise channel which sends a burst of noise *type* (see NOISE TYPES, below). The sound's volume "envelope" is set by the last four inputs.

A sound starts at a certain volume and can go through a number of changes in volume. A volume of 15 is loudest; 0 is silent. *Startvol* (0 through 15) sets the volume of the sound when it starts. *Steps* (0 through 15) sets the number of changes of volume that will occur. *Stepvol* (-7 to +7) is the amount of change of volume that will occur at each step. A positive change increases the volume at each step; a negative change decreases the volume at each step. *Stepvol* multiplied by *steps* gives the total volume change (for a natural-sounding envelope, the total decrease in volume should not be greater than the starting volume). *Steplength* (0 through 15) sets the duration of each step. The total duration of a noise is the number of steps multiplied by *steplength*.

Noise Types

There are two general noises available in SmartLOGO: the first is a kind of buzzing tone called a "sawtooth wave". There are four versions of it, each one at a different filter frequency. A filter is a tone control much like the treble/bass controls on a stereo. Type inputs 0 to 3 turn on the tone at various filter settings. The second noise is called "white noise", an undefined sort of hissy rumble something like the sound of wind. It's available in the same four filter settings, with type inputs 4 to 7. Types 3 and 7 are special in that the filter setting is the same as the pitch setting of TOOT voice 2. The ROCKET and BASSOON procedures are examples of this characteristic.

0	hi-frequency buzz	4	hi-frequency noise
1	midrange buzz	5	midrange noise
2	lo-frequency buzz	6	lo-frequency noise
3	varying buzz	7	varying noise

TO ALARM
NOISE 0 15 1 7 1
ALARM
END

TO SHOOT
NOISE 4 15 1 15 3
END

TO EXPLOSION
NOISE 6 15 1 15 15
END

TO FLYPAST
NOISE 2 0 -1 15 15
NOISE 2 15 1 15 15
END

TO ROCKET
NOISE 7 15 0 15 15
OFFSOUND 200
END

TO OFFSOUND :FREQ
TOOT 2 :FREQ 0 1
IF :FREQ > 3000 [STOP]
OFFSOUND :FREQ + 40
END

TO BASSOON
BUZZER 2092
BASSOON
END

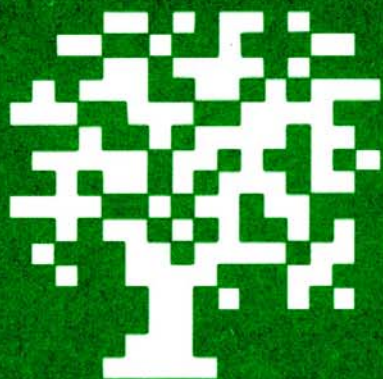
TO BUZZER :FREQ
IF :FREQ < 500 [STOP]
TOOT 2 :FREQ 0 60
NOISE 3 15 0 4 15
BUZZER :FREQ / 2
END

1

2

3

12.



There are two types of *objects* in Logo: *words* and *lists*. *Numbers* are treated as *words*. There are primitives to put them together, take them apart, and examine them.

This chapter will give you definitions and specific examples of the primitives that deal with words, numbers and lists. In order to get a more general understanding of *words* and *lists* read Chapter 2, Logo Grammar.

A *word* is made up of characters.

Example:

HELLO

X

4

314

3.14

R2D2

PIGLATIN

PIG.LATIN

WHO?

!NOW!

These are all *words*. Each character is an *element*

of the word. A word may contain only one element; X and 4 are both words. The word HELLO contains five elements:

H E L L O

A word is usually delimited by spaces. That means, there is usually a space before the word and a space after the word. The spaces set the word off from the rest of the line. There are a few other delimiting characters.

[] () = < > + - * /

SmartLOGO puts a space before and after these characters. To make any of these characters act like a normal alphabetic character, (so SmartLOGO won't put in spaces or perform the operation) put a backslash, \, before it.

Example:

```
?PRINT "PIG\L-LATIN
PIG-LATIN
```

Note that a quotation mark, ", and colon, :, are not word delimiters.

A *list* is made up of Logo *objects*, each of which is a word or another list. We indicate that something is a list by enclosing it in square brackets. The following are all *lists*:

```
[HELLO THERE, OLD CHAP]
[X Y Z]
[HELLO]
[[HOUSE MAISON] [WINDOW FENETRE] [DOG CHIEN]]
[HAL [C3PO R2D2] [QRZ] [ROBBIE SHAKEY]]
[1 [1 2] [17 [17 2]]]
[ ]
```

The list [HELLO THERE, OLD CHAP] contains four elements:

```
HELLO
THERE,
OLD
CHAP
```

Note that the list [1 [1 2] [17 [17 2]]] contains only three elements, not six; the second and third elements are themselves lists:

Element 1: 1

Element 2: [1 2]

Element 3: [17 [17 2]]

The list [], a list with no elements, is the *empty list*. There also exists an empty word, which is a word with no elements. You type in the empty word by typing a quotation mark (") followed by a space. See the entry for EMPTYP for examples of both the empty list and the empty word. The empty list and the empty word are not the same.

SmartLOGO Object Manipulators

The operations FIRST, BUTFIRST (BF), LAST and BUTLAST (BL), are used to break words and lists into pieces. The following chart shows how they work. Use the command SHOW to see them displayed on the screen.

Operation	Input	Outputs
FIRST	"JOHN	J
BF	"JOHN	OHN
FIRST	[MARY JOHN BILL]	MARY
BF	[MARY JOHN BILL]	[JOHN BILL]
FIRST	[[MARY JOHN] BILL]	[MARY JOHN]
BF	[[MARY JOHN] BILL]	[BILL]
FIRST	[] or "	error
BF	[] or "	error

LAST and BUTLAST(BL) work in the same way, separating the last element.

SmartLOGO uses five operations to put words and lists together. These are FPUT, LPUT, LIST, SENTENCE, and WORD. The following chart compares these five primitives. Use the SHOW command to display them on the screen.

Operation	Input 1	Input 1	Output
FPUT	"LOGO	"TIME	error
LIST	"LOGO	"TIME	[LOGO TIME]
LPUT	"LOGO	"TIME	error
SENTENCE	"LOGO	"TIME	[LOGO TIME]
WORD	"LOGO	"TIME	LOGOTIME
FPUT	"TURTLES	[ARE FUN]	[TURTLES ARE FUN]
LIST	"TURTLES	[ARE FUN]	[TURTLES [ARE FUN]]
LPUT	"TURTLES	[ARE FUN]	[ARE FUN TURTLES]
SENTENCE	"TURTLES	[ARE FUN]	[TURTLES ARE FUN]
WORD	"TURTLES	[ARE FUN]	error
FPUT	[AND MORE]	[TO COME]	[[AND MORE] TO COME]
LIST	[AND MORE]	[TO COME]	[[AND MORE] [TO COME]]
LPUT	[AND MORE]	[TO COME]	[TO COME [AND MORE]]
SENTENCE	[AND MORE]	[TO COME]	[AND MORE TO COME]
WORD	[AND MORE]	[TO COME]	error

SmartLOGO Object Reporters

The operations EMPTY?, EQUAL?, LIST?, MEMBER?, NUMBER? and WORD? are predicates. They examine the Logo objects that they are given as inputs and output either TRUE or FALSE.

Variables as Inputs

It is important to keep in mind that the name given to a Logo object — a variable — can always be used in place of the literal object as an input to an operation.

```
?SHOW LAST [1 [12] [17 [17 2 →  
]]]  
[17 [17 2]]  
  
?MAKE "NUMS [1 [12] [17 [17 →  
2]]]  
?SHOW LAST :NUMS  
[17 [17 2]]
```

Inputs from the Keyboard

The word and list primitives can have inputs that are typed at the keyboard. This is possible with the primitives READCHAR and READLIST. See Chapter 17.

```
?PR FIRST FIRST READLIST  
YES I WANT TO PLAY  
Y
```

BUTFIRST *object*

Outputs all but the first element of *object*. **BUTFIRST** of the empty word or the empty list is an error.

Examples:

```
?SHOW BF [BRIAN J. SMITH]
[J. SMITH]
```

```
?SHOW BF "DOGS
OGS
```

```
?SHOW BF [DOGS]
[] The empty list.
```

```
?SHOW BF [[THE A AN] [DOG CA→
T MOUSE] [BARKS MEOWS]]
[[DOG CAT MOUSE] [BARKS MEOW→
S]]
```

```
?PRINT BF "
BF DOESN'T LIKE AS AN INPUT
```

```
?PRINT BF []
BF DOESN'T LIKE [] AS AN INP→
UT
```

To following procedure removes one element at a time from a word or a list.

```
TO TRIANGLE :MESSAGE
IF EMPTY? :MESSAGE [STOP]
PRINT :MESSAGE
TRIANGLE BF :MESSAGE
END
```

```
?TRIANGLE "STROLL
STROLL
TROLL
ROLL
OLL
LL
L
```

```
? TRIANGLE [KANGAROOS JUMP GR→  
ACEFULLY]  
KANGAROOS JUMP GRACEFULLY  
JUMP GRACEFULLY  
GRACEFULLY
```

BUTLAST, BL**operation**

BUTLAST *object*

Outputs all but the last element of *object*. BUTLAST of an empty word or an empty list is an error.

Examples:

```
? SHOW BL [I YOU HE SHE IT]  
[I YOU HE SHE]  
  
? SHOW BL "FLOWER  
FLOWE  
  
? SHOW BL [FLOWER]  
[ ]
```

The input to the following procedure should be an adjective ending in Y:

```
TO COMMENT :ADJECTIVE  
PRINT SE [YOU ARE] :ADJECTIV→  
E  
PRINT SE [I AM] WORD BUTLAST→  
:ADJECTIVE "IER  
END  
  
? COMMENT "FUNNY  
YOU ARE FUNNY  
I AM FUNNIER
```

COUNT *object*

Outputs the number of elements in *object* (a word or a list).

Examples:

```
?PRINT COUNT [A QUICK BROWN →  
FOX]
```

4

```
?PRINT COUNT [A [QUICK BROWN →  
] FOX]
```

3

```
?PRINT COUNT "COMPUTER
```

8

```
?MAKE "CLASS [PAT JENNY CHRI →  
S SCOT TOM MARY JUDY]
```

```
?PRINT COUNT :CLASS
```

7

The following procedure prints a random element of its input:

```
TO RANPICK :DATA  
PRINT ITEM (1 + RANDOM COUNT →  
:DATA) :DATA  
END
```

```
?RANPICK :CLASS See list CLASS above.  
SCOT
```

```
?RANPICK "COMPUTER  
M
```

EMPTYP *object*

Outputs TRUE if *object* is the empty word or the empty list; otherwise FALSE.

Examples:

```
?PRINT EMPTYP "  
TRUE
```

```
?PRINT EMPTYP 0  
FALSE
```

```
?PRINT EMPTYP BF "UNICORN  
FALSE
```

```
?PRINT EMPTYP BL "U  
TRUE
```

```
?PRINT EMPTYP BF [UNICORN]  
TRUE
```

The procedure TALK matches animal sounds to animals:

```
TO TALK :ANIMALS :SOUNDS  
IF OR EMPTYP :SOUNDS EMPTYP →  
:ANIMALS [PRINT [THAT'S ALL →  
THERE IS!]] STOP]  
PRINT SE FIRST :ANIMALS FIRS→  
T :SOUNDS  
TALK BF :ANIMALS BF :SOUNDS  
END
```

```
?TALK [DOGS BIRDS PIGS] [BAR→  
K CHIRP OINK]  
DOGS BARK  
BIRDS CHIRP  
PIGS OINK  
THAT'S ALL THERE IS!
```


EQUALP *object object*

Outputs TRUE if the first *object* and the second *object* are equal numbers, identical words, or identical lists; otherwise FALSE. A word and a list containing that word are not equal. Equivalent to =, an infix operation. (See = at the end of this chapter.)

Examples:

```
?PRINT EQUALP "RED FIRST [RE→  
D YELLOW]  
TRUE
```

```
?PRINT EQUALP "YELLOW [YELLO→  
W]  
FALSE
```

```
?PRINT EQUALP 100 50*2  
TRUE
```

```
?PRINT EQUALP [THE A AN] [TH→  
E A]  
FALSE
```

```
?PRINT EQUALP " []  
FALSE
```

The empty word and the empty list are not identical.

The following operation outputs the position that the first input has in the second input. It outputs 0 if the first input is not an element of the second.

```
TO RANK :ONE :ALL  
IF EMPTY :ALL [OUTPUT 0]  
IF EQUALP :ONE LAST :ALL [OU→  
TPUT COUNT :ALL]  
OUTPUT RANK :ONE BL :ALL  
END
```

```
?PRINT RANK "TWO [ONE TWO TH→  
REE]  
2
```

```
?PRINT RANK "S "PERSONAL  
4
```

FIRST *object*

Outputs the first element of *object*. Note that **FIRST** of a word is a single character; **FIRST** of a list can be a word or a list. It is an error if the input is the empty word or empty list.

Examples:

```
? SHOW FIRST "HOUSE  
H
```

```
? SHOW FIRST [HOUSE]  
HOUSE
```

```
? SHOW FIRST [[THE A AN] [UNI→  
CORN RHINO] [SWIMS FLIES GRO→  
WLS RUNS]]  
[THE A AN]
```

The procedure **PRINTDOWN** prints each element of its input on a separate line.

```
TO PRINTDOWN :OBJECT  
IF EMPTY? :OBJECT [STOP]  
PR FIRST :OBJECT  
PRINTDOWN BF :OBJECT  
END
```

```
? PRINTDOWN "HELP  
H  
E  
L  
P
```

FPUT *object list*

Stands for First PUT. Outputs a new list formed by putting *object* at the beginning of *list*. See the chart at the beginning of this chapter comparing **FPUT** with other operations that combine words and lists.

Example:

The procedure REV puts the elements of the input list in reverse order.

```
TO REV :LIST
IF EMPTY? :LIST [OUTPUT []]
OUTPUT FPUT LAST :LIST REV B →
L :LIST
END

?SHOW REV [[FD 20] PU [RT 90 →
] [FD 20] PD [BK 20]]
[[BK 20] PD [FD 20] [RT 90] →
PU [FD 20]]
```

ITEM

operation

ITEM *number list*

Outputs the element whose position in *list* is *number*. It is an error if *number* is greater than the length of *list* or if *list* is the empty list.

Examples:

```
?MAKE "PETS [DOG CAT HAMSTER →
CANARY]
?PRINT ITEM 3 :PETS
HAMSTER

?PRINT ITEM 1 :PETS
DOG

?PR ITEM 4 "CUPCAKE
C
```

LAST

operation

LAST object

Outputs the last element of *object*. LAST of the empty word or the empty list is an error.

Examples:

```
?SHOW LAST [JUDY SUE BRIAN]
BRIAN
```

```
?SHOW LAST "VANILLA
A
```

```
?SHOW LAST [[THE A] FLAVOR I →
S [VANILLA CHOCOLATE STRAWBE →
RRY]]
[VANILLA CHOCOLATE STRAWBERR →
Y]
```

The following procedure prints a word in reverse order:

```
TO PRINTBACK :INPUT
IF EMPTY? :INPUT [STOP]
TYPE LAST :INPUT
PRINTBACK BL :INPUT
END
```

```
?PRINTBACK "REVERSE
ESREVER?PRINTBACK "SAW
WAS?
```

LIST

operation

LIST object object

Outputs a list whose elements are its inputs. Each input of LIST can be a word or a list. See SENTENCE.

Examples:

```
?SHOW LIST "ROSE [TULIP IRIS →
]
[ROSE [TULIP IRIS]]
```

```
?SHOW LIST [ROSE] [TULIP IRI →
S]
[[ROSE] [TULIP IRIS]]
```

```

?MAKE "DICTIONARY [[HOUSE CA→
SA] [SPANISH ESPANOL] [HOW C→
OMO]]
?SHOW :DICTIONARY
[[HOUSE CASA] [SPANISH ESPAN→
OL] [HOW COMO]]
?NEWENTRY [TABLE MESA]
?SHOW :DICTIONARY
[[HOUSE CASA] [SPANISH ESPAN→
OL] [HOW COMO] [TABLE MESA]]

```

MEMBER

operation

MEMBER *object list*

Outputs a list composed of *object* (which must be an element of *list*) and all following elements. This primitive is useful for making extremely large lists smaller, to speed up processing.

Example:

```

?PR MEMBER "FRED [0 1 2 3 FR→
ED BILL 9 8]
FRED BILL 9 8

```

MEMBERP

operation

MEMBERP *object list*

Outputs TRUE if *object* is an element of *list*; otherwise FALSE.

Examples:

```

?PRINT MEMBERP 3 [2 5 3 6]
TRUE
?PRINT MEMBERP 3 [2 5 [3] 6]
FALSE
?PRINT MEMBERP [2 5] [2 5 3 →
6]
FALSE
?PRINT MEMBERP BF "FOG [OE F→
O OG OH]
TRUE

```

The following procedure outputs TRUE if its input is a vowel, otherwise FALSE:

```
TO VOWELP :LETTER
  OUTPUT MEMBERP :LETTER [A E →
    I O U]
END

?PRINT VOWELP "F
FALSE

?PRINT VOWELP "A
TRUE
```

NUMBERP

operation

NUMBERP *object*

Outputs TRUE if *object* is a number; otherwise FALSE.

Examples:

```
?PRINT NUMBERP 3
TRUE

?PRINT NUMBERP [3]
FALSE

?PRINT NUMBERP "7PM
FALSE

?PRINT NUMBERP "
FALSE

?PRINT NUMBERP BF 3165.2
TRUE
```


SENTENCE *object object*
(**SE** *object object object...*)

Outputs a list made up of the elements included in its inputs. See the chart at the beginning of this chapter comparing **SENTENCE** with other operations that combine words and lists.

Examples:

```
? SHOW SE "PAPER "BOOKS  
[PAPER BOOKS]  
  
? SHOW SE "APPLE [PEAR PLUM B→  
ANANA  
[APPLE PEAR PLUM BANANA]  
  
? SHOW SE [TIME AND TIDE] [WA→  
IT FOR NO PERSON]  
[TIME AND TIDE WAIT FOR NO P→  
ERSON]
```

SENTENCE can be used to make a list of the results of operations. The **STARS** procedure generates a screen full of random dots.

```
TO STARS  
REPEAT 300 [DOT SENTENCE RAN→  
DOM 256 RANDOM 192]  
END  
  
? STARS
```

If **SENTENCE** has more than two inputs, you must enclose **SE** and its inputs in parentheses.

```
? SHOW (SE "HOP "STEP "JUMP)  
[HOP STEP JUMP]  
  
? SHOW SE "BONNIE [ ]  
[BONNIE]
```

This instruction sets the cursor up two lines from the current cursor position. **SE** gives **SETCURSOR** a list.

```
? SETCURSOR SE 0 LAST CURSOR →  
- 3
```

WORD *word word*

(WORD *word word word ...*)

Outputs a word made up of its inputs. If WORD has more than two inputs, you must enclose WORD and its inputs in parentheses. WORD does not work with a list as its input.

Examples:

```
?PRINT WORD "SUN "SHINE
SUNSHINE
```

```
?PRINT (WORD "CHEESE "BURG "
ER)
CHEESEBURGER
```

```
?PRINT WORD "BURG [ER]
WORD DOESN'T LIKE [ER] AS AN→
INPUT
```

```
?PRINT WORD 53 5.75
535.75
```

The procedure PREFIX puts IN at the beginning of its input:

```
TO PREFIX :WD
  OUTPUT WORD "IN :WD
END
```

```
?PRINT PREFIX "ACTIVE
INACTIVE
```

WORD is used to create new words in PIG and LATIN, which translate a sentence into a dialect of Pig Latin:

```
TO LATIN :SENT
  IF EMPTY? :SENT [OP [ ]]
  OP SE PIG FIRST :SENT LATIN →
  BF :SENT
END
```

```
TO PIG :WORD
  IF MEMBERP FIRST :WORD [A E →
  I O U] [OP WORD :WORD "AY]
  OP PIG WORD BF :WORD FIRST :→
  WORD
END
```

```

?PRINT LATIN [NO PIGS HAVE E→
VER SPOKEN PIG LATIN AMONG H→
UMANS]
ONAY IGSPAY AVEHAY EVERAY OK→
ENSPAY IGPAY ATINLAY AMONGAY→
UMANSHAY

```

WORDP

operation

WORDP object

Outputs TRUE if *object* is a word; otherwise FALSE.

