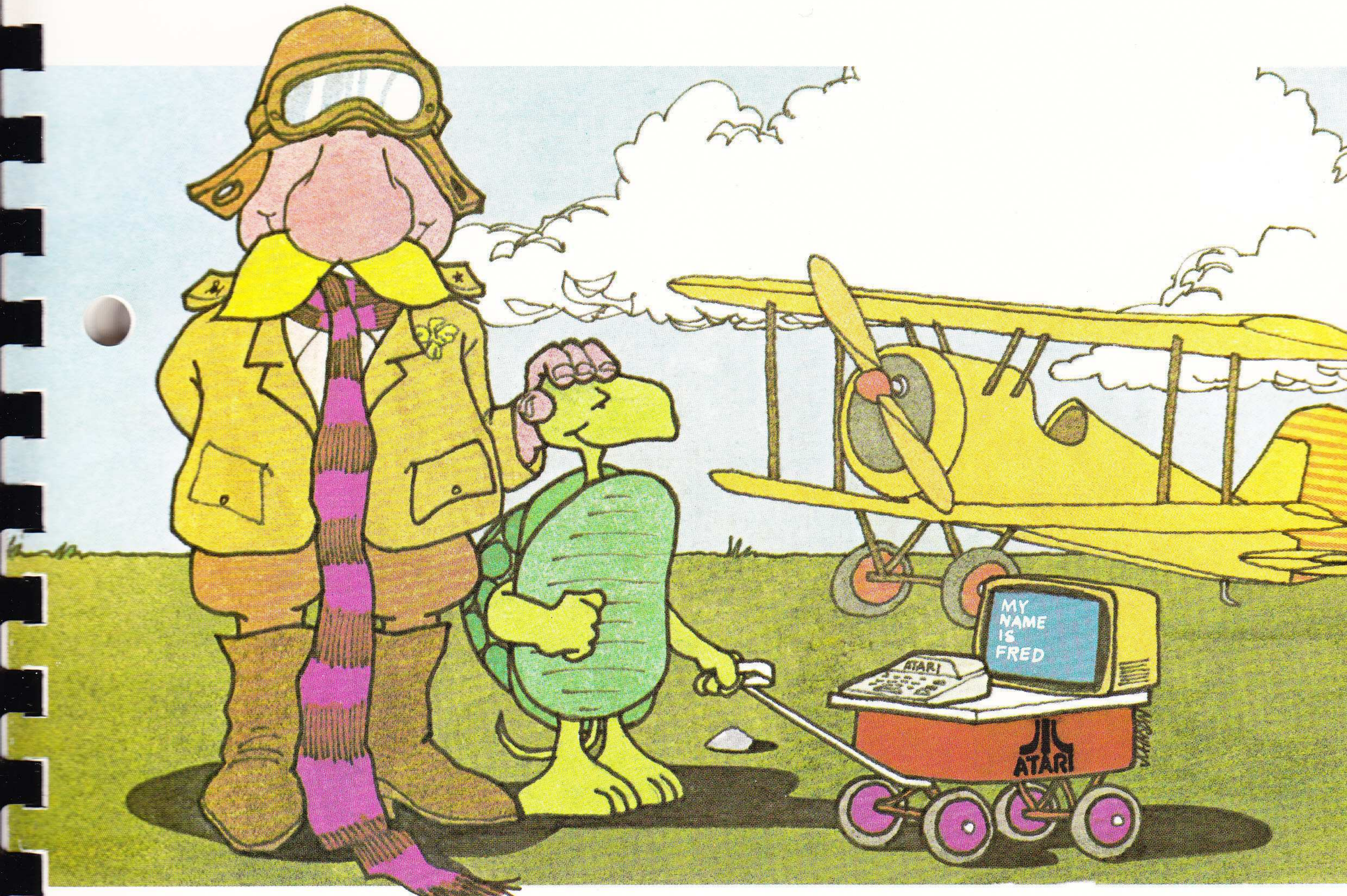
ATARI® 400/800™

STUDENT PILOT

REFERENCE GUIDE



ATARI®

A Warner Communications Company

Use with
ATARI® 400™ or ATARI 800™
PERSONAL COMPUTER SYSTEMS

STUDENT PILOT

Reference Guide

ATARI®

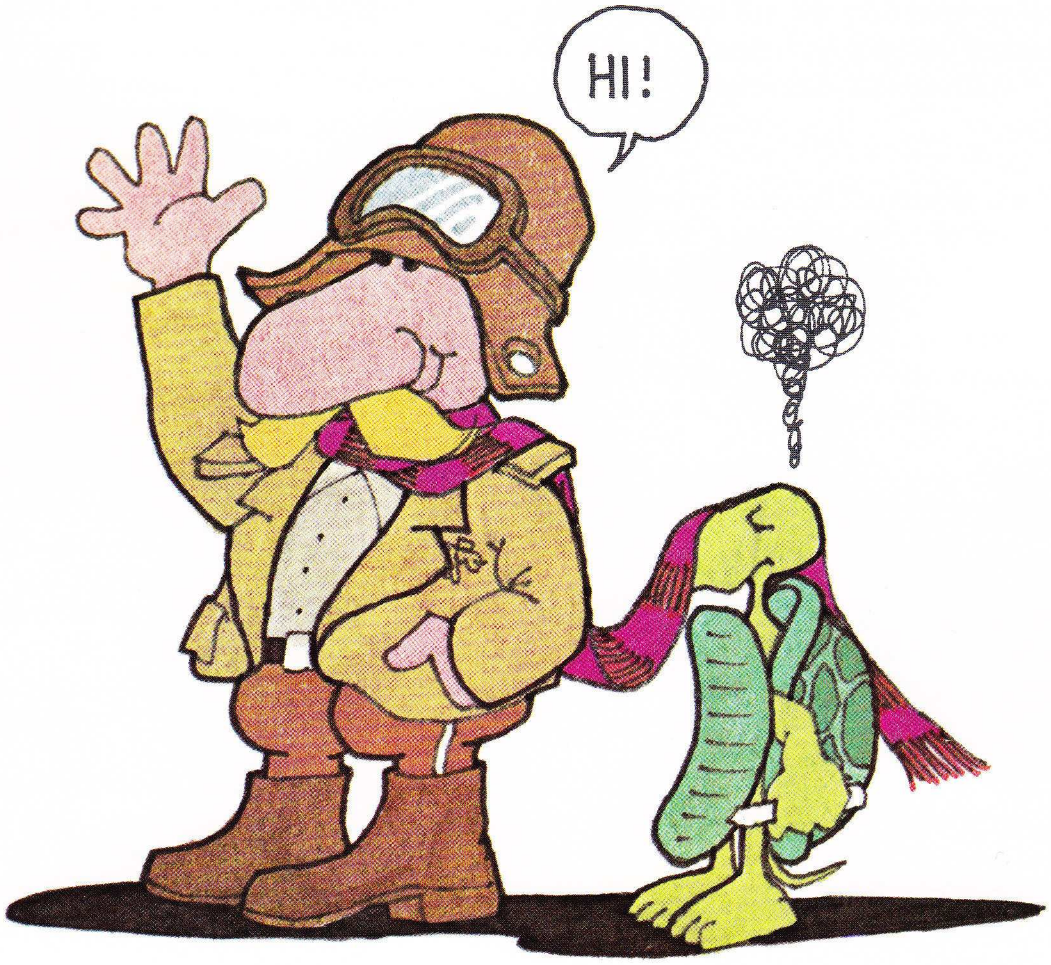
A Warner Communications Company

PREFACE

PILOT is a simple, easy-to-understand programming language. It stands for **P**rogrammed **I**nquiry, **L**earning **O**r **T**eaching, and was originally developed by Dr. John Starkweather at the University of California, San Francisco. PILOT was designed to enable teachers to write programs for their students. However, due to the pioneering work of Dr. Dean Brown at Stanford Research Institute's Education Laboratory (1967-1974), PILOT proved to be a good language for teaching computer programming to children. PILOT was conceived as a tool to make writing conversational programs easy. It was originally designed for text output only. However, we have incorporated **Turtle Graphics**, a concept developed by Dr. Seymour Papert and the LOGO group at the Massachusetts Institute of Technology. The Turtle Graphics concept enables the user to draw pictures when programming in PILOT. These capabilities, along with the ability to generate sounds, make **ATARI® PILOT** a very versatile language.