Examples:

```

?PRINT WORDP "ZAM
TRUE

?PRINT WORDP 3
TRUE

?PRINT WORDP [3]
FALSE

?PRINT WORDP [E GRESS]
FALSE

```

= (Equal Sign)

infix operation

object = object

Outputs TRUE if the first *object* and the second *object* are equal numbers, identical words, or identical lists; otherwise FALSE. Equivalent to EQUALP, a prefix operation.

Examples:

```

?PRINT 3 = FIRST "3.1416
TRUE

?PRINT [THE A AN] = [THE A]
FALSE

?PRINT 7. = 7
TRUE

```

A decimal number is equivalent to the corresponding integer.

```
? PRINT " = [ ]  
FALSE
```

The empty word and the empty list are not identical.

1

2

3

13.



A Logo word can be used as a *variable*. A variable is a “container” that holds a Logo object. This object is called the word’s *value*. A variable can be assigned a value either by using MAKE or NAME or by using procedure *inputs*. For information on the proper use of variables, the difference between global and local variables, and using operations to output values, see Chapter 2, Logo Grammar. For further reference on Logo objects, see Chapter 12, Words, Numbers and Lists.

Global variables can be made at top level or in the SmartLOGO Editor. Modifications to variables are also done in the Editor. A variable name erased in the Editor is not erased from the workspace. Use EDNS to start up the Editor with all the variable names and values. The Editor and editing actions described in Chapter 3 apply to variables.

EDNS

Stands for EDit NameS. Starts up the SmartLOGO Editor with all names and their values in it. These variables can then be edited. When you exit the Editor the MAKE commands are run, so whatever variables and values have been changed or created in the Editor are changed in the workspace. See Chapter 3 for an explanation of the SmartLOGO Editor.

Example:

```
?EDNS  
MAKE "ANIMAL "GIBBON  
MAKE "SPEED 55  
MAKE "AIRCRAFT [JET HELICOPT→  
ER]
```

The “prompt” (?) disappears, indicating that the Editor is operating.

Edit the names so they look like the list below. Then press Smart Key **VI** to exit the Editor.

```
MAKE "ANIMAL "GRYFFIN  
MAKE "SPEED 55  
MAKE "AIRCRAFT [JET HELICOPT→  
ER BLIMP]
```

MAKE *name object*

Creates the variable *name* and gives it the value *object*. Once the variable is created, you can have access to its value by **THING** *name*. The abbreviation *:name* means **THING** “*name*”. The *:*(colon) means “the thing that is called” or “the value of”.

Examples:

```
?MAKE "NATIONS [CANADA USA FRANCE GERMANY ITALY]
?PRINT :NATIONS
CANADA USA FRANCE GERMANY ITALY
```

```
?PRINT "NATIONS
NATIONS
?PRINT THING "NATIONS
CANADA USA FRANCE GERMANY ITALY
```

```
?MAKE "USA [WASHINGTON]
?PRINT :USA
WASHINGTON
```

```
?PRINT THING FIRST BUTFIRST →
:NATIONS
WASHINGTON
```

FIRST BUTFIRST :NATIONS is the second element in the NATIONS list, which is USA, and the value of "USA is WASHINGTON.

```
?MAKE "CANADA [OTTAWA]
?PRINT FIRST :NATIONS
CANADA
?PRINT THING FIRST :NATIONS
OTTAWA
```

The following procedure CAPITAL asks for the capital cities of given countries.

```
TO CAPITAL :NATIONS
  IF EMPTY :NATIONS [STOP]
  MAKE "COUNTRY FIRST :NATIONS
  PR SE [THE CAPITAL OF] :COUN→
  TRY
  MAKE "ANSWER RL
  IF :ANSWER = THING :COUNTRY →
  [PR [CORRECT!]] [PR [OH! SIN→
  CE WHEN?]]
  CAPITAL BF :NATIONS
END
```

```
?CAPITAL :NATIONS
THE CAPITAL OF CANADA
OTTAWA
CORRECT!
THE CAPITAL OF USA
NEW YORK
OH! SINCE WHEN?
```

NAME	command
-------------	---------

NAME *object name*

Gives the value *object* to the variable called *name*. NAME is equivalent to MAKE, but the inputs are reversed. See MAKE.

Examples:

```
?NAME 259 "JOB
?PR :JOB
259
?NAME "WELDER "JOB
?PR :JOB
WELDER

?MAKE "JOB "WELDER
?PR :JOB
WELDER
```

NAMEP	operation
--------------	------------------

NAMEP *name*

Outputs TRUE if *name* is the name of a variable that has a value, that is, if a value for *names* exists; otherwise FALSE.

Examples:

```
?PRINT :ANIMAL
ANIMAL HAS NO VALUE
?PRINT NAMEP "ANIMAL
FALSE
?MAKE "ANIMAL "AARDVARK
?PRINT NAMEP "ANIMAL
TRUE
?PRINT :ANIMAL
AARDVARK
```

The procedure INC listed below, under THING, shows another use of NAMEP.

THING	operation
--------------	------------------

THING *name*

Outputs the value (or thing) associated with the variable *name*. THING "ANY is equivalent to :ANY. The variable can be created by the command MAKE or NAME or by defining a procedure with inputs.

Examples:

```
?MAKE "NUMBERS [0 1 2 3 4 5 →
6 7 8 9]
?PR :NUMBERS
0 1 2 3 4 5 6 7 8 9
?PR THING "NUMBERS
0 1 2 3 4 5 6 7 8 9
```

THING can be used to access the values of variables which are themselves values.

```
?MAKE 0 [A B C D]
?MAKE "A [ERIC BROWN]
?PR THING FIRST :NUMBERS
A B C D
?PR THING FIRST THING FIRST →
:NUMBERS
ERIC BROWN
```

This procedure increments (adds 1 to) the value of a variable:

```
TO INC :X
IF NOT NAMEP :X [STOP]
IF NUMBERP THING :X [MAKE :X →
  1 + THING :X]
END
```

Note the use of MAKE :X rather than MAKE "X. It is not X that's being incremented. The value of X is not a number, but the name of another variable. It is the value of the second variable that is incremented.

```
?MAKE "TOTAL 7
?PRINT :TOTAL
7
?INC "TOTAL
?PRINT :TOTAL
8
?INC "TOTAL
?PRINT :TOTAL
9
```

For other examples, see MAKE.

○

○

○

14.



SmartLOGO can work with real numbers — *integer* and *decimal* numbers:

3 is an integer.

3.14 is a decimal number.

SmartLOGO provides primitives that let you add, subtract, multiply, and divide numbers. You can find sines, cosines, and square roots; and you can test whether a number is equal to, less than, or greater than another number.

Some arithmetic operations (INT, RANDOM, REMAINDER, ROUND) always output integers. Others vary depending on the output of the operation.

Decimal numbers with more than nine digits are converted into exponential form (scientific notation). For example:

2E6 means 2 times 10^6 , or 2,000,000;

2.59N2 means 2.59 times 10^{-2} , or 0.0259

Exponents range from -38 to 37 . The largest number is 9E37 and the smallest number is 1N38.

Decimal numbers with more than nine digits are truncated. For example, the number 2718281828459.045 is converted to 2.7182818E12.

Addition, subtraction, multiplication, and division are available as *prefix operations* or *infix operations*. An infix operation goes between its inputs, not before them as with a prefix operation. Addition and multiplication in the *prefix form* take two or more inputs. The following expressions are equivalent:

2 + 1 (infix)
SUM 2 1 (prefix)

In addition to the primitives described here, the primitive EQUALP is often used in conjunction with arithmetic operations. It is described in Chapter 12, Words, Numbers and Lists. The infix operation = (Equal Sign) is equivalent to EQUALP.

The file called TOOLS on the SmartLOGO digital data pack contains many useful math tools such as a procedure that calculates the LOG of a number. See Appendix B, Program Files Included on the SmartLOGO Digital Data Pack.

Order of Mathematical Operations

When there are several math operations in a line, they are evaluated according to the operation's precedence. The order of precedence from highest to lowest is as follows:

–	a minus sign immediately before a number indicating a negative number (– 3) or the additive inverse of its input (– SPEED)
*, /	multiplication and division
+, –	addition and subtraction
>, <	greater than and less than
=	equal to
other math operations	including such primitives as SIN and SQRT as well as user-defined operations
AND, NOT, OR	logical operations
PRINT, SHOW	primitive commands

This order may be altered with the use of parentheses. Logo follows the standard mathematical practice of performing operations enclosed in parentheses before others. For example:

```
? PR 2 * 4 + 8 / 4
10
```

```
? PR 2 * ( 4 + 8 / 4 )
12
```

```
? PR ( 2 * 4 + 8 ) / 4
4
```

In the first example $2 * 4$ gives 8, and 8 divided by 4 gives 2. 8 is then added to 2 giving 10.

In the second example, the expression inside the parentheses is evaluated first. Inside the parentheses, the division is performed first. So $8/4$ gives 2, which is added to 4 giving 6. That result is multiplied by 2 giving 12.

In the third example, the expression in the parentheses is evaluated, and the result divided by 4. 2 times 4 is 8, 8 plus 8 is 16, and 16 is then divided by 4 giving 4.

ARCTAN

operation

ARCTAN *number*

Outputs the arctangent of *number*, a number between 270 and 360 or 0 and 90; the angle whose tangent is *number*.

The output is a decimal number in degrees, not radians.

Examples:

```
? PR ARCTAN 2
```

```
63.43478
```

```
? PR ARCTAN 444
```

```
89.870973
```

The following procedures define ARCSIN, an operation that outputs the angle whose sine is the input number, and ARCCOS, an operation that outputs the angle whose cosine is the input.

```
TO ARCSIN :X
OP ARCTAN :X / (SQRT 1 - :X →
* :X)
END
```

```
TO ARCCOS :X
OP ARCTAN (SQRT 1 - :X * :X) →
/ :X
END
```

COS

operation

cos degrees

Outputs the cosine of *degrees*. The output is a decimal number. It is an error if *degrees* is greater than 9999.9999 or less than -9999.9999

Examples:

```
?PRINT COS 45  
0.7071067
```

```
?PRINT COS 30  
0.86602522
```

Here is a definition of the tangent function:

```
TO TAN :ANGLE  
  OUTPUT (SIN :ANGLE) / COS :A→  
  NGLE  
END
```

```
?PRINT TAN 45  
1
```

DIFFERENCE

operation

DIFFERENCE a b

Outputs the result of subtracting *b* from *a*. Equivalent to $-$, an infix operation.

Examples:

```
?PR DIFFERENCE 7 1  
6
```

```
?PR DIFFERENCE (5+6) (3*7)  
-10
```

```
?PR DIFFERENCE 10 5  
5
```

```
?PR DIFFERENCE 6.3 107.4  
-101.1
```

INT *number*

Outputs the integer portion of *number* (by removing the decimal portion, if it exists). See ROUND.

Examples:

```
?PR INT 5.2129  
5
```

```
?PR INT 5.5129  
5
```

```
?PR INT 5  
5
```

```
?PR INT -5.8  
-5
```

```
?PR INT -12.3  
-12
```

The procedure INTP tells whether its input is an integer:

```
TO INTP :N  
  IF NOT NUMBERP :N [OUTPUT (N  
    OT A NUMBER)]  
  OUTPUT :N = INT :N  
END
```

```
?PR INTP 17  
TRUE
```

```
?PR INTP 100 / 8  
FALSE
```

```
?PR INTP "ONE  
NOT A NUMBER
```

```
?PR INTP SQRT 50  
FALSE
```

PRODUCT

operation

PRODUCT $a\ b$

(PRODUCT $a\ b\ c\ \dots$)

Outputs the product of its inputs. Equivalent to $*$, an infix operation. If PRODUCT has more than two inputs, parentheses must enclose PRODUCT and its inputs.

Examples:

```
?PR PRODUCT 6 2
```

```
12
```

```
?PR (PRODUCT 2 3 4)
```

```
24
```

```
?PR PRODUCT 2.5 4
```

```
10
```

```
?PR PRODUCT 2.5 2.5
```

```
6.25
```

QUOTIENT

operation

QUOTIENT $a\ b$

Outputs the result of dividing a by b . Equivalent to $/$, an infix operation. It is an error if b is \emptyset .

Examples:

```
?PR QUOTIENT 72 8
```

```
9
```

```
?PR QUOTIENT 12 5
```

```
2.4
```

```
?PR QUOTIENT -12 5
```

```
-2.4
```

```
?PR QUOTIENT 6 2.5
```

```
2.4
```

```
?PR QUOTIENT 3.2 0
```

```
CAN'T DIVIDE BY ZERO
```

RANDOM

operation

RANDOM *number*Outputs a random non-negative integer less than *number*.

Example:

RANDOM 6 could output 0, 1, 2, 3, 4, or 5. The following program simulates a roll of a six-sided die:

```
TO D6
OUTPUT (1 + RANDOM 6)
END

?PR D6
3
?PR D6
5
?PR D6
6
```

Note that the outputs of D6 printed here are just possible numbers and will vary because of RANDOM.

REMAINDER

operation

REMAINDER *a b*

Outputs the remainder obtained by dividing *a* by *b*. The output is always an integer. If *a* and *b* are not integers they are truncated. It is an error if *b* is 0.

Examples:

```
?PR REMAINDER 13 5
3
```

13 divided by 5 is 2 and the remainder is 3.

```
?PR REMAINDER 13 15
13
```

```
?PR REMAINDER -13 5
-3
```

The following procedure tells you whether its input is even:

```
TO EVENP :NUMBER
  OUTPUT 0 = REMAINDER :NUMBER →
    2
END

?PR EVENP 5
FALSE

?PR EVENP 12462
TRUE
```

RERANDOM

command

RERANDOM

Makes RANDOM generate the same sequence of numbers, by returning to the “startup” state of the RANDOM algorithm. RERANDOM, followed by RANDOM with the same input, will always output the same number.

Examples:

```
?RERANDOM
?REPEAT 5 [PR RANDOM 1000]
110
125
699
381
746

?RERANDOM REPEAT 5 [PR RANDO →
M 1000]
110
125
699
381
746
```

ROUND

operation

ROUND *number*

Outputs *number* rounded off to the nearest integer.

Compare with examples under INT.

Examples:

```
? PR ROUND 5.2129
5
```

```
? PR ROUND 5.5129
6
```

INT works differently.

```
? PR INT 5.5129
5
```

```
? PR ROUND .5
1
```

```
? PR ROUND -5.8
-6
```

```
? PR ROUND -12.3
-12
```

SIN

operation

SIN *degrees*

Outputs the sine value of *degrees*. See COS.

Example:

```
? PR SIN 45
0.7071067
```

SQRT

operation

SQRT *number*

Stands for SQuare RooT. Outputs the square root of *number*. It is an error if *number* is negative.

Examples:

?PR SQRT 25
5

?PR SQRT 259
16.093477

SUM

operation

SUM ab

(SUM $abc\dots$)

Outputs the sum of its inputs. Equivalent to $+$, an infix operation. If SUM has more than two inputs, SUM and its inputs must be enclosed in parentheses.

Examples:

?PR SUM 5 2
7

?PR (SUM 1 3 2 -1)
5

?PR SUM 2.3 2.561
4.861

+ (Plus Sign)

infix operation

$a + b$

Outputs the sum of its inputs, a and b . This is equivalent to SUM, a prefix operation.

Examples:

?PR 5 + 2
7

?PR 1 + 3 + 2 + 1
7

?PR 2.54 + 12.3
14.84

$a - b$

Outputs the result of subtracting b from a . Equivalent to DIFFERENCE, a prefix operation. It may be used as the sign for a negative number.

Examples:

```
?PR 7 - 1  
6
```

```
?PR 7 - 1  
6
```

```
?PR PRODUCT 7 - 1  
-7
```

```
?PR -XCOR  
-50
```

This result varies according to the turtle's position. The number shown is generated if the turtle's x-coordinate is 50.

```
?PR - 3  
-3
```

```
?PR -3 - -2  
-1
```

There could be confusion between the negative sign with one input and the minus sign with two inputs. SmartLOGO resolves this as follows:

```
?PR 3 * -4  
-12
```

```
?PR 3 + 4 - 5  
2
```

If there is a space before the “-” and a number immediately after it, SmartLOGO reads that as a negative number. So $7 - 1$ is 6, $7-1$ is 6 and $7-1$ is 6, but $7 -1$ is the pair of numbers 7 and -1.

The procedure **ABS** outputs the absolute value of its input:

```
TO ABS :NUM  
  OUTPUT IF :NUM < 0 [-:NUM] [→  
  :NUM]  
END
```

```
? PR ABS -35  
35
```

```
? PR ABS 35  
35
```

*** (Multiplication Sign)**

infix operation

$a * b$

Outputs the product of its inputs a and b . This is equivalent to **PRODUCT**, a prefix operation.

Examples:

```
? PR 6 * 2  
12
```

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

```
? PR 1.3 * -1.3  
-1.69
```

```
? PR 48 * (.3 + .2)  
24
```

The procedure FACTORIAL outputs the factorial of its input. For example, FACTORIAL 5 outputs the result of $5 * 4 * 3 * 2 * 1$ (120).

```
TO FACTORIAL :N
  IF :N = 0 [OUTPUT 1] [OUTPUT →
    :N * FACTORIAL :N - 1]
?PR FACTORIAL 4
24
?PR FACTORIAL 1
1
```

/ (Division Sign)

infix operation

a/b

Outputs the result of a divided by b . Equivalent to QUOTIENT, a prefix operation. It is an error if b is 0.

Examples:

```
?PR 6 / 3
2
```

```
?PR 8 / 3
2.6666666
```

```
?PR 2.5 / -3.8
-0.65789473
```

```
?PR 0 / 7
0
```

```
?PR 7 / 0
CAN'T DIVIDE BY ZERO
```

< (Less Than Sign)infix operation

 $a < b$

Outputs TRUE if a is less than b ; otherwise FALSE.

Examples:

```
?PR 2 < 3
```

```
TRUE
```

```
?PR -7 < -10
```

```
FALSE
```

= (Equal Sign)infix operation

 $a = b$

Outputs TRUE if a and b are equal numbers, identical words, or identical lists; otherwise FALSE. Equivalent to EQUALP, a prefix operation. See Chapter 12.

Examples:

```
?PR 100 = 50 * 2
```

```
TRUE
```

```
?PR 3 = FIRST "3.1416
```

```
TRUE
```

```
?PR 7. = 7
```

```
TRUE
```

A decimal number is equivalent to the corresponding integer.

```
?PR " = [ ]
```

```
FALSE
```

The empty word and the empty list are not identical.

$a > b$

Outputs TRUE if a is greater than b ; otherwise FALSE.

Examples:

```
?PR 4 > 3
```

```
TRUE
```

```
?PR -10 > -7
```

```
FALSE
```

The procedure **BETWEEN** outputs TRUE if the number given as the first input is greater than the second input and less than the third.

```
TO BETWEEN :N :LOW :HI  
OP AND :N > :LOW :HI > :N  
END
```

```
?PR BETWEEN 15 0 16
```

```
TRUE
```

```
?PR BETWEEN -5 -2 5
```

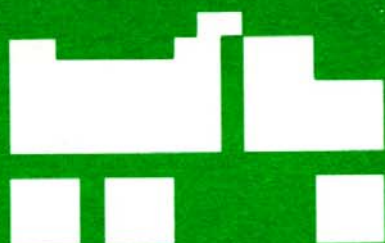
```
FALSE
```

C

C

C

15.



When SmartLOGO runs a procedure, it reads the procedure definition line by line, following the instructions given in each line. If a procedure contains a subprocedure, SmartLOGO reads the lines of the subprocedure before continuing in the superprocedure. "Flow of control" refers to the order in which SmartLOGO follows instructions. There are times when you want to alter SmartLOGO's normal flow of control. There are several ways to do it.

conditionals	"IF such-and-such is true, do one thing; otherwise, do something else."
demons	"WHEN such-and-such occurs, run a list of instructions, then resume."
repetition	"Run a list of instructions one or more times."
halting	"STOP this procedure before it reaches the END."

Conditionals enable SmartLOGO to carry out different instructions, depending on whether a condition is met or not. SmartLOGO predicates, operations that output TRUE or FALSE, create this condition, which is the first input to IF.

COND

operation

COND *condnumber*

Outputs TRUE if the event specified by *condnumber* is happening at the exact time COND is run, otherwise FALSE. The input is an integer from 0 to 5 indicating which event you want to check (see WHEN for the listing of events). COND is most useful when you want to check for an event only once.

ERCS

command

ERCS

Stands for ERase CollisionS. Erases all ON.TOUCH collision demons.

ERDS

command

ERDS

Stands for ERase DemonS. Erases all WHEN and ON.TOUCH demons.

ERES

command

ERES

Stands for ERase EventS. Erases all WHEN event demons.

IF pred instructionlist

IF pred instructionlist instructionlist

The first input, *pred*, is a predicate or condition that IF tests to be TRUE or FALSE. If *pred* is TRUE, the first *instructionlist* is run. If there is a second *instructionlist* it is run when *pred* is FALSE.

In either case, if the selected instruction list outputs, then IF outputs the same thing. If the instruction list does not output, neither does IF. Note that if you use IF with just one instruction list, and follow it on the same line with another command, SmartLOGO will print an error message.

Examples:

DECISION1, DECISION2 and DECISION3 are three equivalent procedures. The first two use IF as a command, one version with two inputs to IF, one with three inputs. The third version of DECISION uses IF (with three inputs) as an operation. All three procedures are operations.

IF as a command:

```
TO DECISION1
  IF 0 = RANDOM 2 [OP "YES]
  OP "NO
END
```

```
TO DECISION2
  IF 0 = RANDOM 2 [OP "YES] [O→
  P "NO]
END
```

IF as an operation:

```
TO DECISION3
  OUTPUT IF 0 = RANDOM 2 ["YES→
  ] ["NO]
END
```

DECISION1, DECISION2 and DECISION3 will all output YES or NO.

```
?PR DECISION1  
YES
```

DECIDE1 and DECIDE2 are both equivalent procedures. In DECIDE IF is used as a command. DECIDE2 uses IF as an operation. Both procedures are commands.

```
TO DECIDE1  
IF 0 = RANDOM 2 [PR "YES] [P →  
R "NO]  
  
TO DECIDE2  
PR IF 0 = RANDOM 2 ["YES] [" →  
NO]  
END
```

DECIDE1 and DECIDE2 print either YES or NO.

```
?DECIDE2  
NO
```

IF can be used inside another IF clause. For example;

```
TO POSITIVE? :NUM  
IF NUMBERP :NUM [IF :NUM > 0 →  
[PR [POSITIVE NUMBER]] [PR →  
[NEGATIVE NUMBER]]][PR [NOT →  
A NUMBER]]  
END
```

```
?POSITIVE? 10  
POSITIVE NUMBER
```

```
?POSITIVE? -5  
NEGATIVE NUMBER
```

```
?POSITIVE? SUM -10 5  
NEGATIVE NUMBER
```

```
?POSITIVE? "TEN  
NOT A NUMBER
```

ON.TOUCHcommand

ON.TOUCH *turtlenumber turtlenumber instructionlist*

This command can be used at top level or within a procedure. It sets a demon to watch for a collision between the two turtles specified. When the collision occurs, any current procedure is temporarily interrupted and *instructionlist* is run. If the *instructionlist* is the empty list the demon is erased.

Only ten collisions may be set at one time.

It's a good idea to include, in *instructionlist*, a command which moves at least one of the turtles away from the other, otherwise the collision may keep on registering.

Once a demon is set, it will keep watch until it is erased. To erase demons the ERDS command can be used. See WHEN.

OUTPUT, OPcommand

OUTPUT *object*

This command can be used only within a procedure, not at top level. It makes *object* the output of this procedure and returns control to the caller. Note that OUTPUT is itself a command, but the procedure containing it is an operation because the procedure is made to output (compare with STOP).

Examples:

```
TO MARK.TWAIN  
  OUTPUT [SAMUEL CLEMENS]  
END
```

```
?PR SE MARK.TWAIN [IS A GREA→  
T AUTHOR]  
SAMUEL CLEMENS IS A GREAT AU→  
THOR
```

SOMEWHERE outputs a 2-element list of random screen co-ordinates.

```
TO SOMEWHERE
OP SE RANDOM 256 RANDOM 192
END

?SETPOS SOMEWHERE
?REPEAT 300 [DOT SOMEWHERE]
```

The following procedure tells whether its first input is a subset of its second input. It outputs TRUE or FALSE. This is how you make your own *predicate*.

```
TO SUBSET :SUB :ALL
IF EMPTY? :SUB [OUTPUT "TRUE →
]
IF MEMBERP FIRST :SUB :ALL [ →
OP SUBSET BF :SUB :ALL] [OP →
"FALSE]
END

?PR SUBSET [W E F] [A E I O →
U]
FALSE

?IF SUBSET [I E] [A E I O U] →
[PR "VOWELS]
VOWELS
```

POC

command

POC *turtlenumber turtlenumber*

Stands for Print Out Collision. Prints out the specified ON.TOUCH collision and its instruction list.

POCS

command

POCS

Stands for Print Out CollisionS. Prints out all current ON.TOUCH collisions and their instruction lists.

PODS

command

PODS

Stands for Print Out DemonS. Prints out all current WHEN and ON.TOUCH demons and their instruction lists.

POE

command

POE *condnumber*

Stands for Print Out Event. Prints out WHEN *condnumber* and its instruction list.

POES

command

POES

Stands for Print Out EventS. Prints out all current WHEN events and their instruction lists.

REPEAT

command

REPEAT *number instructionlist*

Runs a list of instructions the specified number of times. It is an error if *number* is negative. If *number* is not an integer it is truncated to an integer.

Examples:

?REPEAT 4 [FD 80 RT 90]

Draws a square with 80 turtle steps on each side.

?REPEAT 5 [PRINT RANDOM 20]

Prints 5 random numbers from 0 to 19.

The following procedure draws a polygon:

```
TO POLY :SIDE :ANGLE  
REPEAT 360 / :ANGLE [FD :SIDE  
  RT :ANGLE]  
END  
  
?POLY 50 120
```

RUN

command or operation

RUN *instructionlist*

Runs the specified list of instructions as if it were typed in directly. If *instructionlist* is an operation, then RUN outputs whatever *instructionlist* outputs.

Examples:

The following procedure simulates a calculator:

```
TO CALCULATOR  
PRINT RUN RL  
PRINT []  
CALCULATOR  
END  
  
?CALCULATOR  
2 + 3  
5  
  
17.5 * 3  
52.5  
  
42 = 8 * 7  
FALSE  
  
REMAINDER 12 5  
2
```

Press **ESCAPE/WP** to stop.

PRINT RUN RL prints the output from any expression typed in by the user.

The procedure **WHILE** runs a list of instructions while a specified condition is true:

```
TO WHILE :CONDITION :INSTRUC →  
TIONLIST  
IF RUN :CONDITION :INSTRUCTI →  
ONLIST [STOP]  
WHILE :CONDITION :INSTRUCTIO →  
NLIST  
END
```

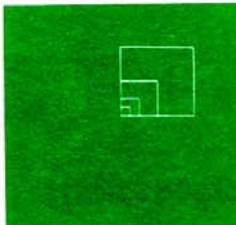
```
?RT 90  
?WHILE [XCOR < 100] [FD 25 P →  
R POS]  
25 0  
50 0  
75 0  
100 0
```

The following procedure applies a command to each element of a list in turn:

```
TO MAP :CMD :LIST  
IF EMPTY :LIST [STOP]  
RUN LIST :CMD WORD "" FIRST →  
:LIST  
MAP :CMD BF :LIST  
END
```

```
TO SQUARE :SIDE  
REPEAT 4 [FD :SIDE RT 90]  
END
```

```
?MAP "SQUARE [10 20 40 80]
```



```
?MAKE "NEW. ENGLAND [ME NH VT →  
  MA RI CT]  
?MAP "PRINT :NEW. ENGLAND  
ME  
NH  
VT  
MA  
RI  
CT
```

The FOREVER procedure repeats its input forever (unless it encounters an error or is stopped with **ESCAPE/WP**):

```
TO FOREVER :INSTRUCTIONLIST  
  RUN :INSTRUCTIONLIST  
  FOREVER :INSTRUCTIONLIST  
END  
  
?FOREVER [FD 1 RT 1]
```

STARTUP

command**STARTUP**

Runs the instruction list associated with the STARTUP variable, if one exists. See Appendix C, Startup.

STOP

command**STOP**

Stops the procedure that is running and returns control to the caller. This command is only used within a procedure — not at top level. Note that a procedure containing STOP is a command. Compare with OUTPUT.

Example:

```
TO COUNTDOWN :NUM
PR :NUM
IF :NUM = 0 [PR [BLAST OFF!] →
  STOP]
COUNTDOWN :NUM - 1
END

?COUNTDOWN 4
4
3
2
1
0
BLAST OFF!
```

TOUCHINGP

operation

TOUCHINGP *turtlenumber*

Outputs TRUE if a collision between the current turtle or any one of the current turtles and *turtlenumber* is occurring at the moment TOUCHINGP is run, otherwise FALSE. TOUCHINGP is useful when the test is to be run only once. Compare with ON.TOUCH.

WAIT

command

WAIT *number*

Tells SmartLOGO to wait for *number* 60ths of a second.

Example:

The procedure SLEEP makes SmartLOGO “sleep” for a while.

```
TO SLEEP
PR [FOR HOW MANY SECONDS?]
WAIT (FIRST RL) * 60
PR [THANKS. I NEEDED THAT.]
END
```

```
? SLEEP  
FOR HOW MANY SECONDS?  
10 SECONDS  
THANKS. I NEEDED THAT.
```

The input to WAIT is the first item in READLIST, multiplied by 60.

WHEN	command
------	---------

WHEN *condnumber instructionlist*

Sets up a WHEN demon for detecting an event *condnumber*. *Condnumber* is an integer from 0 to 5 symbolizing an event. When this event occurs, *instructionlist* is run. If the *instructionlist* includes turtle commands, the current turtle(s) (see WHO) carries them out. If the *instructionlist* is the emptylist, the demon is erased.

Note that WHEN's effect is global: this command needs to be given only once. See POD, POE, PODS and POES in Chapter 9 for checking which demons are in action.

It is possible to give more than one WHEN command at one time, but the demons will not be active simultaneously. Their speed in detecting an event depends on their strength. WHEN demon 0 is the strongest and therefore the fastest demon; WHEN demon 5 is the weakest and slowest. When one demon is busy (the event has occurred and instruction list is running), the other demons go to sleep and don't wake up until the demon has completed its task.

When setting up a game or project using demons, it is helpful to follow these guidelines:

Try to give a WHEN demon a task (instruction list) that can be executed as fast as possible.

Since only one demon can be active at a time, a WHEN or ON.TOUCH demon cannot be part of another demon's *instructionlist*. Rather, a test such as COND or TOUCHINGP should be used.

Table of events

condnumber event

0	once per second
1	a key is pressed
2	joystick on game controller 0 is moved
3	joystick on game controller 1 is moved
4	a button on game controller 0 is pressed
5	a button on game controller 1 is pressed

Examples:

When either button on game controller 0 is pressed (event 4) SmartLOGO checks whether the current turtle is touching turtle 1. If so, the current turtle disappears.

```
?TELL 0  
?WHEN 4 [IF TOUCHINGP 1 [SET→  
C 0]]
```

Whenever the joystick on game controller 0 changes position (event number 2), DRAW is executed, allowing you to draw with the joystick.

```
TO DRAW :DIR  
IF :DIR = -1 [STOP]  
SETH 45 * :DIR  
FD 5  
DRAW JOY 0  
END  
  
?WHEN 2 [DRAW JOY 0]
```

The following instructions use the button 0 demon, WHEN 4, to change the color and pencolor of the turtle. Try it while the turtle is drawing for colorful effects.

```
? WHEN 4 [SETC COLOR + 1 SETP →  
C COLOR]  
? CS  
? RT 92  
? SETSP 15
```

Press the button on game controller 0 to change the color of the turtle and of the pen.

16.



Predicates are operations that output only TRUE or FALSE. Most of their names end in P.

There are some SmartLOGO predicates whose inputs must be TRUE or FALSE. They are called logical operations. Their names do not end in P. The designers of SmartLOGO have chosen to retain the traditional names AND, OR, and NOT for these logical operations. Logical operations are used to combine predicates into logical expressions, similar to the way in which arithmetic operations form arithmetic expressions. Just as arithmetic operations receive and output only numbers, logical operations receive and output only TRUE or FALSE.

The inputs to logical operations are usually predicates. Predicates are found throughout the other chapters of this manual.

Predicate	Chapter
COND	15
DEFINEDP	3
EMPTY	12
EQUALP	12
KEYP	17
LISTP	12
MEMBERP	12
NAMEP	13
NUMBERP	12
PRIMITIVEP	3
SHOWNP	6
TOUCHINGP	8
WORDP	12
>	14
=	12, 14
<	14

AND

operation

AND pred pred

(AND pred pred pred ...)

Outputs TRUE if all its inputs are true, otherwise FALSE. If AND has more than two inputs, AND and its inputs must be enclosed in parentheses.

Examples:

```
?PRINT AND "TRUE "TRUE
TRUE
```

```
?PRINT AND "TRUE "FALSE
FALSE
```

```
?PRINT AND "FALSE "FALSE
FALSE
```

```
?PRINT (AND "TRUE "TRUE "FALSE "TRUE)
FALSE
```

```
?PRINT AND PENCOLOR = 0 BACK→  
GROUND = 0  
FALSE
```

The infix operation = returns TRUE or FALSE to AND.

```
?PRINT AND 5 7  
7 IS NOT TRUE OR FALSE
```

The following procedure, DECIMALP, tells whether its input is a decimal number:

```
TO DECIMALP :OBJ  
OP AND NUMBERP :OBJ CHECK :0→  
BJ  
END
```

```
TO CHECK :OBJ  
IF EMPTY? :OBJ [OP "FALSE]  
IF EQUALP FIRST :OBJ "." [OP →  
"TRUE  
OP CHECK BF :OBJ  
END
```

```
?PRINT DECIMALP 17.2  
TRUE  
?PRINT DECIMALP 17  
FALSE  
?PRINT DECIMALP 17.0  
FALSE
```

```
?PRINT DECIMALP 48.098  
TRUE  
?PRINT DECIMALP "STOP.STOP  
FALSE
```

FALSE

operation

FALSE

FALSE is a special word that is used as an input to AND, IF, NOT and OR.

NOT *pred*

Outputs TRUE if *pred* is FALSE; outputs FALSE if *pred* is TRUE.

Examples:

```
?PRINT NOT EQUALP "A "B  
TRUE
```

```
?PRINT NOT EQUALP "A "A  
FALSE
```

```
?PRINT NOT "A = FIRST "DOG  
TRUE
```

```
?PRINT NOT "A  
A IS NOT TRUE OR FALSE
```

VISIBLEP outputs TRUE if the turtle is visible, otherwise FALSE. The turtle must be showing and "visible" on the screen; it cannot be transparent or the same color as the background.

```
TO VISIBLEP  
IF SHOWNP [OUTPUT AND (NOT C→  
OLOR = BACKGROUND) (NOT COLO→  
R = 0)]  
OUTPUT "FALSE  
END
```

```
?ST  
?SETC 0  
?PR VISIBLEP  
FALSE
```

```
?SETC 1  
?PR VISIBLEP  
TRUE
```

The following procedure tells whether its input is a "word that isn't a number":

```
TO REALWORDP :OBJ
  OUTPUT AND WORDP :OBJ NOT NU →
  MBERP :OBJ
  END

?PRINT REALWORDP "KANGAROO
TRUE

?PRINT REALWORDP HEADING
FALSE

?PRINT REALWORDP FIRST PEN
TRUE
```

OR

operation

OR pred pred

(OR pred pred pred ...)

Outputs TRUE if one of its inputs is true, otherwise FALSE.

Examples:

```
?PRINT OR "TRUE "TRUE
TRUE

?PRINT OR "TRUE "FALSE
TRUE

?PRINT OR "FALSE "FALSE
FALSE

?PRINT OR 5 7
7 IS NOT TRUE OR FALSE

?PRINT OR COLOR = 0 COLOR = →
BG
TRUE
```

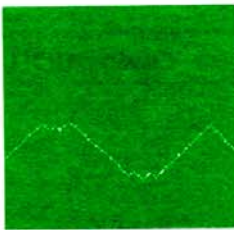
This outputs TRUE only if the turtle's color is 0 or if the turtle's color is the same as the background color.

The procedure MOUNTAINS draws "mountains":

```
TO MOUNTAINS
CS
RT 45
SUBMOUNTAIN
END

TO SUBMOUNTAIN
FD 5 + RANDOM 10
IF OR YCOR > 50 YCOR < 0 [SE→
TH 180 - HEADING]
SUBMOUNTAIN
END

?MOUNTAINS
```



TRUE

operation

TRUE

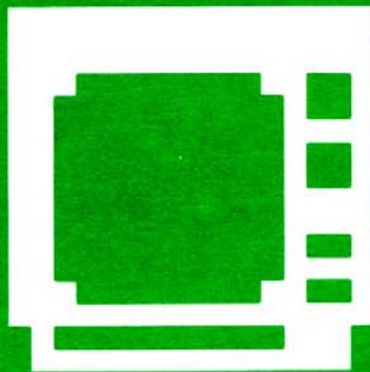
TRUE is a special word that is used as an input to AND, IF, NOT, and OR.

C

C

C

17.



Interacting with SmartLOGO: Peripheral Devices

Chapter 17

This chapter describes primitives for communicating with various devices through the computer. The devices include the game controllers, the keyboard, the printer and optional accessories such as an extra digital data pack or diskette storage devices.

The Game Controllers

The ADAM™ computer has provisions for two game controllers. Each game controller has a joystick, a right and a left button and a numeric keypad. The following primitives output information about the paddlenumber (0 or 1) given as input. The numeric keypad enters characters as if they were entered from the keyboard.

JOY *paddlenumber*

Outputs a number from -1 to 7 representing the position of the joystick on game controller *paddlenumber*. If the joystick is in the center position it outputs -1.

Example:

WHEN 2 sets up a demon which waits for the joystick on game controller 0 to be moved. When this occurs the DRAW procedure is called. The number that JOY 0 outputs is multiplied by 45 to give the turtle a new heading, and SETSP makes the turtle move in the direction of its heading. This allows you to draw graphic designs using the joystick.

```
TO DRAW :DIR
  IF :DIR = -1 [SETSP 0 STOP]
  SETH 45 * :DIR
  SETSP 5
  DRAW JOY 0
END

?WHEN 2 [DRAW JOY 0]
```

Use ERDS to erase the demon.

JOYP *paddlenumber*

Outputs TRUE if the joystick on game controller *paddlenumber* is off-center, otherwise FALSE.

Example:

SETCOURSE sets up an obstacle course of red trucks, then allows you to move the turtle through the course with RUNNER. Pressing the left button on the game controller stops the procedure.

```

TO SETCOURSE :STICKNUM
CS
SETPC 9
SETSH 25
REPEAT 20 [PU SETPOS SE RAND→
OM 255 RANDOM 191 PD STAMP]
PU
SETSH 36
SETC 15
SETSP 10
RUNNER :STICKNUM
END

TO RUNNER :STICKNUM
IF LBUTTONP :STICKNUM [SETSP→
0 STOP]
IF JOYP :STICKNUM [SETH (JOY→
:STICKNUM) * 45]
RUNNER :STICKNUM
END

?SETCOURSE 0

```

LBUTTONP

operation

LBUTTONP *paddlenumber*

Outputs TRUE if the left button on game controller *paddlenumber* is depressed, otherwise FALSE.

Examples:

The following line moves the turtle FD 75 if the left button on the game controller 0 is pressed, otherwise moves the turtle BK 75.

```

?IF LBUTTONP 0 [FD 75] [BK 7→
5]

```

GROWSQ draws a bigger square each time it is run. Pressing the left button on joystick 0 stops this recursive procedure.

```
TO GROWSQ :SIZE
  IF LBUTTONP 0 [STOP]
  SQUARE :SIZE
  GROWSQ :SIZE + 5
END

TO SQUARE :SIZE
  REPEAT 4 [FD :SIZE RT 90]
END

?GROWSQ 10
```

PADDLE	operation
---------------	-----------

PADDLE *paddlenumber*

Outputs a number from 0 to 247, representing the rotation of the joystick knob on game controller *paddlenumber*. This primitive works with Super Action™ Controllers, Roller Controller, and Expansion Module #2 (the driving controller).

RBUTTONP	operation
-----------------	-----------

RBUTTONP *paddlenumber*

Outputs TRUE if the right button on game controller *paddlenumber* is depressed, otherwise FALSE.

Examples:

```
? IF RBUTTONP 0 [PR "YES] [PR →
  "NO]
```

COLORSPI draws a spiral. Each time the right button on game controller 0 is pressed the pen color is changed. Pressing the left button stops the procedure.

```

TO COLORSPI :SIDE :ANGLE :INC →
C
IF LBUTTONP 0 [STOP]
IF RBUTTONP 0 [SETPC PENCOLO →
R + 1]
FD :SIDE
RT :ANGLE
COLORSPI :SIDE + :INC :ANGLE →
:INC
END

?COLORSPI 4 80 2

```

The Keyboard

SmartLOGO can accept characters, words or sentences input from the keyboard, and use them in user-defined procedures, with the following primitives:

KEYP	operation
-------------	-----------

KEYP

Outputs TRUE if there is at least one character waiting to be read from the keyboard, FALSE if no character is waiting to be read.