TABLE OF CONTENTS

INTRODUCTION



This reference guide is designed for your use if you are learning PILOT at home or in school, or if you already know PILOT and need a handy guide to the language. It gives brief explanations of all the main PILOT commands and examples of their use. Throughout this guide there are illustrations to enhance your understanding and enjoyment of PILOT. You are the pilot character; you give PILOT commands to the computer. Anytime you are in Graphics mode, the turtle character is ready to obey your commands. Each PILOT command is highlighted in large **bold** letters at the top of a page so you can easily find any command to which you wish to refer. This reference guide is divided into the following sections: PILOT Information, Core PILOT, Executive Commands, Graphics, Sound and Pause, and Conditionals.

In the first section, you will find terms to know and general information about the PILOT language. You should be familiar with the contents of this section in order to use this guide.

The second section reviews the Core PILOT commands. You put these commands together as a set of numbered statements to form a program. You can also execute them one at a time in Immediate mode, just as you would use a pocket calculator.

The third section reviews Executive commands. These commands allow you to do various things with programs as a whole. You give executive commands directly from the keyboard; **usually** they are not part of a program.

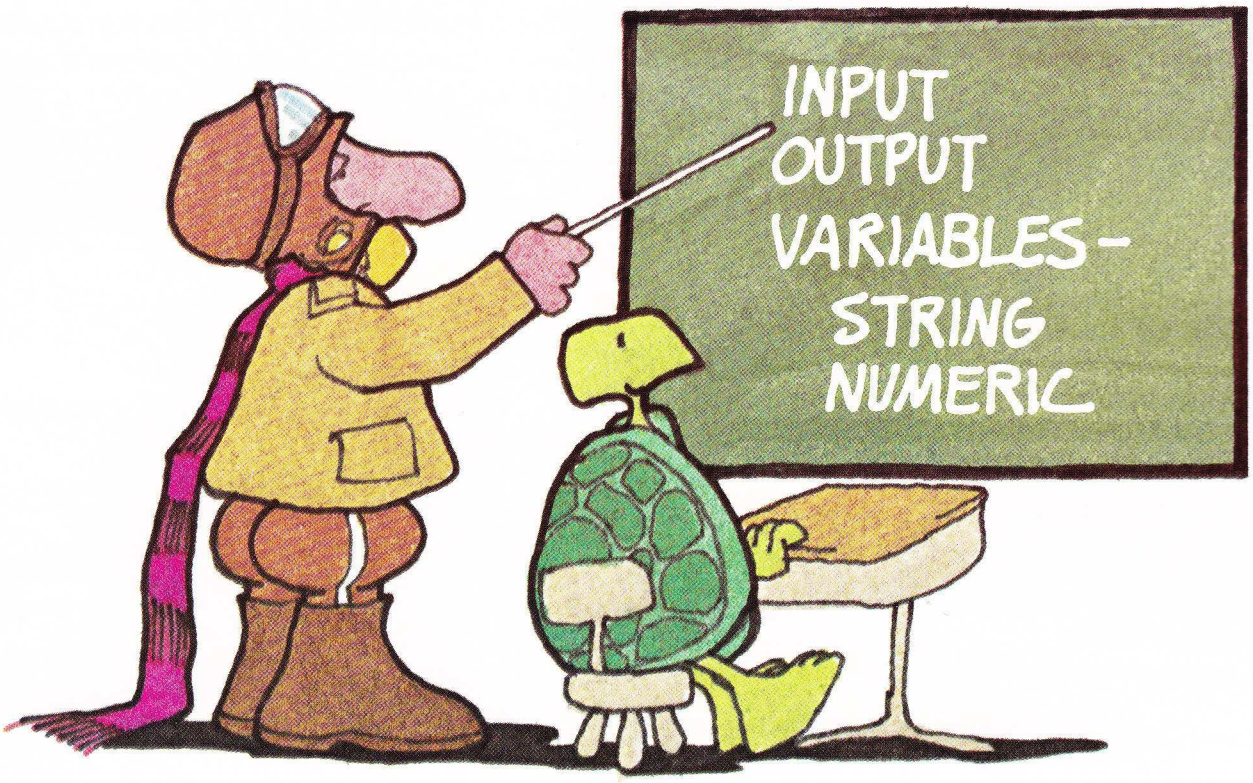
The fourth section covers turtle Graphics commands, which you use to draw designs or pictures on the screen. When PILOT is in graphics mode, an invisible turtle, holding an invisible pen, moves around the screen following your commands, drawing lines, and filling in spaces.