Example:

The following procedure will continue to draw a spiral until you press a key. The turtle will then stop drawing. The IGNORE procedure accepts the character that was read from the keyboard as input. The value is not used for anything; it is ignored.

```

TO SPIRAL :SIDE :ANGLE :INC
FD :SIDE RT :ANGLE
IF KEYP [IGNORE RC STOP]
SPIRAL :SIDE + :INC :ANGLE :→
INC
END

TO IGNORE :KEY
END

? SPIRAL 10 150 2

```

READCHAR, RC

operation

READCHAR

Stands for READ CHARacter. Outputs the first character read from the keyboard. If no character is waiting to be read, RC waits until the user types something. This character is not echoed on the screen. See KEYP.

Examples:

Keys which do not print characters on the screen can also be used if they have ASCII codes (see the primitives ASCII and CHAR). To easily find the ASCII value of a character:

```
? PRINT ASCII RC
```

```
13
```

Press a key, in this case, the
RETURN key.

The following procedures give the turtle a slow speed, then use single keystroke commands, R and L, to turn. If the RETURN key is pressed, the turtle's speed is set to zero and the procedures stop.

```

TO DRIVE
SETSP 2
TURN RC
END

```

```

TO TURN :DIR
IF :DIR = "R [RT 10]
IF :DIR = "L [LT 10]
IF :DIR = CHAR 13 [SETSP 0 →
STOP]
TURN RC
END

?DRIVE

```

READLIST, RL

operation

READLIST

Outputs as a list the first line of words read from the keyboard. If no list is waiting to be read, RL waits for the user to type something. If more than one line has already been typed, it outputs the first line that has been typed but not yet read. Whatever you type will be echoed on the screen.

Examples:

```

TO GET.USER
PRINT [WHAT IS YOUR NAME?]
MAKE "USER RL
PRINT SE [WELCOME TO SMARTLO →
GO.] :USER
END

?GET.USER
WHAT IS YOUR NAME?
BARBARA
WELCOME TO SMARTLOGO, BARBAR →
A

```

READWORD outputs the first element of READLIST. RW is the short form.

```
TO READWORD
OP FIRST READLIST
END

TO RW
OP READWORD
END
```

The following procedure asks your age and then prints how old you will be next year.

```
TO AGE
PRINT [HOW OLD ARE YOU?]
PRINT MESSAGE RW
END

TO MESSAGE :AGE
OP SE [NEXT YEAR YOU WILL BE →
] :AGE + 1
END

?AGE
HOW OLD ARE YOU?
22
NEXT YEAR YOU WILL BE 23

?AGE
HOW OLD ARE YOU?
28
NEXT YEAR YOU WILL BE 29
```

Using Special Characters

ASCII and CHAR make it possible to use special characters that don't show on the screen or that can't be accessed from the keyboard. These two primitives have very special uses alone or with the keyboard primitives described earlier.

ASCII character

Outputs the ASCII code for *character*. If the input word contains more than one character, ASCII uses only the first character. ASCII can access characters that don't show on the screen. See CHAR.

Examples:

```
?SHOW ASCII "B  
66
```

To find out the ASCII value of a character:

```
?PRINT ASCII RC  
163
```

The  was pressed.

The procedure SECRETCODE makes a new word by using the Caesar cipher (adding 3 to each letter):

```
TO SECRETCODE :WD  
IF EMPTY :WD [OUTPUT "]  
OUTPUT WORD CODE FIRST :WD S→  
SECRETCODE BF :WD  
END
```

```
TO CODE :LET  
MAKE "NUM (ASCII :LET) + 3  
IF :NUM > ASCII "Z [MAKE "NU→  
M :NUM - 26]  
OUTPUT CHAR :NUM  
END
```

```
?PRINT SECRETCODE "CAT  
FDW
```

```
?PRINT SECRETCODE "CRAYON  
FUDBRQ
```

CHAR number

Outputs the character whose ASCII code is *number*, an integer from 0 through 255. CHAR can output characters that can not be accessed from the keyboard. See ASCII.

Examples:

```
? TYPE CHAR 32
```

```
?
```

This types an empty space.

```
? PRINT CHAR 0
```

```
→
```

An arrow.

```
? PRINT CHAR 22
```

```
d
```

A half note.

```
? PRINT CHAR 4
```

```
♥
```

A heart.

To see all the available characters use the following procedure:

```
TO SEE.CHAR :NUM
IF :NUM > 127 [STOP]
TYPE ASCII (CHAR :NUM)
TYPE CHAR 32
PR CHAR :NUM
SEE.CHAR :NUM + 1
END
```

```
? SEE.CHAR 0
```

Both these procedures, in different ways, check to see if the **SPACEBAR** has been pressed.

```
TO SPACE?1 :K
IF :K = CHAR 32 [PR "STOP] →
[PR "CONTINUE]
END
```

```
TO SPACE?2 :K
IF (ASCII :K) = 32 [PR "STO →
P] [PR "CONTINUE]
END
```

```
? SPACE?1  RC
STOP
```

SPACEBAR is pressed.

```
? SPACE?2  RC
CONTINUE
```

Any other key is pressed.

This procedure outputs the lowercase alphabet character of the input. If you give it a character other than a letter of the alphabet, it outputs the same character.

```
TO LOWERCASE :LETTER
MAKE "SS 32 + ASCII :LETTER
IF AND :SS > 96 :SS < 123 [O→
P CHAR :SS] [OP :LETTER]
END
```

```
? PRINT LOWERCASE "A
```

a

```
? PRINT LOWERCASE "R
```

r

The SmartWRITER Printer

The ADAM™ computer has a printer which allows you to make hard copies of your work. You can use the primitives in Chapter 10 and Chapter 18 to display your workspace or your files.

NOPRINTER**command**

NOPRINTER

Turns off the output channel to the SmartWRITER printer.
See PRINTER.

Example:

The following commands print the contents of the named file onto the printer, then turn off the printer.

```
? PRINTER  
? POFIL "TOOLS"  
? NOPRINTER
```

PRINTER**command**

PRINTER

Turns on the output channel to the SmartWRITER printer.
All information subsequently displayed on the screen will also be printed on the printer. The NOPRINTER command turns the channel off.

Example:

The following commands print on paper all procedures, variables and properties in the workspace, then turn off the printer.

```
? PRINTER  
? POALL  
? NOPRINTER
```

Optional Devices

The optional file storage devices for the ADAM™ computer are: a second digital data pack drive and one or two diskette drives.

DEVICE	operation
--------	-----------

DEVICE

Outputs the number representing the current drive. See SETDEVICE.

Example:

```
?PR DEVICE
0
```

SETDEVICE	command
-----------	---------

SETDEVICE *number*

Sets the device to *number*. *Number* must be an integer from 0 to 5, with 0 addressing the standard digital data pack drive and 1 addressing an optional second digital data pack drive. 4 and 5 address the diskette drives. When SmartLOGO starts it is set to device 0.

18.



Your *workspace* consists of the procedures, variables and properties that SmartLOGO knows about at a given time. It does not include primitives or demons.

Running a demon's instruction list does use workspace, but demons are not affected by any of the following workspace management primitives. The primitives that deal with demons are explained in Chapter 15, Flow of Control and Conditionals.

There are several primitives that let you see what you have in your workspace. You can selectively erase procedures, variables and properties from your workspace. It is possible to examine the size of your workspace and to free space.

The workspace is a temporary space. Your procedures, variables and properties will be erased when you turn off the computer. If you want to keep them for future use, you must store them on a digital data pack in the form of files. See Chapter 19 for information on files. If you want to use the printer to make hard copies of your workspace contents, see Chapter 17.

ERALL**command**

ERALL

Stands for ERase ALL. Erases all procedures, variables and properties from the workspace. It does not erase demons. Make sure that all the procedures, variables and properties you want to keep are saved in a file on tape before you use this command. Use RECYCLE after ERALL to free all nodes.

ERASE, ER**command**

ERASE *name***ERASE** *namelist*

Erases the named procedure or procedures from the workspace. This command does not affect any procedures saved in a file on tape.

Examples:

```
?ERASE "TRIANGLE
```

Erases the TRIANGLE procedure.

```
?ERASE [TRIANGLE SQUARE]
```

Erases the TRIANGLE and SQUARE procedures.

ERN**command**

ERN *name***ERN** *namelist*

Stands for ERase Name. Erases the named variable or variables for the workspace.

Examples:

```
?ERN "LENGTH
```

Erases the LENGTH variable.

```
?ERN [LENGTH PI]
```

Erases the LENGTH and PI variables.

ERNS

command

ERNS

Stands for ERase NameS. Erases all variables from the workspace.

ERPROPS

command

ERPROPS

Stands for ERase PROPeritieS. Erases all the properties of all property lists. See REMPROP, in Chapter 20, to erase individual properties.

ERPS

command

ERPS

Stands for ERase ProcedureS. Erases all procedures from the workspace.

NODES

operation

NODES

Outputs the number of free nodes. This gives you an idea of how much space you have left in your workspace for procedures, variables, properties and the running of procedures. NODES is most useful if run immediately after RECYCLE. See Appendix D, Memory Space.

PO**command**

*PO name**PO namelist*

Stands for Print Out. Prints the definitions of the named procedure or procedures. You cannot print out any SmartLOGO primitives.

Examples:

```
?PO "POLY
TO POLY :SIDE :ANGLE
FD :SIDE
RT :ANGLE
POLY :SIDE :ANGLE
END
```

```
?PO [POLY GREET]
TO POLY :SIDE :ANGLE
FD :SIDE
RT :SIDE
POLY :SIDE :ANGLE
END
```

```
TO GREET
PRINT [HI THERE]
END
```

POALL**command**

POALL

Stands for Print Out ALL. Prints the definition of every procedure, the name and value of every variable and the name, property and value of all properties in the workspace.

Example:

```
?POALL
TO POLY :SIDE :ANGLE
FD :SIDE
RT :ANGLE
POLY :SIDE :ANGLE
END
```

```

TO GREET
PRINT [HI THERE]
END

TO SPI :SIDE :ANGLE :INC
FD :SIDE
RT :ANGLE
SPI :SIDE + :INC :ANGLE :INC
END

MAKE "ANIMAL "AARDVARK
MAKE "LENGTH 3.98
MAKE "NAMES [LINDA MIKE]

PPROP "RELATIVE1 "NAME [AUNT→
MADGE]
PPROP "RELATIVE1 "ADDRESS [5→
5 MAPLE STREET]
PPROP "RELATIVE1 "PHONE [234→
- 5555]

```

PONS

command

PONS

Stands for Print Out NameS. Prints the name and value of every variable in the workspace.

Example:

```

? PONS
MAKE "ANIMAL "AARDVARK
MAKE "LENGTH 3.98
MAKE "NAMES [LINDA MIKE]

```

POPS**command**

POPS

Stands for Print Out ProcedureS. Prints the definition of every procedure in the workspace.

Example:

```
?POPS
TO POLY :SIDE :ANGLE
FD :SIDE
RT :ANGLE
POLY :SIDE :ANGLE
END

TO GREET
PRINT [HI THERE]
END

TO SPI :SIDE :ANGLE :INC
FD :SIDE
RT :ANGLE
SPI :SIDE + :INC :ANGLE :INC
END
```

POTS**command**

POTS

Stands for Print Out TitleS. Prints the title line of every procedure in the workspace.

Example:

```
?POTS
TO POLY :SIDE :ANGLE
TO GREET
TO SPI :SIDE :ANGLE :INC
```


PPS

Stands for Print Properties. Prints out all property lists in the workspace.

Example:

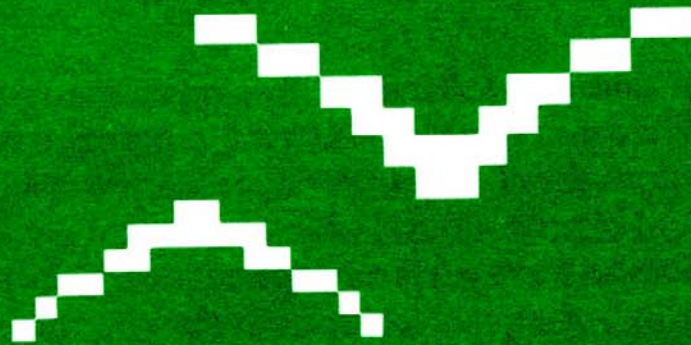
```
?PPS
PPROP "RELATIVE1" "NAME [AUNT→
MADGE]
PPROP "RELATIVE1" "ADDRESS [5→
5 MAPLE STREET]
PPROP "RELATIVE1" "PHONE [234→
- 5555]
```

RECYCLE

Performs a "garbage collection", freeing as many nodes as possible. When you don't use RECYCLE, a garbage collection happens automatically whenever necessary, but it takes a small amount of time, during which execution pauses.

Running RECYCLE before a time dependent activity prevents the automatic garbage collector from slowing things down at an awkward time. See NODES.

19.



The procedure and variables you created in the workspace are erased when you turn off the computer. If you want to keep them for future use, you can store them on a digital data pack. The information is organized in *files*. You decide what should go into each file.

There are two types of files; text files and picture files. Picture files use `SAVEPICT` and `LOADPICT`. All text files use `LOAD` but text files can be saved according to what you want them to contain. It is possible to save only procedures, or only variables, or only properties, or the entire workspace in a file. All text files are edited, erased, printed out and loaded in the same manner.

The name of a file can be 1 to 10 characters in length.

SmartLOGO can have a special file called `STARTUP` which will automatically load when SmartLOGO starts up. There is also a variable called `STARTUP` which will automatically run a list of instructions when a file is loaded. See Appendix C for information on `STARTUP`.

CATALOG	command
----------------	---------

CATALOG

Prints out the names of all the files on the digital data pack.

EDFILE	command
---------------	---------

EDFILE *filename*

Stands for EDit FILE. Starts up the Editor with the file *filename* in the Editor so that all procedures, variables and properties can be edited at once. The procedures, variables and properties do not enter the workspace nor does replacing the file save any of the workspace. If *filename* does not exist, the file is created. When the Smart Key **VI** is pressed, all the changes are stored on tape, **replacing the existing contents of *filename*.**

All editing keys (see Chapter 3, the SmartLOGO Editor) exist, including the **ESCAPE/WP** key to cancel editing.

ERASEFILE, ERF	command
-----------------------	---------

ERASEFILE *filename*

Erases the file *filename* from the digital data pack. It is an error if *filename* does not exist.

Example:

```
?ERF "BEAR
```

Erases the file called BEAR from the digital data pack.

LOAD	command
-------------	---------

LOAD *filename*

Loads the contents of *filename* into the workspace, as if typed in directly. It is an error if *filename* does not exist. The **ESCAPE/WP** key interrupts LOAD.

A listing of all procedure names in *filename* is printed on the screen as they are loaded.

After the file is loaded, you can verify the contents of your workspace using various print out commands. See Chapter 18, Workspace Management.

Example:

```
?ERALL
```

Your workspace is now empty.

```
?LOAD "BEAR
EYES DEFINED
PLAY DEFINED
JOYH DEFINED
```

LOADPICT

command

LOADPICT *filename*

Stands for LOAD PICTURE. Loads the picture file *filename* (see SAVEPICT) onto the screen. You must have saved the picture using SAVEPICT in order to use this command.

POFILE

command

POFILE *filename*

Stands for Print Out FILE. Prints out the contents of *filename*. The contents of *filename* do not enter the workspace. Files of text (for instructions or menus) can be edited with EDFILE and displayed with POFILE, so that the text is displayed without wasting workspace.

Example:

The following example prints the contents of a file on the printer.

```
?PRINTER
?POFILE "TOOLS
?NOPRINTER
```

SAVEcommand

SAVE *filename*

Saves the whole workspace in a file called *filename* which includes all procedures, variables and properties.

Never use the **ESCAPE/WP** key when a file is being saved; you might lose your workspace.

It is good practice to check what you are saving, and erase the procedures, variables and properties you don't need, before you use **SAVE**. See **POALL** and other workspace management primitives in Chapter 18.

Example:

```
?SAVE "MARIO
```

Saves the content of the workspace into the file called **MARIO** on the digital data pack.

SAVENScommand

SAVENS *filename*

Stands for **SAVE NameS**. Creates a file called *filename* in which all variable names and their values are saved. No procedures or properties are saved in this file. Use **LOAD** *filename* to load the file. This primitive is very useful when you only want to save the variables, such as the ones representing new turtle shapes created in Chapter 6.

Example:

```
?SAVENS "SHAPES
```

SAVEPICT**command**

SAVEPICT *filename*

Stands for SAVE PICTURE. Saves the current screen image as a picture file on the digital data pack in a file called *filename*. The resulting file can only be loaded with the LOADPICT command.

NOTE: Picture files are quite large. They consume 12 “blocks” of tape storage, as opposed to 1 to 3 blocks for most SmartLOGO files. If possible, it is best to keep a separate digital data pack for picture files only.

SAVEPROPS**command**

SAVEPROPS *filename*

Stands for SAVE PROPERTIES. Creates a file called *filename* in which only the properties in the workspace are saved. Use LOAD *filename* to load the file. This primitive is useful when only properties in the workspace need to be saved, such as the turtle properties created in Chapter 20.

Example:

```
? SAVEPROPS "TURTLEPROPS
```

SAVEPS**command**

SAVEPS *filename*

Stands for SAVE ProcedureS. Creates a file called *filename* which contains all the procedures in the workspace. Use LOAD *filename* to load the file. It is not necessary to erase names and properties from the workspace before using SAVEPS.

Creating a Digital Data Pack for File Storage

INITIALIZE

command

INITIALIZE *name number*

This is a dangerous primitive that must be used only when using a new digital data pack or one with replaceable data on it. **Whatever data is on the digital data pack will be destroyed.** INITIALIZE gives *name* to the digital data pack. The second input, a number from 1 to 5, indicates the size of the directory.

Example:

To be used on a new digital data pack:

```
? INITIALIZE "MYNAME 2  
? CATALOG  
? DIRECTORY :MYNAME
```

C

C

C

20.



Any SmartLOGO word can have a special list called a *property list* associated with it. A property list consists of an even number of elements. Each pair of elements consists of the name of a property and its value. For example, you might want a property list associated with the word `TURTLE0` which represents the first `TURTLE`. The property list named `TURTLE0` would then look like this:

```
[SHAPE 36 COLOR 15 SPEED 50]
```

Files of data may be created using property lists, and the properties stored using `SAVEPROPS` command. `SAVEPROPS` saves only properties, whereas `SAVE` saves the whole workspace. See Chapter 19. A typical property list of reminders about Aunt Madge might be:

```
[NAME [AUNT MADGE] ADDRESS [55 MAPLE STREET] PHONE  
[234-5555] BIRTHDAY [NOV. 15] AGE 53 SIZE 10]
```


You cannot create a property list with the MAKE or NAME commands. Property lists are built by assigning property-value pairs using the PROP primitive. A property list has the form [PROPERTY1 VALUE1 PROPERTY2 VALUE2...]. Property lists cannot be accessed as normal variables. To print or count TURTLE0's property list, you must use PLIST to tell SmartLOGO that TURTLE0 is the name of a property list not an ordinary variable. PPS prints out all the property lists.

Property lists can be very useful in keeping records or any data requiring a structured data base. Suppose you want to keep track of the properties of the turtles. TURTLE0 acts as a placekeeper for the first turtle.

You can create the property list by typing in the following:

```
?PPROP "TURTLE0 "SHAPE 36
?PPROP "TURTLE0 "COLOR 15
?PPROP "TURTLE0 "SPEED 50
```

To examine the TURTLE0 property list type:

```
?PR PLIST "TURTLE0
SHAPE 36 COLOR 15 SPEED 50

?PR GPROP "TURTLE0 "SHAPE
36
```

You can manipulate property lists using the primitives in this chapter. These primitives apply only to property lists. All of the list handling and reporting features can also be used with property lists.

```
?PR COUNT PLIST "TURTLE0
6
```

The following procedure formats the printing of a property list so that each property-value pair is printed on a separate line.

```

TO POPPROPS :PROPS
  IF EMPTY? :PROPS [STOP]
  PR (SE ITEM 1 :PROPS ITEM 2 →
    :PROPS)
  POPPROPS BF BF :PROPS
END

? POPPROPS PLIST "TURTLE0
SHAPE 36
COLOR 15
SPEED 50

```

You can use **GPROP** to write procedures that search through the list of turtles to do such things as find a given turtle's shape or list all the turtles with the same color.

Following the definitions and examples of the property list primitives, are some examples which show the uses for most of the property list primitives in this chapter.

ERPROPS

command

ERPROPS

Stands for ERase PROPerTieS. Erases all property lists. See **REMPROP** to erase individual properties. Note that **ERALL** erases all properties as well as all procedures and variables.

GPROP

operation

GPROP *name prop*

Stands for Get PROPerTy. Outputs the value of the *prop* property of *name*; outputs the empty list if there is no such property. See **PPROP** and **PLIST**.

Examples:

```
?SHOW GPROP "TURTLE0 "SHAPE
36
```

```
?SHOW GPROP "AIDS "ANY
[ ]
```

```
?SETSP GPROP "TURTLE0 "SPEED
```

See the PRINT.PROP procedure at the end of this chapter.

PLIST

operation

PLIST *name*

Outputs the property list associated with *name*. This is a list of property names paired with their values, in the form [PROPERTY1 VALUE1 PROPERTY2 VALUE2...].

Example:

```
?PR PLIST "TURTLE0
SHAPE 36 COLOR 15 SPEED 50
```

See the examples at the end of this chapter.

PPROP

command

PPROP *name prop object*

Stands for Put PROPerTy. Gives *name* the property *prop* with value *object*.

Examples:

To create the property list TURTLE1 enter the following:

```
?PPROP "TURTLE1 "SHAPE 25
?PPROP "TURTLE1 "COLOR 8

?PR PLIST "TURTLE1
SHAPE 25 COLOR 8
```

You can add another property to this list in the following way:

```
? PPROP "TURTLE1" "SPEED 25
? PR PLIST "TURTLE1
SHAPE 25 COLOR 8 SPEED 25
```

PPS

command

PPS

Stands for Print Properties. Prints the properties of all property lists in the workspace.

Example:

```
? PPS
PPROP "TURTLE1" "SHAPE 25
PPROP "TURTLE1" "COLOR 8
PPROP "TURTLE1" "SPEED 25
PPROP "TURTLE0" "SHAPE 36
PPROP "TURTLE0" "COLOR 15
PPROP "TURTLE0" "SPEED 50
```

REMPROP

command

REMPROP *name prop*

Removes property *prop* and its value from the property list of *name*. See ERPROPS and ERALL.

Example:

If you have a property list HEART you can remove properties from it in the following way:

```
? PR PLIST "HEART
SPEED 10 HEADING 40 COLOR 8 →
SHAPE 17
? REMPROP "HEART" "HEADING
? PR PLIST "HEART
SPEED 10 COLOR 8 SHAPE 17
```

A Sample Project Using Property Lists

This sample project uses the primitives described in this chapter as well as the list handling primitives. By using these primitives, property lists can be created to represent the turtles' properties. These property lists can later be used to assign properties to the turtles. These properties can be saved and then loaded when needed.

This example uses two variables, `PROPERTIES` and `SETPROPERTIES`. Both contain lists consisting of the names of the turtles' properties — they are the names of operations or commands listed in Chapters 4 through 7.

```
?MAKE "PROPERTIES ([PEN] [SP  
EED] [COLOR] [SHAPE] [POS] [  
HEADING])  
?MAKE "SETPROPERTIES ([SETP  
EN] [SETSP] [SETC] [SETSH] [S  
ETPOS] [SETH])
```

Creating Properties

The `TURTLE.STATE` procedure takes a turtle number as input. It calls another procedure, `MAKE.T.STATE`, that makes a property list for the turtle whose number was input. `TURTLE.STATE` then outputs the property list made by `MAKE.T.STATE`. Since `TURTLE.STATE` outputs a property list (uses `PLIST`) it can be used directly as the input to commands like `PRINT`, `COUNT` and `POPPOPS` (defined above).

```
TO TURTLE.STATE :TNUM
MAKE.T.STATE :TNUM :PROPERTIES
OUTPUT PLIST WORD "TURTLE :TNUM
END
```

```
TO MAKE.T.STATE :TNUM :PROPS
IF EMPTY? :PROPS [STOP]
PPROP WORD "TURTLE :TNUM FIRST :PROPS ASK :TNUM [RETURN FIRST :PROPS]
MAKE.T.STATE :TNUM BF :PROPS
END
```

The following examples show possible properties of turtles 2 and 3:

```
? POPROPS TURTLE.STATE 2
```

```
PEN PENUP 15
SPEED 5
COLOR 6
SHAPE 6
POS 54.4687 56.4414
HEADING 45
```

```
? PR TURTLE.STATE 3
```

```
PEN [PENDOWN 9] SPEED 0 COLOR 12
SHAPE 36 POS [0 0] HEADING 0
```

```
? PR COUNT TURTLE.STATE 3
```

```
12
```

You can see from TURTLE.STATE 2 that turtle 2 is actually moving. Its SPEED property is 5, so its POS (position) property is a momentary position taken "on the fly", at the moment the procedure is executed.

Assigning Properties

After creating property lists for some of the turtles by using TURTLE.STATE or PROP, CREATE.TURTLE will give these properties to the turtle. If a file contains turtle properties, it can be loaded and the properties assigned to the turtles. When a turtle number is given as input, CREATE.TURTLE uses the SETPROPERTIES and PROPERTIES variables as well as the CREATE.T.STATE procedure to determine the properties and give them to the turtle.

```
TO CREATE.TURTLE :TNUM
  TELL :TNUM
  ST
  CREATE.T.STATE :TNUM :SETPRO→
  PERTIES :PROPERTIES
  END

TO CREATE.T.STATE :TNUM :SET→
  PROPS :PROPS
  IF OR EMPTY P :SETPROPS EMPTY→
  P :PROPS [STOP]
  RUN LIST FIRST FIRST :SETPRO→
  PS GPROP WORD "TURTLE :TNUM →
  FIRST FIRST :PROPS
  CREATE.T.STATE :TNUM BF :SET→
  PROPS BF :PROPS
  END

?CREATE.TURTLE 2

?ASK 2 [PR SPEED]
5

?CREATE.TURTLE 3

?ASK 3 [PR SPEED]
0
```

Accessing the value of a property

A procedure like PRINT.PROP can be used to access the value of any property of any turtle whose property list has been created. If a property list has not been created for the turtle whose number is input, PRINT.PROP returns an empty list ([]).

```
TO PRINT.PROP :TNUM :PROP
SHOW GPROP (WORD "TURTLE :TNUM) :PROP
END
```

```
?PRINT.PROP 2 "COLOR
6
```

```
?PRINT.PROP 3 "SPEED
5
```

```
?PRINT.PROP 5 "HEADING
[ ]
```

To see all these properties type PPS.

```
?PPS
PPROP "TURTLE3 "PEN [PENDOWN→
9]
PPROP "TURTLE3 "SPEED 0
PPROP "TURTLE3 "COLOR 12
PPROP "TURTLE3 "SHAPE 36
PPROP "TURTLE3 "POS [0 0]
PPROP "TURTLE3 "HEADING 0
PPROP "TURTLE2 "PEN [PENUP 1→
5]
PPROP "TURTLE2 "SPEED 5
PPROP "TURTLE2 "COLOR 6
PPROP "TURTLE2 "SHAPE 6
PPROP "TURTLE2 "POS [54.4687→
56.4414]
PPROP "TURTLE2 "HEADING 45
```


21.



This chapter presents special primitives, some of which may affect the SmartLOGO system itself. Some machine language programming is required to use many of them successfully. They give you the power of directly accessing the computer memory, or modifying what's in it. At the same time, **they are dangerous primitives, because you can lose the contents of your workspace by using them carelessly.** If that happens, you will need to restart SmartLOGO. The names of these primitives start with a dot to warn you that they are dangerous. **You should save your work before experimenting with them.**

A note on inputs: With the primitives below, *byte* and *address* must be positive numbers in decimal format; any fraction will be ignored. *Byte* may range from 0 through 255. *Address* may range from 0 through 65535.

ALLOCATE *byte*

Reserves *byte* bytes of memory for special use.

This command allocates a block of memory to hold a machine language subroutine, or for other direct data storage (see **.CALL** and **.DEPOSIT** below). The highest machine address in this allocated block of bytes will always be 31740, and the lowest address will be $31740 - \textit{byte} + 1$.

Note that **ALLOCATE** decreases the number of nodes available to hold text during editing. One node will be eliminated for every five bytes, or fraction of five bytes, reserved.

.CALL *address*

Transfers control to a machine language subroutine starting at *address*. A RET instruction in your subroutine returns control back to SmartLOGO. Saving registers or status flags is not necessary. Your subroutine should not attempt to change values in memory locations occupied by SmartLOGO, such as page zero.

Examples:

The following procedures will install a machine language subroutine in memory, beginning at the starting address provided as the first input. The second input to the **.INSTALL** procedure should be a list of the machine language opcodes and operands of the subroutine, in hexadecimal format.

```

TO .INSTALL :BASE.ADR :BYTE. →
LIST
IF EMPTY :BYTE.LIST [STOP]
.DEPOSIT :BASE.ADR DEC FIRST →
:BYTE.LIST
.INSTALL :BASE.ADR + 1 BF :B →
YTE.LIST
END

TO DEC :HEXN
IF EMPTY :HEXN [OP 0]
IF EMPTY BL :HEXN [OP SUM A →
SCII :HEXN IF NUMBERP :HEXN →
[-48] [-55]]
OP 16 * (DEC BL :HEXN) + DEC →
LAST :HEXN
END

?.ALLOCATE 3
?.MAKE "START.ADR 31740 - 2
?.INSTALL :START.ADR [C3 AE 39]

```

.CALL might also be used to access a few subroutines in the Logo code itself. For example, the three-byte subroutine installed above causes control to jump to machine address 14766. The Logo code beginning at that address halts program execution. The following procedure uses this routine to terminate a program immediately, and return control directly to toplevel:

```

TO THROW.TOPLEVEL
.CALL :START.ADR
END

```

.CONTENTS

Outputs a list of all the objects that SmartLOGO knows about, other than primitives. These objects include your procedures, names, and properties, and any other unique words and symbols you have typed. The .CONTENTS list will become empty again each time you type ERALL and then RECYCLE.

.DEPOSITcommand

.DEPOSIT *address byte*

Writes the value *byte* into memory at machine address *address*.

Example:

The special primitives .DEPOSIT and .EXAMINE can be used to access integer data directly, in a block of memory allocated for this purpose. The procedures below imitate a one-dimensional array for integers in the range -128 through $+127$. This could be useful for the efficient storage of Cartesian coordinates. The procedure THROW.TOPLEVEL is listed elsewhere in this chapter.

```
TO DIMENSION :START.ADR :SIZE →  
  E  
  MAKE "BASE.ARRAY :START.ADR  
  MAKE "MAX.INDEX :SIZE - 1  
  END  
  
TO PUTARRAY :INDEX :VALUE  
  IF OR :INDEX < 0 :INDEX > :M →  
  AX :INDEX [RANGE.ERR "PUTARRA →  
  Y :INDEX]  
  .DEPOSIT :BASE.ARRAY + :INDE →  
  X :VALUE + 128  
  END
```

```

TO GETARRAY :INDEX
IF OR :INDEX < 0 :INDEX > :M→
AX INDEX [RANGE.ERR "GETARRA→
Y :INDEX
OP (.EXAMINE :BASE.ARRAY + :→
INDEX) - 128
END

TO RANGE.ERR :PROC.NAME :INP→
UT
PR (SE :PROC.NAME [DOESN'T.L→
IKE] :INPUT [AS INDEX])
THROW.TOPLEVEL
END

?.ALLOCATE 23
?.DIMENSION (31740 - 22) 20
?.PUTARRAY 0 98
?.PR GETARRAY 0
?.98

```

.EXAMINE

operation

.EXAMINE *address*

Outputs the byte currently stored at machine address *address*. The value output will be an integer from 0 through 255.

.PRIMITIVES

command

.PRIMITIVES

Prints out all the SmartLOGO primitives.

A.

Logo reads procedure definitions line by line and from left to right (the only exceptions are described in Chapter 14, Mathematical Operations, and Chapter 2, Logo Grammar). When Logo comes upon an expression that it cannot execute, it stops the current procedure and prints a message on the screen. The message attempts to describe the condition or "error" that must be remedied for Logo to execute the procedure. This appendix contains a fuller explanation of all the error messages SmartLOGO can send.

NOT ENOUGH SPACE TO EDIT

The edit buffer does not have the amount of room required to edit. Split a very large procedure into several smaller procedures or if you were trying to edit many procedures, edit only one or two.

NOT ENOUGH SPACE TO PROCEED

Your workspace is almost completely filled. It's best to erase some procedures and/or names from your workspace.

PRIMITIVE DOESN'T LIKE *OBJECT* AS AN INPUT
An incorrect input was given to a primitive.

WORD HAS NO VALUE
A variable was used that was not given a value.

NOT ENOUGH ITEMS IN *LIST*
A list does not have the required number of elements.

PRIMITIVE IS A PRIMITIVE
A primitive name was given as an input to TO or EDIT.

OBJECT IS NOT TRUE OR FALSE
An input was given to IF, AND, OR, or NOT that was not a predicate (didn't output TRUE or FALSE).

FILENAME NOT FOUND
The file name given as input to LOAD is nonexistent.

I CAN'T OPEN *FILENAME*
The input to SAVE or LOAD is incorrect.

YOU'RE AT TOPLEVEL
The command STOP or OUTPUT was used outside of a procedure.

STOPPED!!!
The ESCAPE/WP key was pressed, interrupting whatever was running.

CAN'T DIVIDE BY ZERO
A command was given to divide a number by zero.

FILENAME ALREADY EXISTS

A file whose name already exists can not be saved. Erase the existing file and then save, or use another file name.

DEVICE UNAVAILABLE

There is either no digital data pack in the drive or the specified device is not present.

TAPE FULL

There is no space left on the tape to save files. Old files must be erased or the digital data pack exchanged in order to save new files.

I'M HAVING TROUBLE WITH THE TAPE

This indicates a problem with your tape.

I'M HAVING TROUBLE WITH THE PRINTER

This indicates a problem with your SmartWRITER printer.

FILENAME ALREADY OPEN

A file command can not be performed while a file is loading.

TOO MANY DEMONS

A total of 16 demons can be addressed at one time; all 6 WHEN demons and no more than 10 ONTOUCH demons.

TOO COMPLEX TO FILL OR SHADE

The irregular closed area to be filled or shaded has too many sides.

TURTLE NOT IN WINDOW

COLOR.OVER, FILL and SHADE can not be performed while the turtle is beyond the visible bounds of the screen.

!!!LOGO SYSTEM BUG!!!

Should not occur. Please write to Logo Computer Systems Inc. if it does.

B



Program Files Included on the SmartLOGO Data Pack

Appendix B

Along with the file that is SmartLOGO itself, there are quite a few files provided to teach SmartLOGO, to help you build your procedures, and to help young children acquire a familiarity with the computer.

EXPLORING SMARTLOGO is a set of 9 files, each dealing with an area of SmartLOGO. Each file may be loaded on its own, or the EXPLORE file may be loaded, and the file selected from the menu.

The **DEMOS** are demonstration programs which illustrate SmartLOGO's graphic, word and list and music features.

The **EASY** files are programs which act as aids. There is a typing tutor for beginning typists and a Shape Editor aid.

The **TOOLS** are to help you construct your own procedures. The tools are divided into sections: Graphics Tools, Math Tools and Programming Tools. Some of these procedures appear elsewhere in this manual (refer to the Index).

Erasing these files is not recommended; someone else may want to learn SmartLOGO, or you may want to examine the programs, in order to learn more advanced SmartLOGO programming. All the files on the SmartLOGO Data Pack are programmed in SmartLOGO.

Using the Tutorial Files

From start-up

1. Turn ADAM™ **ON**.
2. Insert the SmartLOGO digital data pack.
3. Press the Computer **RESET** button.

SmartLOGO is loaded when the tape drive stops and the screen shows:

```
COPYRIGHT 1984
LOGO COMPUTER SYSTEMS INC.
WELCOME TO SMARTLOGO
TUTORIALS? YES OR NO
```

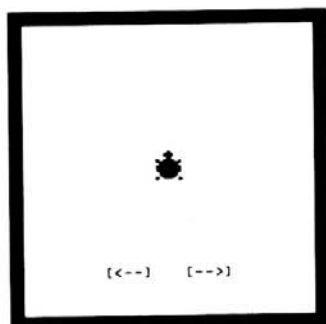
Type YES and press **RETURN**.

Or, at any time you can type:

```
?ERALL
?LOAD "EXPLORE
```

Then select any subject from the menu.

You can step through the programs at your own rate. When the picture of two arrows appears at the bottom of the screen:



Press the right arrow to go on to the next section.
Press the left arrow to go back a bit, something like turning back a couple of pages.

Using the Demonstration Programs

These demonstration programs can be entered by typing LOAD "DEMO, or each program can be loaded separately. Once a file has been loaded it will run automatically, starting with a brief description. The programs are:

Program	File Name
Poetry generator	POETRY
Hangman game	HANGMAN
Simon game	SIMON
Castle scene	CASTLE
Recursive patterns	CURVES

The *poetry generator* prints short poems that are randomly created. *Hangman* is a game in which you must guess the letters of a word. The list of words that the user must guess can be altered by editing the procedure called WORDS and changing the list of words defined by MAKE "WORDS. The *Simon game* is a musical memory game. The computer plays a note or a sequence of notes and you must repeat them. Each time you respond correctly a note is added to the sequence. The *castle scene* is a SmartLOGO picture that uses animation. There are five different selections in the *recursive patterns* programs. Each one draws a design on the screen. These programs are examples of good SmartLOGO programming. Look at them closely; they show what SmartLOGO can do.

Using the EASY programs

There are two EASY programs which can be loaded separately at any time. It's a good idea to ERALL before loading either of them.

These two programs are:

EASYSHAPE, a program that lets the user create turtle shapes with the Shape Editor. The program makes and saves the shape variables and stores them on tape. **EASYSHAPE** starts automatically once loaded. A number must then be given. This number will be the first shape number put into the Shape Editor. The following ten shapes will then be edited if desired. After ten shapes have been created, or when the user has stopped editing, these shapes can be saved in a file. Whenever this new file is loaded, the new shapes will automatically be placed in their shape numbers, replacing the contents of those shape numbers.

EASYTYPE guides the user to type specific keys to spell out SmartLOGO instructions. The predefined list of SmartLOGO instructions may be expanded. After each instruction is typed correctly, SmartLOGO runs the instruction.

Using the Useful Tools

The TOOLS file contains Graphics Tools, Math Tools and Programming Tools. The file is loaded by typing:

```
? ERALL  
? LOAD "TOOLS
```

These procedures can be incorporated into your programs, or used to calculate math problems, or to help you debug your programs.

Following is an explanation of each of the TOOLS. Some tools use subprocedures which are not mentioned but are in the TOOLS file:

Graphics Tools

ARCR and **ARCL** draw right and left turn arcs, respectively. Their inputs are :RADIUS, the radius of the circle from which the arc is taken, and :DEGREES, the degrees of the arc (the length of the edge).

Examples:

```
? ARCR 36 90  
? ARCL 45 180
```

CIRCLER and **CIRCLEL** draw right and left circles with a specified radius as input.

Examples:

```
? CIRCLER 60  
? CIRCLEL 25
```

INIT.TURTLE clears the screen and resets the screen and turtles to their initial state.

Example:

```
? INIT.TURTLE
```

POLY draws a polygon over and over. Its inputs are the length of the side and the angle. Press ESCAPE/WP to stop POLY.

Examples:

```
? POLY 25 75  
? POLY 40 155
```

ABS outputs the absolute value of its input.

Example:

```
? PR ABS -50  
50
```

CONVERT converts its first input, a number, from a base value, the second input, to another base value, the third input.

Examples:

```
? PR CONVERT 34 10 2  
100010
```

```
? PR CONVERT 1111 2 7  
21
```

DECTOHEX uses **CONVERT** to convert its input (a decimal number) to a hexadecimal number.

HEXTODEC uses **CONVERT** to convert its input (a hexadecimal number) to a decimal number.

Examples:

```
? PR DECTOHEX 77  
4D
```

```
? PR HEXTODEC "4D  
77
```

DIVISORP outputs **TRUE** if its first input divides evenly into its second, otherwise outputs **FALSE**.

Example:

```
? PR DIVISORP 3 111  
TRUE
```

LOG returns the logarithm to the base 10 of its input number. It uses the **LN** procedure.

Example:

```
? PR LOG 1000  
2.9999966
```

LN uses many math procedures, that are in the file, as subprocedures to calculate the natural logarithm of its input number.