The fifth section has to do with the SOUND and PAUSE commands. SOUND is most frequently used when you want to make music, but can also help you create sound for a special effect. PAUSE lets you create a dramatic effect in a program, but it is used most often when creating music; for example, when you need to indicate how long a tone is to be played.

The last section discusses Conditional commands. Any PILOT command can be made conditional; that is, it will only be carried out if certain conditions are met. This section reviews the various kinds of conditions you can set up in your programs.

This is just a reference guide. If you need further explanation of any PILOT command, or if you wish to learn PILOT from a book, you should refer to the *PILOT Primer: An Instructional Manual for the PILOT Programming Language*.

1 PILOT INFORMATION



This section contains general information about PILOT. It includes explanations of terms often used in this guide, PILOT operating modes, variables, arithmetic and relational operators, and error messages.

TERMS

COMMANDS	Commands tell the computer what to do. There are two kinds of commands: Program Commands and Executive Commands. Program Commands are the building blocks of the PILOT language. They are composed of one or two letters followed by a colon (:). When placed in a logical sequence, these commands make up a program. Executive Commands tell the computer what to do with a program as a whole. Executive commands are either a short word such as RUN or an abbreviation such as DOS.
CONSTANT	A constant is a value expressed as a number rather than represented by a variable name. For example, in the statement: **C:#X = 100** #X is a variable, and 100 is a constant.
CURSOR	The white square that moves across the television screen showing your location on the screen as you type.
EXPRESSION	An expression is any combination of variables, constants, and arithmetic or relational operators used together to compute a value. For example, **#A + #B\ 2** is an expression; so is **#A<=#B**.
INPUT	Any information *put into* the computer from the keyboard, cassette tape, disk, and so on.
LABEL	Code name or symbols given to a statement in a program to identify that statement for reference by a J: or U: command. PILOT labels may be any combination up to 254 characters. Labels must start with an asterisk (*) and a letter; for example, * HERE or * M35PDQ.
MEMORY	The part of the computer where information is stored. This information is both program statements and data (constants and variables) that the program uses when it is run.

OUTPUT	Information *put out* from the computer memory to external devices.
PILOT	PILOT stands for Programmed Inquiry, Learning Or Teaching.
PROGRAM	A set of statements in a logical sequence that tells the computer what to do. The computer follows the program statements in order, from beginning to end, unless a J: JUMP or U: USE command breaks the sequence.
SCREEN DISPLAY	What you see on the television screen.
STATEMENT	A statement is one line of a program. All PILOT statements begin with line numbers (used for editing and sequencing). A PILOT statement may be labeled with a ✻ LABEL between the line number and the command.
STRING	A group of characters—letters, numbers, punctuation, or ATARI Graphics (CTRL) characters. The following are all strings: ABCXYZZZ "Quotes Too" 02BAK9 /\/\/\/\/\/\/ 5 + 2 = Note that quotation marks are not used to identify a string. Any group of characters, including quotation marks, is a string in PILOT. A string is like a constant, in that it may be stored in a string variable.
VARIABLE	A variable is the name of a place in memory to store a string or numeric value that is used by a program. This value may (or may not) change as the program is executed. (See the following page for an explanation of the different types of PILOT variables.)

OPERATING MODES

PILOT has two kinds of modes: *Programming Modes* which determine how PILOT commands are used by the computer; and *Screen Modes* which determine the layout of the display screen.

Programming Modes

AUTO	The mode in which the statements will automatically be numbered when writing a program.
IMMEDIATE	The Immediate mode is used when you want your commands to respond right away after pressing the RETURN key.
RUN	In Run mode, PILOT is executing (carrying out) the program currently stored in memory.

Screen Modes

GRAPHICS	The Graphics mode is used when you want to draw a picture of any kind.
TEXT	The Text mode displays only text (letters, numbers, punctuation) and graphics (CTRL) characters.

VARIABLES

PILOT has two main kinds of variables: **string variables** and **numeric variables**.

String Variables can store strings of up to 254 characters. They are named with a **$** symbol, followed by any combination of letters and numbers. For example, $NAME, $M29ABD, $FOOD9 are acceptable string variable names. String variables *cannot* be used in arithmetic expressions. A program can contain any number of string variables up to the limits of memory.