Example:

```
? PRINT LN 50
3.9120229
? PRINT LN 2.71828
0.99999931
```

PWR returns the value of its first input to the second input power. If the second input is a fraction and the first input is not equal to 1, **PWR** uses the natural functions **EXP** and **LN**. If the first input is less than 0 and the second input is a fraction, the result should be a complex number.

EXP is the natural exponential function, calculated using a Taylor series.

Examples:

```
? PRINT PWR 23
8
? PRINT PWR 32
9
? PRINT PWR 30
1
```

Programming Tools

COMMENT allows you to embed comments in your programs. They are not seen when the program is executed, only when edited or printed out. They must be in this form: ; [THIS IS A COMMENT].

Example:

```
?PO "COLORS
TO COLORS:X
; [SHOWS ALL COLORS, USE ESC→
APE/WP TO STOP]
SETBG :X
COLORS :X + 1
END
```

FOREVER runs a list of instructions until ESCAPE/WP is pressed or the power is turned off.

Example:

```
?FOREVER [PR [ON AND]]
ON AND
ON AND
ON AND
STOPPED!!!
```

MAP applies a command to every element of a list.

Example:

```
?MAP "SQ [50 40 30 20 10]
```

SORT takes a list of words and outputs them alphabetically.
SUPERSORT arranges them in a flat list.

Examples:

```
?MAKE "SORTLIST SORT [A D E →
F T C Z ] [ ]
?PR SUPERSORT :SORTLIST
A C D E F T Z
```

Then type

```
?MAKE "SORTLIST SORT [FOO BA→
R BAZ] :SORTLIST
?PR SUPERSORT :SORTLIST
A BAR BAZ C D E F FOO T Z
?PR SUPERSORT SORT [DOG CAT →
HORSE MOUSE]
CAT DOG HORSE MOUSE
```

WHILE repeats a group of instructions, the second input,
until its first input becomes FALSE.

Example:

```
?RT 45  
?WHILE [XCOR < 75] [FD 10 ST→  
AMP]
```


C.



SmartLOGO has special features allowing you to automatically load a file when SmartLOGO starts up, and to automatically run a list of instructions when a file is loaded. These are both called **STARTUP**; the **STARTUP** file and the **STARTUP** variable.

When you first boot the SmartLOGO digital data pack, you are asked the question **TUTORIALS? YES OR NO**. This question is not part of SmartLOGO itself, it's in the provided **STARTUP** file, and it's automatically printed on the screen by the **STARTUP** variable contained in that file. You may cancel the question and load in your preferred SmartLOGO procedures by modifying the **STARTUP** file.

The **STARTUP** File

When SmartLOGO boots up, it looks for the existence of a file named **STARTUP**. There can be only one. If it finds a file named **STARTUP**, it loads it into the workspace, just as if you had typed in **LOAD "STARTUP**. Apart from this feature, the **STARTUP** file is no different from any other.

The STARTUP Variable

When any file is loaded, SmartLOGO looks for a variable named STARTUP. The STARTUP variable, if it exists, must have an instruction list as its value. If the STARTUP variable is found, SmartLOGO immediately runs that list of instructions. Any file may contain a STARTUP variable, even the STARTUP file. The STARTUP variable instruction list may even include a command to load another file.

Changing The STARTUP File: A Note of Caution

SmartLOGO contains a STARTUP file, which uses the STARTUP variable to ask a question. Depending on the answer to that question, the tutorial files may be loaded. Changing the STARTUP file may remove this feature. You should save the file under a different name in case you want it back. To do this, first, clear the workspace with an ERALL command. Then type:

```
? LOAD "STARTUP
```

When the question TUTORIALS? YES OR NO appears, do not answer, press **ESCAPE/WP** to cancel the procedure. Then save the file under the name OLDSTART or any name you prefer. For example:

```
? SAVE "OLDSTART
```

A Sample STARTUP File and Variable

Having copied the STARTUP file under another name, bring the old file into the Editor with the command:

```
?EDFILE "STARTUP
```

The tape file contents will appear in the Editor. To entirely change the file, use the CLEAR key to erase each line. Then write a procedure such as WELCOME.

```
TO WELCOME  
PR [HELLO AGAIN, ERIC]  
TYPE [HOW ARE YOU TODAY?]  
MAKE "ANSWER FIRST RL  
IF MEMBERP :ANSWER [FINE OK →  
GREAT] [PR [GOOD TO HEAR IT] →  
STOP]  
PR [WELL, LET'S HOPE LOGO-IN →  
G WILL HELP]  
END
```

Any other procedures may be added, and they will be loaded in directly after SmartLOGO itself. For WELCOME to greet you, it must be in the STARTUP variable instruction list.

Variable are stored at the end of a tape file, so move the cursor there using the down-arrow key. Find the line beginning with MAKE "STARTUP (if you haven't erased it). Edit it to look like this:

```
MAKE "STARTUP [WELCOME]
```

* Press the Smart Key **VI** to replace the old STARTUP variable with the new one, and try it out by rebooting SmartLOGO.

D.

SmartLOGO procedures and variables take up space; more space is used when the procedures are run.

Some SmartLOGO users may wish to know how space is used in SmartLOGO and how to conserve it. In general, saving space is not something you should worry about. Instead you should try to write procedures as clearly and elegantly as possible. However, we recognize that SmartLOGO has a finite memory. This appendix discusses how space is allocated in SmartLOGO and how you can use less of it.

How It Works

Space in SmartLOGO is allocated in nodes, each of which is five bytes long. All SmartLOGO objects and procedures are built out of nodes. The internal workings of SmartLOGO also use nodes. The interpreter knows about certain free nodes that are available for use. When there are no more free nodes, a special part of SmartLOGO called the garbage collector looks through all the nodes and reclaims any nodes that are not being used.

For example, during execution of the following statements:

```
?MAKE "NUMBER 7  
?MAKE "NUMBER 90
```

When you enter MAKE "NUMBER 7, NUMBER is assigned to two nodes that hold the value 7. After executing MAKE "NUMBER 90, the nodes containing the 7 can be reused; they will be reclaimed as free nodes the next time the garbage collector runs. The garbage collector runs automatically when necessary, but you can make it run with the SmartLOGO comand RECYCLE. ERALL cleans the whole workspace but needs to be followed by RECYCLE in order for the garbage collection to be done immediately.

The operation NODES outputs the number of free nodes. However, to know the actual number of free nodes available, RECYCLE must be run before printing NODES.

```
?RECYCLE PRINT NODES
```

How Space Is Used

Every SmartLOGO word used is stored only once: all occurrences of that word are actually pointers to the word. The first time a word is used it takes up four nodes, plus one node for every two letters in its name. Each time a word is used, other than the first time, it uses only one node. This is true for all SmartLOGO words (procedure names, variables and properties). A list takes one node for each element. A number, whether integer or decimal, takes up two nodes (exponent and mantissa).

Space Saving Hints

It is considered bad form to save space by writing procedures that are less readable because of the use of short or obscure words. Here are some ways of saving space:

1. Rewrite the program using procedures to replace repetitive sections of the program.
2. Avoid using new words. The names of inputs of procedures can be the same as names of inputs of other procedures. The names of procedures and primitives can also be used as variable names.
3. SE takes up more node space than LIST or FPUT. So if you have the choice, use LIST or FPUT.
4. The ways of doing repetition are ordered as follows in terms of least to most space: tail recursion (no commands after the recursive line), REPEAT, true recursion.

Note that if the recursive call is enclosed in brackets (a list), this is interpreted as true recursion.

E.

Parsing

Appendix E

When you type a line in Logo, it recognizes the characters as words and lists, and builds a list which is Logo's internal representation of the line. This process is called parsing. This appendix will help you understand how lines are parsed.

Delimiters

A word is usually delimited by spaces. That is, there is a space before the word and a space after the word; they set the word off from the rest of line. There are a few other delimiting characters:

[] () = () + - * /

There is no need to type a space between a word and any of these characters. For example, to find out how this line is parsed:

```
? IF 1(2[PRINT(3+4)/5][PRINT →  
:X+6]
```

Enter:

```
TO TEST
IF 1(2[PRINT(3+4)/5][PRINT : →
X+6]
END

?ED "TEST
```

The procedure will look like this:

```
TO TEST
IF 1 ( 2 [PRINT ( 3 + 4 ) / →
5] [PRINT :X + 6]
END
```

To treat any of the characters mentioned above as a normal alphabetic character, put a backslash “\” before it. For example:

```
?PRINT "SAN\ FRANCISCO
SAN FRANCISCO
```

Infix Procedures

The characters =, >, <, +, -, *, / are the names of infix procedures. They are treated as procedures with two inputs, but the name is written between the two inputs.

Brackets and Parentheses

Left bracket “[” and right bracket “]” indicate the start and end of a list or sublist.

Parentheses () group things in ways SmartLOGO ordinarily would not, and vary the number of inputs for certain primitives.

If the end of a SmartLOGO line is reached (that is, the **RETURN** key is pressed) and brackets or parentheses are still open, all sublists or expressions are closed. For example:

```
? REPEAT 4 [PRINT [THIS [IS [→  
A [TEST  
THIS [IS [A [TEST]]]  
THIS [IS [A [TEST]]]  
THIS [IS [A [TEST]]]  
THIS [IS [A [TEST]]]
```

If a right bracket is found for which there was no corresponding left bracket, SmartLOGO stops execution of the rest of the line or procedure. For example:

```
? ]PRINT "ABC  
? █
```

Quotes and Delimiters

Normally, you have to put a backslash before the characters [,], +, -, *, /, =, (,), >, <, and \ itself, but the first character after a quotation mark (") does not need to have a backslash preceding it. For example:

```
? PRINT " *  
*
```

If a delimiter is occupying any position but the first after the quotation mark, it must have a backslash preceding it. For example:

```
? PRINT " * * * *  
NOT ENOUGH INPUTS TO *
```

The only exception to the above general rule is [] (brackets). You must always precede a bracket that is being quoted by the backslash.

```
?PRINT "[
YOU DON'T SAY WHAT TO DO WITH
H [ ]
?PRINT "\[
[
```

The Minus Sign

The way in which the minus sign “-” is parsed is an unusual case. The problem here is that one character is used to represent three different things:

1. Part of a number to indicate that it is negative, as in -3.
2. A procedure with one input, called unary minus, which outputs the additive inverse of its input, as in -XCOR or -:DISTANCE.
3. A procedure with two inputs, which outputs the difference between its first input and its second, as in 7 - 3 and XCOR - YCOR.

The parser tries to be clever about this potential ambiguity and figure out which one was meant by the following rules:

1. If the “-” immediately precedes a number, and follows any delimiter (including a space) except right parenthesis “)”, the number is parsed as a negative number. This allows the following behavior:

?PR 3 * -1 parses as 3 times
-3 negative 1

?PR 3 *-4 parses as 3 times
-12 negative 4

?PR FIRST [- 3 4] prints -
-

?PR FIRST [-3 4] prints -3
-3

2. If “-” is preceded by a numeric expression, it works like an infix “-”.

?PR 3-4
-1

?PR XCOR - YCOR

The following are interpreted the same way:

?MAKE "A SE XCOR -YCOR 3
?MAKE "A SE XCOR - YCOR 3
?MAKE "A SE XCOR-YCOR 3

3. If “-” is not preceded by a numeric expression, it works like a unary minus.

?PR -XCOR
?PR -(3+4)

F

This appendix lists the SmartLOGO primitives in alphabetic order. Examples and a brief explanation of each primitive are also given. For complete information on each primitive, see the appropriate chapter.

A number sign (#) indicates a procedure which can take more than the number of inputs indicated; if you give it other than the number indicated, the primitive and its input must be included in parentheses.

ALL

TELL ALL PR WHO

Outputs the numbers 0 through 29.

#AND *pred pred*

IF AND (XCOR < 60) (YCOR < 0) [HT] [ST]

Outputs TRUE if all its inputs are true.

ARCTAN *number*

PR ARCTAN 4

Outputs tangent value in degrees (– 90 to 90).

ASCII *character*

PR ASCII "S

Outputs ASCII number of the input character.

ASK *turtlenumber instructionlist*

ASK *turtlenumberlist instructionlist*

ASK 0 [SETSH 6]

Makes the turtle(s) run the instruction list, does not change the WHO list.

BACK, BK *number*

BK 40

Moves the turtle back the input number of steps.

BACKGROUND, BG

PR BG

Outputs the background color number.

BUTFIRST, BF *object*

PR BF WHO

Outputs all but the first item of the input list.

BUTLAST, BL *object*

PR BL WHO

Outputs all but the last item of the input list.

CATALOG

CATALOG

Prints the names of the files on the current digital data pack.

CHANGE.COLOR *colornumber colornumber*

CHANGE.COLOR 4 6

Changes everything on the screen from the first color number to the second color number.

CHAR *number*

PR CHAR 4

Outputs the character of the input ASCII number.

CLEAN

CLEAN

Cleans the graphics screen without changing the turtle state.

CLEARGRAPHICS, CG**CG**

Clears graphics and sends the current turtle HOME.

CLEARSCREEN, CS**CS**

Clears the screen of text and graphics, sends the current turtle HOME.

CLEARTEXT, CT**CT**

Clears text from screen.

COLOR**PR COLOR**

Outputs the turtle's color.

COLOR.OVER**PR COLOR.OVER**

Outputs the number of the color beneath the turtle's pen.

COND *condnumber***IF COND 2 [SETC 0]**

Outputs TRUE if event given as input is happening.

COPYDEF *name newname***COPYDEF "OLDNAME "NEWNAME**

Copies the definition of the first input into a new procedure, the second input.

COPYSH *shapenumber newshapenumber***COPYSH 5 13**

Copies the shape of the first input into the new shape, the second input.

COS *degrees***PR COS 3**

Outputs the cosine (in degrees) of its input.

COUNT *object*

PR COUNT [0 1 2 3]

PR COUNT "BILLY

Outputs the number of items in the input (a list or a word).

CURSOR

PR CURSOR

Outputs the position of the cursor.

DEFINE *name list*

DEFINE "HI [] [PR "HELLO]

DEFINE "OK [] [OP : J]

Defines a new procedure without the Editor; generally used within a procedure.

DEFINEDP *name*

PR DEFINEDP "HI

Outputs TRUE if the input is a defined procedure.

DEVICE

PR DEVICE

Outputs a number representing the present device (drive).

DIFFERENCE *a b*

PR DIFFERENCE 10 5

Outputs the result of the second input subtracted from the first input.

DISTANCE *position*

PR DISTANCE [10 0]

Outputs the distance from the turtle to the position given as input.

DOT *position*

DOT [20 50]

Puts a dot at the specified position.

EACH *instructionlist*

EACH [ST SETSP WHO]

Causes a sequential run of the instruction list by each current turtle.

EDFILE *filename*

EDFILE "PROGRAMFILE

Places entire file in Editor, saving it when the Smart Key

VI is pressed.

EDIT, ED *name*

EDIT *namelist*

EDIT "POLY

EDIT [POLY SPI]

Invokes the Editor (identical to TO but can take a list).

EDITSHAPE, ES *shapenumber*

ES 4

Puts the input shape number in the shape Editor.

EDNS

EDNS

Places all variables and their values in the Editor.

EMPTY *object*

PR EMPTY "NAMELIST [STOP]

Outputs TRUE if the input (a list or word) is empty.

END

END

Ends the procedure definition started by TO.

EQUALP *object object*

IF EQUALP :Q 0 [STOP]

Outputs TRUE if the inputs are equal.

ERALL

ERALL

Erases all procedures, variables and properties from the workspace.

ERASE, ER *name*

ERASE *namelist*

ER "POLY

Erases the named procedure(s) from the workspace.

ERASEFILE, ERF *filename*

ERF "PROGRAMFILE

Erases the named file from the digital data pack.

ERCS

ERCS

Erases all ON.TOUCH collision demons.

ERDS

ERDS

Erases all demons (collisions and events).

ERES

ERES

Erases all WHEN event demons.

ERN *name*

ERN *namelist*

ERN "VAR.NAME

Erases the named variable(s) from the workspace.

ERNS

ERNS

Erases all variables from the workspace.

ERPROPS

ERPROPS

Erases all properties from the workspace.

ERPS

ERPS

Erases all procedures from the workspace.

FALSE

Special input for AND, IF, NOT, OR.

FILL

FILL

Fills an area with the current turtle's pen color.

FIRST *object*

PR FIRST WHO

Outputs the first item of the input (a list or word).

FORWARD, FD *number*

FD 50

Moves the turtle forward the input number of steps.

FPUT *object list*

PR FPUT 1 [2 3]

Outputs a list, with the first input as the first item.

FREEZE

FREEZE

Suspends turtle speed until THAW.

GETSH *shapenumber*

MAKE "HEART GETSH 17

Outputs a list of 32 numbers representing the turtle shape.

See PUTSH.

GPROP *name prop*

GPROP "TURTLE1 "COLOR

Outputs the value of the second input of the named property list.

HEADING

PR HEADING

Outputs the turtle's screen heading, in degrees.

HOME

TELL 0 [HOME]

Sends the current turtle to the center of the screen [0 0] with a heading of 0.

HT

HT

Makes the current turtle invisible.

IF *pred instructionlist*

IF *pred instructionlist instructionlist*

IF XCOR = 0 [RT 180] [RT 90]

If the first input outputs TRUE, the first instructionlist is run. If FALSE, the optional second instructionlist is run.

IN.WINDOWP

IF NOT IN.WINDOWP [HOME]

Outputs TRUE if the turtle is within the visible portion of the screen (see WINDOW and WRAP).

INT *number*

PR INT 1.714

Outputs the integer value of input number (truncated).

ITEM *number list*

PR ITEM 2 [0 1 2]

PR ITEM 2 "BLABLA

Outputs the first input item of the second input (a list or a word).

JOY *paddlenumber*

PR JOY 0

Outputs a number representing the current position of the joystick on the paddle number given as input.

JOYP *paddlenumber*

IF JOYP 0 [SETH JOY 0 * 45]

Outputs TRUE if the joystick on the paddle number given as input is off center.

KEYP

IF KEYP [STOP]

Outputs TRUE if a key has been typed but not yet read.

LAST *object*

PR LAST [0 1 2 3 4]

Outputs the last item of the input (a list or a word).

LBUTTONP *paddlenumber*

IF LBUTTONP 0 [PR "FINISHED]

Outputs TRUE if the left button on joystick number given as input is pressed.

LEFT, LT *degrees*

LT 90

Turns the turtle left the input number of degrees.

LIST *object object*

SETPOS LIST 35 * 2 66

PR LIST " ONE [TWO THREE]

Outputs a list composed of its inputs, including a list of lists (see SE).

LISTP *object*

IF LISTP :Q [RUN :Q]

Outputs TRUE if the input is a list.

LOAD *filename*

LOAD "TOOLS

Loads the named file into the workspace.

LOADPICT *filename*

LOADPICT "PICTURE

Loads the named picture file onto the screen.

LPUT *object list*

PR LPUT "ONE [2 3]

Outputs a list with its first input as the last item. (See FPUT.)

MAKE *name object*

MAKE "MYNAME "ERIC

MAKE "NUMLIST [1 2 3]

MAKE "GANG WHO

Sets a *global variable*. Gives the first input (the name, quoted) the second input as value. The value remains until reset or erased.

MEMBER *object list*

PR MEMBER "M [A G M P S]

Outputs a list containing the first input and all following members of the second input (a list).

MEMBERP *object list*

IF MEMBERP "ERIC : NUMLIST [PR "YES]

Outputs TRUE if the first input is a member of the second input (a list).

NAME *object name*

NAME 1.25 "AMOUNT

Sets a *global variable*. Gives the second input (the name) the first input as value. The value remains until reset or erased.

NAMEP *name*

PR NAMEP "MYNAME

Outputs TRUE if the input is a global variable name.

NODES

PR NODES

Outputs the size of available workspace, in nodes.

NOISE *type startvol stepvol steps steplength*

NOISE 6 15 1 3 3

Sound primitive with 5 inputs (0 to 7, 0 to 15, - 7 to 7, 0 to 15, 0 to 15).

NOPRINTER

NOPRINTER

Turns off the output channel to the printer.

NOT *pred*

IF NOT 0 = HEADING [DO.IT]

Outputs TRUE if the input is FALSE.

NUMBERP *object*

IF NUMBERP FIRST: NUMLIST [DO.IT] [DON'T]

Outputs TRUE if the input is a number.

ON.TOUCH *turtlenumber turtlenumber instructionlist*

ON. TOUCH 0 1 [RT 90]

Sets a demon to wait for a collision between the two input turtles, then run the instruction list.

OR *pred pred*

IF OR (:X < 2) (:X > -2) [STOP]

Outputs TRUE if any of its inputs are TRUE.

OUTPUT, OP *object*

Stops the current procedure and outputs (a list, word, TRUE or FALSE). Used within a procedure only.