Numeric Variables can store numbers from -32768 through 32767. They are named with a # symbol, followed by a single letter; #A, #B, #C, and so on up to #Z are the acceptable numeric variable names. A program can have up to 26 numeric variables. They can be used in arithmetic and relational expressions.

Note: PILOT has two other kinds of variables that are *not discussed in this guide*. **Pointer** variables, named with an @ symbol followed by a decimal memory location, are used to examine or change the contents of a specific memory location (equivalent to PEEK and POKE in the BASIC language). **Special** variables, named with a **%** symbol followed by a letter or a letter and a number, are used to sense the values of the various ATARI controllers (joysticks, paddles, and light pen) and to read other special values maintained by the PILOT system.

The use of controllers and special graphics variables is explained in *The PILOT Primer, An Instruction Manual for the PILOT PROGRAMMING LANGUAGE*.

Information on the use of pointer variables and other special variables may be obtained by phoning ATARI, INC., Customer Support:

(800) 538-8547 (Outside California)
(800) 672-1430 (Within California)

ARITHMETIC OPERATORS

The ATARI PILOT language uses five arithmetic operators:

+	addition
-	subtraction (also denotes a negative numbers; e.g., -5)
✱	multiplication
/	division
\	modulo (the remainder after a division)

Note: PILOT only performs integer (whole numbers) arithmetic.

Parentheses ( ) can be used to change the order in which arthmetic is performed. For example C:#A=5+6 ✱ 2 puts the number 22 in variable #A; on the other hand, C:#A=5+(6 ✱ 2) puts the number 17 in variable #A.

RELATIONAL OPERATORS

PILOT allows the result of the comparison between two variables or constants to be used as a condition for executing a command. The comparison is made with the use of the following relational operators:

#A < #B	#A is less than #B
#A > #B	#A is greater than #B
#A = #B	#A equals #B
#A < =#B	#A is less than or equal to #B
#A > =#B	#A is greater than or equal to #B
#A < >#B	#A is not equal to #B

(See Section 6, Conditionals.)

ERROR MESSAGES

If you make an error, PILOT responds by showing the command or statement in error. It highlights the source of the error or the first character it does not understand. It also types out a message explaining the error.

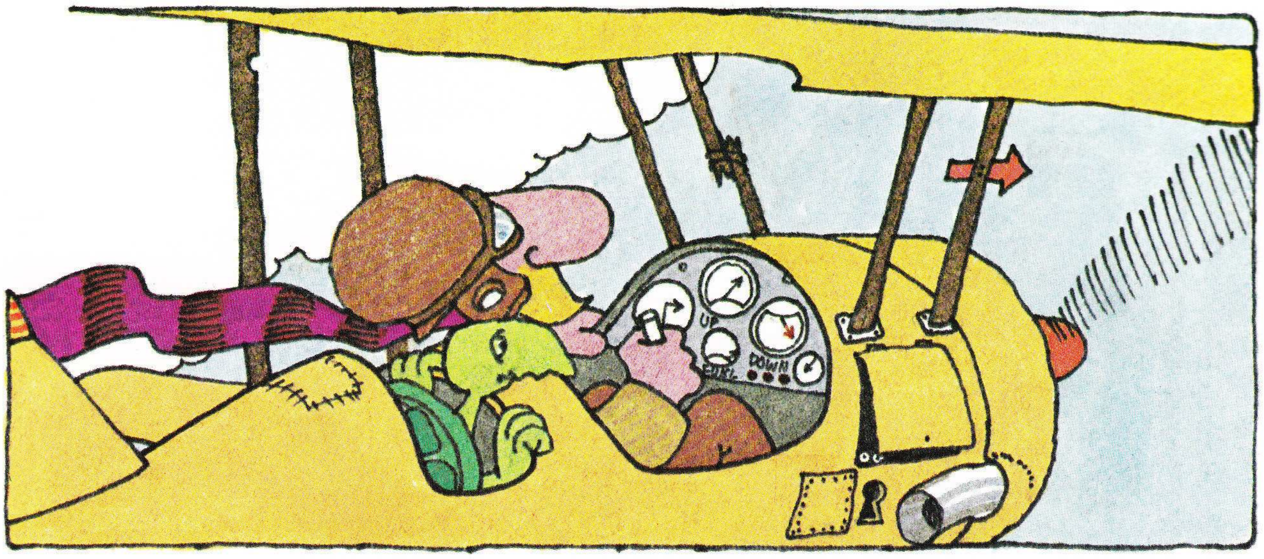
The are two kinds of errors: **Syntax errors** result from incorrectly typing in a command. Syntax errors are found by PILOT as you are typing commands or program statements into the computer.