PADDLE *paddlenumber*

SETSP PADDLE 0

Outputs a number (0 to 247) representing the rotation of paddle knob on the paddle number given as input. Works only with additional game paddles.

PEN

PR PEN

Outputs the state and color of the turtle's pen.

PENCOLOR, PC

PR PC

Outputs the current turtle's pen color.

PENDOWN, PD

PD

Sets the current turtle's pen to draw lines.

PENERASE, PE

PE

Sets the current turtle's pen to erase any lines crossed.

PENREVERSE, PX

PX

Sets a pen mode which draws lines unless over a drawn line; in that case it erases the line.

PENUP, PU

PU

Raises the current turtle's pen.

PLIST *name*

PLIST "TURTLE0

Outputs the property list associated with the input.

PO *namelist*

PO "POLY

PO [POLY SPI]

Prints out the named procedure(s).

POALL

POALL

Prints out the entire contents of the workspace.

POC *turtlenumber turtlenumber*

POC 0 1

Prints out the named ON.TOUCH collision demon.

POCS

POCS

Prints out all ON.TOUCH collision demons.

PODS

PODS

Prints out all demons and their instruction lists.

POE *condnumber*

POE 0

Prints out the specified WHEN event demon.

POES

POES

Prints all WHEN event demons.

POFILE *filename*

POFILE "MENU

Prints out the named file.

PONS

PONS

Prints out all global variables and their values.

POPS

POPS

Prints out all procedures in the workspace.

POS

PR POS

Outputs the screen position of the current turtle.

POTS

POTS

Prints out the titles of all procedures in the workspace.

PPROP *name prop object*

PPROP "TURTLE0 "SHAPE 8

Gives the first input the property (second print) with the third input as the value.

PPS

PPS

Prints out all properties in the workspace.

PRIMITIVEP *name*

PR PRIMITIVEP "FD

Outputs TRUE if the input is a primitive.

PRINT, PR *object*

PR "HELLO

Prints its input on the screen, followed by a carriage-return.

PRINTER

PRINTER

Turns on the output channel to the printer.

PRODUCT *a b*

PR PRODUCT 3 5

Outputs the product of its two inputs (multiplies).

PUTSH *shapenumber shapespec*

PUTSH 15 :HEART

Gives the named shape number the form defined by the second input (a list of 32 numbers). See GETSH.

QUOTIENT *a b*

PR QUOTIENT 6 3

Outputs the result of dividing the first input by the second input.

RANDOM *number*

PR RANDOM 8

Outputs a random integer between 0 and its input - 1.

RBUTTONP *paddlenumber*

IF RBUTTONP 0 [PR "START]

Outputs TRUE if the right button on joystick number given as input is pressed.

READCHAR, RC

PR RC

Reads one keyboard character.

READLIST, RL

MAKE "NAME RL

Outputs the list input from the keyboard. (Pressing the RETURN key signifies the end of the list.)

RECYCLE

RECYCLE

Forces a "garbage collection" freeing unused nodes.

REMAINDER *a b*

PR REMAINDER 10 3

Outputs the remainder of the first input divided by the second input.

REMPROP *name prop*

REMPROP "TURTLE "SHAPE

Removes the named property and its value from the named property list.

REPEAT *number instructionlist*

REPEAT 3 [FD 50 RT 120]

Runs the input instruction list the specified number of times.

RERANDOM

RERANDOM

PR RANDOM 10

Generates the same sequence of random numbers.

RIGHT, RT *degrees*

RT 75

Turns the turtle right the input number of degrees.

ROUND *number*

PR ROUND 3.7

Outputs the rounded integer value of the input number.

RUN *instructionlist*

RUN [FD 69]

Runs the input instruction list.

SAVE *filename*

SAVE "NEWFILE

Saves all the workspace as a file onto the digital data pack.

SAVENS *filename*

SAVENS "SHAPES

Saves only variables in the workspace onto the digital data pack.

SAVEPICT *filename*

SAVEPICT "SCENE

Saves the screen image onto the digital data pack.

SAVEPROPS *filename*

SAVE "TURTLES

Saves only property lists in the workspace onto the digital data pack.

SAVEPS *filename*

SAVE "GAME

Saves only procedures in the workspace onto the digital data pack.

SCRUNCH

PR SCRUNCH

Outputs the current aspect ratio of the screen.

SENTENCE, SE *object object*

PR (SE "ONE [TWO THREE 4])

SETPOS SE 5*10 35

Makes the inputs into a single list; removing inner brackets (see LIST).

SETBG *colornumber*

SETBG 1

Sets the background color to the input color number.

SETCOLOR, SETC

SETC 3

Sets the color of the current turtle.

SETCURSOR *position*

SETCURSOR [10 10]

Sets the text cursor to the specified text position.

SETDEVICE *number*

SETDEVICE 0

Sets the device to the device represented by the input.

SETHEADING, SETH *degrees*

SETH 270

Sets the heading of the current turtle, in degrees (absolute).

SETPEN *list*

SETPEN [PD 5]

Sets the color and state of the current turtle's pen.

SETPENCOLOR, SETPC *colornumber*

SETPC 5

Sets the color of the current turtle's pen.

SETPOS *position*

SETPOS [20 - 65]

Sets the turtle's screen position (absolute).

SETSCRUNCH, SETSCR *ratio*

SETSCRUNCH 2

Changes the size of a turtle step on the x-axis relative to the y-axis (normally set to 1).

SETSHAPE, SETSH *shapenumber*

SETSH 10

Sets the turtle's shape, (0 to 59).

SETSPEED, SETSP

SETSP 30

Sets the turtle's speed, (−128 to 128).

SETTEXT *line*

SETTEXT 18

Sets the topmost line for text (0 top, 23 bottom).

SETWIDTH *number*

SETWIDTH 2

Sets the width of the text lines. The input number sets the left column.

SETX *x*

SETX 90

Places the turtle at the given x-coordinate.

SETXVEL *speed*

SETXVEL 0

Sets the x-component of the turtle's speed.

SETY *y*

SETY −65

Places the turtle at the given y-coordinate.

SETYVEL *speed*

SETYVEL ASK 0 [YVEL]

Sets the y-component of the turtle's speed.

SHADE

SHADE

Shades the area of the screen containing the turtle with copies of its shape.

SHAPE

ASK FIRST WHO [PR SHAPE]

Outputs the turtle's shape number.

SHOW *object*

SHOW [HELLO THERE]

Prints its inputs followed by a carriage return (does not remove brackets).

SHOWNP

IF SHOWNP [HT]

Outputs TRUE if the turtle is in a ST state.

SIN *degrees*

PR SIN 1.1417

Outputs the sine of the input (in degrees).

SNAP

SNAP

Replaces the shape of the current turtle with the graphic image under the turtle.

SPEED

PR SPEED

Outputs the turtle's speed.

SQRT *number*

PR SQRT 4

Outputs the square root of the input number.

ST

ST

Makes the turtle visible.

STAMP

STAMP

Stamps a copy of the current turtle's shape on the screen.

STARTUP *instructionlist*

MAKE "STARTUP [BEGIN]

SAVE "MYFILE

Signals SmartLOGO to automatically run the value (an instruction list) of the variable called STARTUP when a file is loaded.

STARTUP (file)

SAVE "STARTUP

Signals SmartLOGO to automatically load the file called STARTUP when SmartLOGO is booted.

STOP

IF HEADING = :DEGREES [STOP]

Stops the current procedure. Used within a procedure only.

SUM *a b*

PR SUM 6 4

Outputs the sum of the two inputs (adds).

TELL *turtlenumber turtlenumberlist*

TELL [0 6 21]

Makes the list of current turtles.

TEXT *name*

TEXT "SQUARE

Outputs the definition of the named procedure as a list.

THAW

THAW

Following a FREEZE, resets previous turtle speeds.

THING *name*

PR THING "MYNAME

Outputs the contents of the variable.

TO *name*

TO SQUARE

Invokes the Editor.

TOOT *voice freq volume duration*

TOOT 2 440 15 180

Sound primitive with 4 inputs (0 to 2, 128 to 9999, 0 to 15, 0 to 255).

:TOUCHINGP *turtlenumber*

IF TOUCHINGP 1 [CRASH]

Outputs TRUE if there is a collision between any of the current turtles and any turtle given as input.

TOWARDS *position*

SETH TOWARDS [20 20]

Outputs the heading from the turtle towards the input position.

TRUE

Special input for AND IF NOT or OR.

TYPE *object*

TYPE "HELLO

Prints the input (a word or a list) with no carriage return following.

WAIT *number*

WAIT 60

Causes a pause in operation, in 60ths of a second.

WHEN *condnumber instructionlist*

WHEN 0 [FD 1 RT 1]

Sets a demon to wait for the specified event, and then run the instruction list.

WHO

PR WHO

Outputs the list of current turtle.

WINDOW

WINDOW

Sets a graphic mode in which the screen is a "window" onto a much larger area. See WRAP.

WORD *word word*

PR WORD "BIG "WORD

Outputs the inputs as one word, with no spaces.

WORDP *object*

PR WORDP "HI

Outputs TRUE if the input is a word.

WRAP**WRAP**

Sets a graphics mode in which the turtle wraps around the screen if it leaves the visible screen area. This is the default screen mode. See WINDOW.

XCOR**PR XCOR**

Outputs the turtle's x-coordinate.

XVEL**PR XVEL**

Outputs the x-axis component of the turtle's speed.

YCOR**SETY (YCOR + 30)**

Outputs the turtle's y-coordinate.

YVEL**SETYVEL -(YVEL)**

Outputs the vertical component of the turtle's velocity.

()**IF 1 + (WHO * 3) = 0 [CS]**

Round brackets group for clarity and to impose a sequence of operations.

 $a + b$ **PR 8 + 2**

Outputs the result of the arithmetic operation (infix).

 a / b **PR 14 / 2**

Outputs the result of the arithmetic operation (infix).

 $a * b$ **PR 3 * 5**

Outputs the result of the arithmetic operation (infix).

 $a - b$ **PR 12 - 6**

Outputs the result of the arithmetic operation (infix).

$a < b$

IF :SIDE < 0 [STOP]

Outputs TRUE if the first input is less than the second input.

$a > b$

IF XCOR > 100 [SETX 0]

Outputs TRUE if the first input is greater than the input.

$a = b$

IF HEADING = :DEGREES [STOP]

Outputs TRUE if the inputs are equal.

Dangerous Primitives

These primitives may destroy data. Use with caution.

.ALLOCATE *byte*

Reserves the specified amount of bytes of memory for special use.

.CALL *address*

Calls the machine language subroutine at the address of the input.

.CONTENTS

Outputs a list containing names and other words that have been typed.

.DEPOSIT *address byte*

Stores the value of the second input at the specified address.

.EXAMINE *address*

Outputs the byte stored at the specified address.

.INITIALIZE *word number*

.INITIALIZE "LOGOFILES 2

Dangerous. Redivides the directory and gives it the name specified. All files will be lost.

.PRIMITIVES

.PRIMITIVES

Prints out a list of all primitives.

G.

address

The location of a register, a particular part of memory, or some other data source or destination.

allocate

To assign a resource, such a disk file or a part of memory, to a specific task.

alphanumeric

Information containing both numeric and alphabetic characters.

ASCII

(American Standard Code for Information Interchange)
The standard code used for exchanging information about data processing systems and associated equipment.

binary

Something that has two possible values or states. Also refers to the base 2 numbering system.

bit

A binary digit.

boot

The process of loading a language or application program into the computer's memory, as in when you start up Logo.

buffer

An area of memory for temporary storage of data used when transferring data from one device to another. Buffer usually refers to an area reserved for an input output operation into which data is read or from which data is written.

bug

An error in a program.

byte

Eight bits.

call

To bring a computer program, a procedure, or a subprocedure into effect.

carriage return

A character which causes the cursor to move to the first position on the line.

case

The form of letters being used. For example, CANADA is printed in uppercase letters.

catalog

A table on a digital data pack of the names of all the files on that digital data pack, along with information that tells the operating system where to find the files.

character

A letter, digit, or other symbol that is used as part of the organization, control, or representation of data.

command

A Logo procedure, either a primitive or one that you define, that has no output. CLEARSCREEN, FORWARD, and PRINT are examples of commands. See **operation**.

conditional

A statement that causes Logo to carry out different instructions, depending on whether a condition is met.

coordinates

Numbers which identify a location on the display screen.

cursor

A movable marker that is used to indicate a position on the display screen.

debug

To find and eliminate mistakes in a program.

default

A value or option that is provided by the program when none is specified.

delete buffer

The portion of the computer's memory which is reserved for last line of text entered, or stored by pressing the

CLEAR key.

demon

A parallel procedure which can be set to continually watch for a specific event or collision between turtles. When the collision or event occurs, any currently executing procedure is interrupted and the demon's instruction list is run.

device

Anything attached to the computer, such as a printer, video display, or data drive.

diskette (or floppy disk)

A removable magnetic file storage medium. Diskettes are 5¼" in diameter and are encased in protection sleeves.

edit

To enter, modify, or delete data.

edit buffer

The portion of the computer's memory that contains all the text that is in the Logo Editor.

element

A member of a set in particular, an item in a series.

empty list

A list that has no elements. You write the empty list as [].

empty word

A word that has no characters. You write the empty word as ""

erase

To remove information permanently from either the workspace or a file.

envelope

The changes in the volume of a sound as it occurs; if a sound starts out loud and becomes gradually quieter, that is its envelope.

execute

To perform an instruction or a computer program.

file

An organized collection of information that can be permanently stored for specific purposes.

garbage collection

Cleaning the computer's memory to make more space available for storage.

global variable

A variable that is always in the workspace, such as a variable you create with the MAKE primitive. See **local variable**.

grammar

The rules by which Logo instructions are written.

hard copy

A printed copy of machine output in a visually readable form.

infix notation

A way of expressing an arithmetic operation where the operation symbol is placed between the two numerical inputs. See **prefix notation**.

input

The information that a Logo primitive or procedure needs to begin execution.

instruction

In a programming language, any meaningful expression that specifies one command and its inputs.

integer

A positive or negative number that does not contain any fractional parts.

interactive

A program that creates a dialogue between the computer and the user.

interrupt

To stop a process in such a way that it can be resumed.

K

When referring to storage capacity, two to the tenth power or 1024 in decimal notation.

line feed

A character that causes the print or display position to move to the corresponding position on the next line.

list

A collection of Logo objects, a sequence of words or lists that begins and ends with brackets.

literal word

An explicit representation of a value, especially the value of a word or list. A literal word is preceded by the quotation mark character (").

local variable

A variable that exists only when a procedure is being executed. See **global variable**.

location

Any place in which data may be stored.

logical operation

A predicate whose input must be either TRUE or FALSE.

menu

A list of choices displayed on your screen from which you select an action or setting.

name

A word used as a container for a value in the workspace.

node

A division of your workspace. Each node is five bytes long.

object

A word, a list or a number.

operation

A Logo operation, either a primitive or one that you define, that has some kind of output, SUM, POS are examples of operations. See **command**.

output

The information that a Logo primitive or procedure gives to another primitive or procedure.

parse

The process by which phrases are associated with the component names of the grammar that generated the string. In Logo, to make sense out of a Logo line.

predicate

A procedure that outputs either TRUE or FALSE.

prefix notation

A way of expressing an arithmetic operation where the operation symbol or primitive is placed before the numerical inputs. See **infix notation**.

primitive

A procedure that is built into Logo.

procedure

A single instruction or a sequence of instructions to Logo, which has a name and can be permanently stored.

procedure call

A request to execute a named procedure. You call a procedure either from the top level or from within another procedure.

program

A set of procedures that work together.

prompt

A question the computer asks or a signal it displays when it wants you to supply information.

property list

A list consisting of an even number of elements. Each pair of element consist of a property (such as I.D.) and its value, a word or list (such as Robin).

real number

Any positive or negative decimal number.

recursive procedure

A procedure that calls itself as a subprocedure.

reserved

An area of the computer or a name having a restricted use.

scientific notation

The expression of numbers using an exponent.

scroll

To move all or part of the display image vertically so that new data appears at the bottom and old data disappears from the top.

subprocedure

A procedure used in the definition of another procedure.

superprocedure

A procedure that calls another procedure.

syntax

The rules governing the structure of a language.

top level

The mode in which commands can be executed directly without being embodied in a program.

truncate

To remove the ending elements from a word. For a number, to remove the fractional part.

value

The contents of a variable.

variable

A container that holds a value and has a name.

word

A series of characters with no spaces treated as a unit.

workspace

The part of the computer's memory that holds variables, procedures, and properties only as long as the computer is turned on.

Every effort has been made to ensure the accuracy of the product documentation in this manual. However, because we are constantly improving and updating our computer software and documentation, Logo Computer Systems Inc. is unable to guarantee the accuracy of printed material after the date of publication and disclaims liability for changes, errors or omissions.

No reproduction of this document or any portion of its contents is allowed without the specific written permission of Logo Computer Systems Inc.

© 1984 Logo Computer
Systems Inc.
All rights reserved.

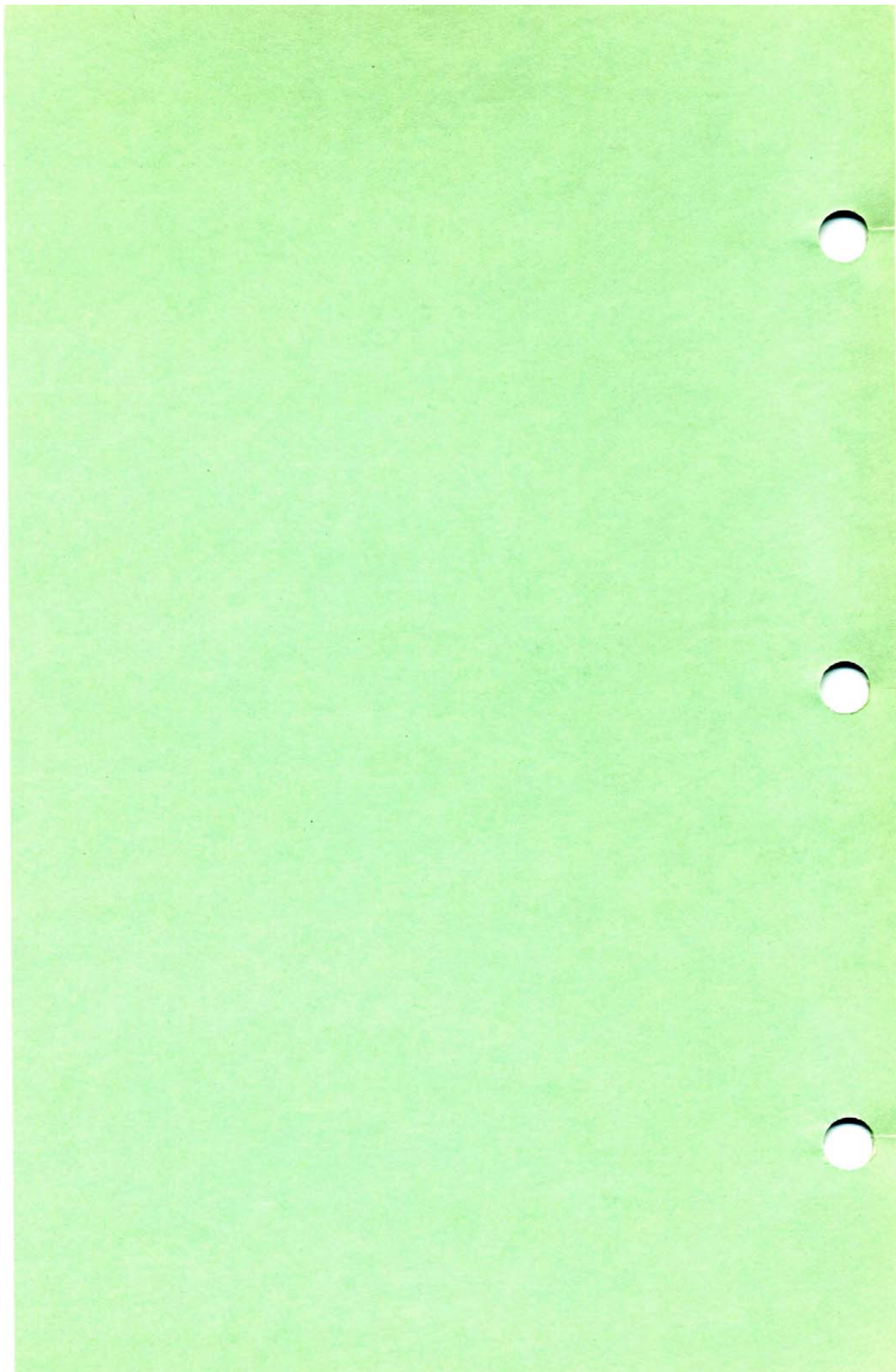


Printed in U.S.A. 14381



Logo Computer Systems Inc.
9960 Côte de Liesse Road
Lachine, Québec, Canada H8T 1A1

INDEX



Index

T indicates a page in Turtle Talk.

R indicates a page in the SmartLOGO Reference Manual.

App. indicates a listing in an Appendix.

* T43, R157
+ T43, R155
- T43, R156, App.E
/ T44, R158
\ R12, R105, App.E
: T54, R11, R24
" T54, R11
[R11, R24
] R11, R24
= R133, R159
> R159
< R160
(R12
) R12
← key T18, R9, R36, R71
→ key T18, R9, R35,
R36, R71
↑ key T18, R9, R36, R71
↓ key T18, R9, R36, R71

A

ALLOCATE R228
ABS R127, App.B
active turtle R85
addition R155
AGE R194
ALARM R112
ALL R86
AND R180
ARCCOS R148
ARCL App.B
ARCR App.B
ARCSIN R148
ARCTAN R148
arctangent R148
ASCII R195
ASK R87

B

BACK, BK T7, R43
 BACKGROUND, BG R58
 background color R57,
 R58
 backslash R12, R105
BACKSPACE key R10,
 R36
 BASSOON R112
 BETWEEN R160
 BIRDS T54
 booting R6
 BOUNCE T66, R81
 BOX R6
 brackets, [] R11
 buffer R10, R35, App.G
 BURST R86
 BUTFIRST, BF R120
 BUTLAST, BL R121
 BUZZER R112
 BYE R39
 byte R228

C

CALL R228
 .CONTENTS R230
 C R109
 CALCULATOR R171
 CAPITAL R140
 CARS T33
 CASTLE App.B
 CATALOG T60, R210
 CHANGE.COLOR R58
 CHANGE BG R61
 CHAR R196
 character R8, R194
 CHECK R181

CHORD R108
 CIRCLEL App.B
 CIRCLER App.B
 CLEAN R93
CLEAR key T47, R10,
 R35, R37
 CLEARGRAPHICS, CG R94
 CLEARSCREEN, CS T8, R94
 CLEARTEXT, CT T61, R94
 CLOCK R48
 CODE R195
 collision R168
 colon (:) T54, R11, R24
 COLOR T37, R59
 color, background R57,
 R58
 color, list R58
 color, pen R57, R60
 color, turtle R57, R59
 COLOR.OVER R59
 COLORSPI R191
 command T8, R25
 COMMENT R121
 COMMENT (:) App.B
 computer **RESET**
 button T5, R7
 COND R165
 conditionals R163
CONTROL key R70
 CONVERT App.B
 COPYDEF R39
 COPYSH R72
 COS R149
 cosine R149
 COUNT R122
 COUNTDOWN R174
 CREATE.TURTLE R224
 CREATE.STATE R224

current turtle R85
CURSOR R101
cursor motion R35, R36,
R70
CURVES App.B

D

.DEPOSIT R230
D6 R152
DEC R229
DECTOHEX App.B
DECIDE1 R167
DECIDE2 R167
decimal number R145
DECIMALP R181
DECISION1 R166
DECISION2 R166
DECISION3 R166
DEFINE R39
DEFINEDP R41
defining
procedures R33
delete buffer R10
DELETE key T25, R10,
R36
delimiter R116, App.E
DEMO App.B
demon R163
DEVICE R199
DIFFERENCE R149
Digital data pack T5,
App.B
DIMENSION R330
DISTANCE R44
division sign, / R158
DIVISORP App.B
DOT R44

DRAW R176, R188
DRIVE R192

E

.EXAMINE R231
E R109
EACH T40, R88
EASYSHAPE App.B
EASYTYPE App.B
EDFILE R210
EDGE T32
EDIT, ED R38
edit buffer App.G
EDITSHAPE, ES T18, R72
EDNS R138
element R116
empty list R117
empty word R117
EMPTYP R123
END R38
EQUALP R124
equal sign R133
ERALL R202
ERASE, ER T27, R202
ERASEFILE, ERF R210
ERCS R165
ERDS R165
ERES R165
ERN R202
ERNS R203
ERPROPS R203, R219
ERPS R203
error message App.A
ESCAPE/WP key T39, R9,
R38, R70
EVENP R153
events R176

EXP	App.B	global variable	R28
EXPLORE	App.B	GROWSQ	R190
EXPLOSION	R112	GPROP	R219
exponent	R145	greater than, >	R159
exponential form	R145	GREET	R24, R34

F

FACTORIAL	R158
FALSE	R181
file name	R209
files	R209
FILL	T65, R44
FIRST	R125
FLAG	T25
FLAP	R48
FLIP	R26
flow of control	R163
FLY.AND.DRIVE	R81
FLYPAST	R112
FOLLOW	R55
FOREVER	R173, App.B
FORWARD, FD	T6, R45
four-turtles-in-a-line rule	R18
FPUT	R125
FREEZE	T66, R80

G

G	R109
game controller	R187
garbage collector	App.D
GETARRAY	R231
GET.USER	R193
GETSH	T61, R72

H

HANGMAN	App.B
HEADING	R46
HEXTODEC	App.B
HI	R39
HOME	T10, R46
HOME key	T18, R36, R70
HOUSE	R6
HT	T10, R74

I

.INSTALL	R229
IF	R166
IGNORE	R192
IN.WINDOWP	R96
INC	R142
INIT.TURTLE	App.B
INITIALIZE	R214
input	T8, T26, R14, R23, R119
INSERT key	R10, R36
instruction	T26, R85, R164
instruction list	R164
INT	R150
integer	R145
INTP	R150
invisible color	R57
ITEM	R126

J

JOY R188
JOYP R188
joystick R187

K

keyboard R8, R191
KEYP R191

L

LANES T33
LAST R127
LATIN R132
LBUTTONP R189
LEFT, LT T7, R47
less than, < R160
line buffer R29, R35
LIST R127
LISTP R128
LN App.B
LOAD T60, R210
LOADPICT T60, R211
local variable R28
LOG App.B
Logo line R29
Logo object R14, R115
LOST.IN.SPACE R97
LOWERCASE R197
LPUT R128

M

MAKE R138
MAKE.T.STATE R223
MAP R172
MARK.TWAIN R168, App.B
MEMBER R129

MEMBERP R129
memory App.D
MESSAGE R194
minus sign, - R156
MOUNTAINS R184
MOVE R105
MOVE/COPY key R10,
R35, R37
multiplication sign, *
R157

N

NAME T51, R140
NAMEP R141
NEWENTRY R128
NEWROAD T32
NEXT T38
node R203, App.D
NODES R203
NOISE R110
Noise Types R111
NOPRINTER T34, R198
NOT R182
number R115, R145
NUMBERP R130

O

Object
Manipulators R117
Object Reporters R118
OFFSOUND R112
ON.TOUCH T66, R168
operation T44, R24, R117
OR R183
order of math operations
R147

order of precedence
for turtles R16
OUTPUT, OP R168

P

.PRIMITIVES R231
PADDLE R190
parentheses () R12
PEN R47
PENCOLOR, PC R60
PENDOWN, PD T9, R47
PENERASE, PE T10, R48
PENREVERSE, PX R48
PENUP, PU T9, R48
picture files R209
PIG R132
PLIST R220
plus sign, + R155
PO R204
POALL T34, R204
POC R169
POCS R169
PODS R170
POE R170
POES R170
POETRY App.B
POFILE R211
POLY R34, R171, App.B
PONS R205
POPPOPS R219
POPS R206
POS R49
position R43, R101
POSITIVE? R167
POTS T34, R206
PPROP R220
PPS R207, R221

predicate R179
PREFIX R132
primitive R1
PRIMITIVEP R41
PRINT, PR T37, R102
PRINT.PROP R225
PRINTBACK R127
PRINTDOWN R125
PRINTER T34, R198
PRINTMESSAGE R35
PRODUCT R151
prompt R7, R34
PROMPT R105
property list R217
PUTARRAY R230
PUTSH T61, R74
PWR App.B

Q

quotation mark (") R11
QUOTIENT R151

R

RACE R60
RANDOM R152
RANGE.ERR R231
RANK R124
RANPICK R122
RBUTTONP R190
READCHAR, RC R192
READLIST, RL R193
READWORD R194
REALWORDP R183
recursion T39, R23
RECYCLE R207
REMAINDER R152
REMPROP R221

REPEAT T10, R170
 reporters (operations)
 T37
 REPRINT R102
 RERANDOM R153
 RETURN key T6, R8, R35,
 R36
 REV R126
 RIGHT, RT T8, R50
 ROAD T31
 ROCKET R112
 ROUND R154
 RUN R171
 RUNNER R189
 RW R194

S

SAVE T59, R212
 SAVENS T61, R212
 SAVEPICT T60, R213
 SAVEPROPS R213
 SAVEPS R213
 SCRUNCH R98
 SECRETCODE R195
 SEE.CHAR R196
 SENTENCE, SE R131
 SETBG R61
 SETCOLOR, SETC T14, R61
 SETCOURSE R189
 SETCURSOR R103
 SETDEVICE R199
 SETHEADING, SETH R50
 SETPEN R51
 SETPENCOLOR, SETPC T15,
 R61
 SETPOS R52
 SETSCRUNCH, SETSCR R98

SETSHAPE, SETSH T14, R74
 SETSPEED, SETSP T13, R80
 SETTEXT R95
 SETWIDTH R95
 SETX R52
 SETXVEL R81
 SETY R52
 SETYVEL R82
 SHADE R53
 SHAPE T39, R75
 Shape list R65
 SHIFT key R8
 SHOOT R112
 SHOW R103
 SHOWNP R75
 SIMON App.B
 SIN R154
 SLEEP R174
 Smart Key V R9
 Smart Key VI T19, R9,
 R37, R71
 SmartLOGO Editor
 T24, R33
 SmartLOGO Shape
 Editor T18, R70
 SmartWRITER printer
 R197
 SNAP R76
 SOMEWHERE R169
 SORT App.B
 SPACEBAR R8
 SPACE?1 R196
 SPACE?2 R196
 SPEED T39, R82
 SPI R205
 SPIRAL R192
 SQRT R154
 SQUARE T23, R40

ST T10, R76
STAMP R54
STAR T26
STARS R131
startup T5, R7, App.C
STARTUP R173, App.C
STOP R173
SUBMOUNTAIN R184
subprocedure T31, R23
SUBSET R169
SUM R155
superprocedure R23

T

Table of Frequencies

R110
TALK R123
TAN R149
TELL T15, R89
TEST R60
TEXT R40
THAW T66, R82
THING R141
THROW.TOPLEVEL R229
TO R38
TOOLS App.B
TOOT T49, R108
top level R22
total turtle trip
theorem T30
TOUCHINGP R174
TOWARDS R54
TRACKS R54
transparent color R57
TRI R6
TRIANGLE R120

TRUE R184
TURN R193
turtle field R97
TURTLE.STATE R223
tutorials R7, App.B
TYPE R104

V

variables R26, R119, R137
VISIBLEP R182
VOWELP R130

W

WAIT T38, R174
WARMWELCOME R22
WELCOME R22
WHEN R175
WHILE R172, App.B
WHO T40, R90
WINDOW R96
word R14, R115
WORD R132
WORDP R133
workspace R201, App.A
WOW T39
WRAP R97

X

XCOR R55
XVEL R82

Y

YCOR R55
YVEL R82

TURTLE TATTLE

It's Okay!

SmartLogo™ is a wonderful programming language that is easy to learn. It lets you teach the computer how to do many exciting things. Like most programming languages, SmartLogo™ has certain quirks or limitations. At first you may think that either you or the computer is doing something wrong. This may not be the case. Of course, you should always check your work to be sure you haven't made any careless mistakes. If your work looks okay, you may have found one of SmartLogo™'s limitations. Here are some examples of quirks:

SETSP used in combination with PENREVERSE can produce spotty lines, or no lines.

Depending on the shape of an object, FILL may cause color to spill out of the boundary lines.

If you fill a shape with the same color as the screen background, the boundary lines may be erased.

Show Me!

The DEMO programs were included to show you some of SmartLogo's features, and to give you some ideas as to the kinds of programs you can write. These demos are not intended to be fully-developed game programs, and therefore, may have some limitations.

For example, you should always use the keyboard when playing the SIMON game. If you're really good, and Simon must play a 10–15 note sequence, the musical notes may sound before the square flashes. In the HANGMAN game, when you have more than 10 guesses, two sets of dashes will appear on the screen.

Oops, We Goofed!

On menu #2, screen 4 of the on-line tutorials, Instructions is missing the "r."

Some of the procedures in the TOOLS file need to be corrected before they will run properly. The procedures that need to be corrected are listed below in their corrected form. The lines that have been changed are printed in green. See Chapter 3 in the Reference Manual if you need help making the changes.

To SAVE a copy of your corrected TOOLS procedures, see page 212 of the Reference Manual.

The first screen of the EASYTYPE procedure displays the line beginning "You can stop after drawing a complete shape." This line should be omitted.

```
TO EFRAC :FRAC :COUNT :TERM
IF :COUNT > 9 [OP 0]
MAKE "TERM :TERM * :FRAC / :COUNT
OP :TERM + EFRAC :FRAC :COUNT + 1 :TERM
END
```

```
TO EXP :X
MAKE "E 2.71828
IF :X - (INT :X) = 0 [OP INTPWR :E :X]
OP ( INTPWR :E INT :X ) * ( 1 + EFRAC ( :X - INT :X ) 1 1
END
```

```
TO BEFORE :A :B
IF OR EMPTYP :A EMPTYP :B [OP EMPTYP :A]
IF NOT EQUALP FIRST :A FIRST :B [OP ( ASCII :A ) < ( ASCII :B )]
OP BEFORE BF :A BF :B
END
```

```
TO WHILE :CONDITION :INSTRUCTIONLIST
IF NOT RUN :CONDITION [STOP]
RUN :INSTRUCTIONLIST
WHILE :CONDITION :INSTRUCTIONLIST
END
```

```
TO SORT :ARG :LIST
IF EMPTYP :ARG [OP :LIST]
MAKE "LIST INSERT FIRST :ARG :LIST
OP SORT BF :ARG :LIST
END
```

**Quick
Reference
Guide**

LOGO

Editing Procedures

TO name
Invokes the Editor.

EDIT, ED name
EDIT namelist
Invokes the Editor (identical to **TO** but can take a list).

END
Ends the procedure definition started by **TO**.

Naming Things (Variables)

NAME object name
Sets a *global variable*. Gives the second input (the name) the first input as value. The value remains until reset or erased.

MAKE name object
Sets a *global variable*. Gives the first input (the name, quoted) the second input as value. The value remains until reset or erased.

THING name
Outputs the contents of the variable.

GETSH number
Outputs a list of 32 numbers representing the turtle shape. See **PUTSH**.

PUTSH number shapespec
Gives the named shape number the form defined by the second input (a list of 32 numbers). See **GETSH**.

Flow of Control

REPEAT number instructionlist
Runs the input instruction list the specified number of times.

IF pred instructionlist
IF pred instructionlist instructionlist
If the first input outputs **TRUE**, the first instructionlist is run. If **FALSE**, the optional second instructionlist is run.

STOP
Stops the current procedure. Used within a procedure only.

OUTPUT, OP object
Stops the current procedure and outputs (a list, word, **TRUE** or **FALSE**). Used within a procedure only.

Words and Lists

FIRST object
Outputs the first item of the input (a list or word).

LAST object
Outputs the last item of the input (a list or a word).

ITEM number list
Outputs the first input item of the second input (a list or a word).

COUNT object
Outputs the number of items in the input (a list or a word).

SENTENCE, SE object object
Makes the inputs into a single list; removing inner brackets (see **LIST**).

LIST object object
Outputs a list composed of its inputs, including a list of lists (see **SE**).

WORD word word
Outputs the inputs as one word, with no spaces.

Turtle Graphics

FORWARD, FD number

Moves the turtle forward the input number of steps.

BACK, BK number

Moves the turtle back the input number of steps.

RIGHT, RT degrees

Turns the turtle right the input number of degrees.

LEFT, LT degrees

Turns the turtle left the input number of degrees.

HOME

Sends the current turtle to the center of the screen [0 0] with a heading of 0.

SETPOS position

Sets the turtle's screen position (absolute).

POS

Outputs the screen position of the current turtle.

PENDOWN, PD

Sets the current turtle's pen to draw lines.

PENUP, PU

Raises the current turtle's pen.

PENERASE, PE

Sets the current turtle's pen to erase any lines crossed.

PENREVERSE, PX

Sets a pen mode which draws lines unless over a drawn line; in that case it erases the line.

SETPENCOLOR, SETPC number

Sets the color of the current turtle's pen.

PENCOLOR, PC

Outputs the current turtle's pen color.

DOT position

Puts a dot at the specified position.

FILL

Fills an area with the current turtle's pen color.

SHADE

Shades the area of the screen containing the turtle with copies of its shape.

STAMP

Stamps a copy of the current turtle's shape on the screen.

HT

Makes the current turtle invisible.

ST

Makes the turtle visible.

SETCOLOR, SETC number

Sets the color of the current turtle.

COLOR

Outputs the turtle's color.

SETSHAPE, SETSH number

Sets the turtle's shape, (0 to 59).

SHAPE

Outputs the turtle's shape number.

SETHEADING, SETH degrees

Sets the heading of the current turtle, in degrees (absolute).

HEADING

Outputs the turtle's screen heading, in degrees.

Clearing the Screen

CLEARGRAPHICS, CG

Clears graphics and sends the current turtle HOME.

CLEARTEXT, CT

Clears text from screen.

CLEAN

Cleans the graphics screen without changing the turtle state.

CLEARSCREEN, CS

Clears the screen of text and graphics, sends the current turtle HOME.

Moving

SETSPEED, SETSP

Sets the turtle's speed, (- 128 to 128).

SPEED

Outputs the turtle's speed.

FREEZE

Suspends turtle speed until THAW.

THAW

Following a FREEZE, resets previous turtle speeds.

Turtles

TELL turtlenumber

TELL turtlenumberlist

Makes the list of current turtles.

ALL

Outputs the numbers 0 through 29.

EACH instructionlist

Causes a sequential run of the instruction list by each current turtle.

WHO

Outputs the list of current turtle.

Demons

ON.TOUCH turtlenumber turtlenumber instructionlist
Sets a demon to wait for a collision between the two input turtles, then runs the instruction list.

WHEN condnumber instructionlist

Sets a demon to wait for the specified event, and then run the instruction list.

ERDS

Erases all demons (collisions and events).

PODS

Prints out all demons and their instruction lists.

This procedure restores the screen and the turtles to the original state.

TO RESTORE

ERDS

THAW

SETBG 5

TELL ALL

CS HT PD

SETSH 36

SETC 15 SETPC 15

TELL 0 ST

END

Making Sounds

TOOT voice freq volume duration

Sound primitive with 4 inputs (0 to 2, 128 to 9999, 0 to 15, 0 to 255).

NOISE type startvol stepvol steps steplength

Sound primitive with 5 inputs (0 to 7, 0 to 15, -7 to 7, 0 to 15, 0 to 15).

Demons

ON.TOUCH *turtlenumber turtlenumber instructionlist*
Sets a demon to wait for a collision between the two input turtles, then runs the instruction list.

WHEN *condnumber instructionlist*
Sets a demon to wait for the specified event, and then run the instruction list.

ERDS
Erases all demons (collisions and events).

PODS
Prints out all demons and their instruction lists.

This procedure restores the screen and the turtles to the original state.

TO RESTORE

ERDS

THAW

SETBG 5

TELL ALL

CS HT PD

SETSH 36

SETC 15 SETPC 15

TELL 0 ST

END

Making Sounds

TOOT *voice freq volume duration*
Sound primitive with 4 inputs (0 to 2, 128 to 9999, 0 to 15, 0 to 255).

NOISE *type startvol stepvol steps steplength*
Sound primitive with 5 inputs (0 to 7, 0 to 15, -7 to 7, 0 to 15, 0 to 15).

Shape Editor

EDITSHAPE, ES *number*
Puts the input shape number in the shape Editor.

Printing

PRINT, PR *object*
Prints its input on the screen, followed by a carriage-return.

POTS
Prints out the titles of all procedures in the workspace.

PONS
Prints out all global variables and their values.

POALL
Prints out the entire contents of the workspace.

On the Screen

SETCURSOR *position*
Sets the text cursor to the specified text position.

SETTEXT *line*
Sets the topmost line for text (0 top, 23 bottom).

On the Printer

PRINTER
Turns on the output channel to the printer.

NOPRINTER
Turns off the output channel to the printer.