The second type of error is a **Run-time error** which results from mistakes in the logic of your program - for example, trying to jump to a statement that you have not labeled. Run-time errors are only found when you type RUN and PILOT tries to execute your program.

The following are some common PILOT error messages:

WHAT'S THAT?	Indicates that PILOT could not understand your command or what you wanted it to do. This message is often the result of typing or spelling errors.
I/O ERROR xxx	In the course of performing an I/O operation the I/O subsystem detected an error and returned the status indicated. See Appendix B for a list of I/O error codes.
NO ROOM	Indicates that the requested operation could not be performed because there was not enough free memory.
WHERE?	Indicates that the * LABEL named in a U: (USE) or J: (JUMP) command does not exist in the program storage area.
U: TOO DEEP	The program has exceeded eight levels of nested USE commands.
DIVIDE BY 0	A COMPUTE command has just attempted to perform a division by zero.

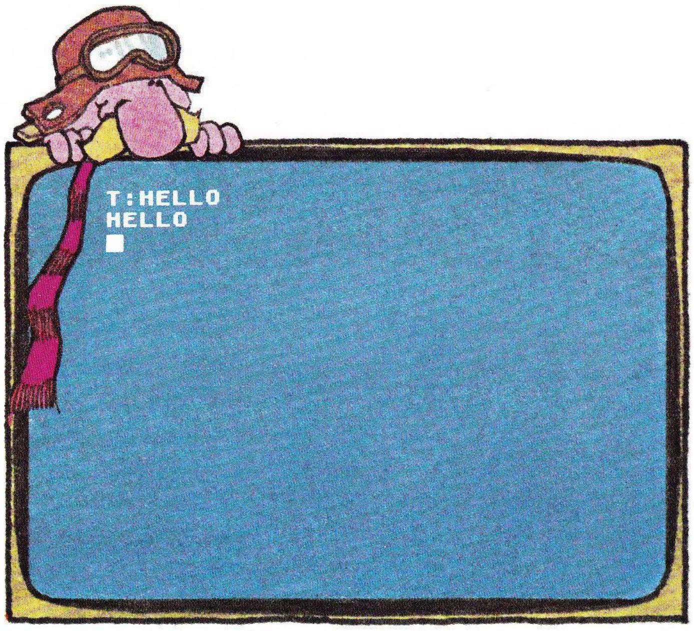
2 CORE PILOT



T:

Type

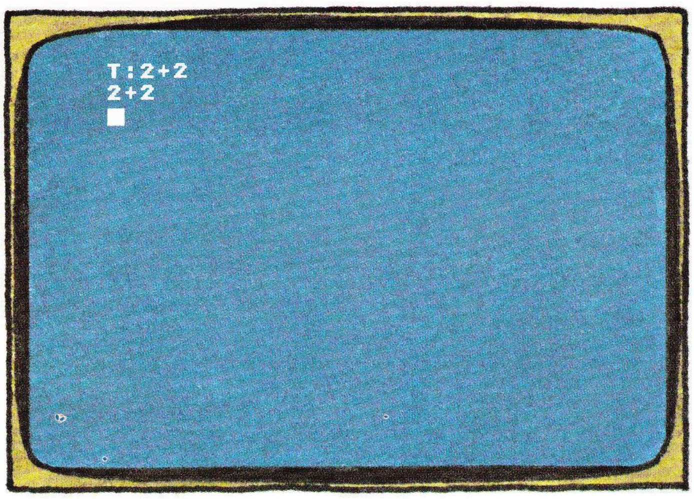
The T: (TYPE) command tells the computer to display what you type. You must type T: first, then your message. For instance, T:HELLO will cause the computer to respond *HELLO* as you see in Figure 2-1.



```
T:HELLO
HELLO
■
```

Figure 2-1

The T: command will display anything you can type from the keyboard *literally*. That is, the T: command will not add 2 + 2; it will display it as 2 + 2 (see Figure 2-2).

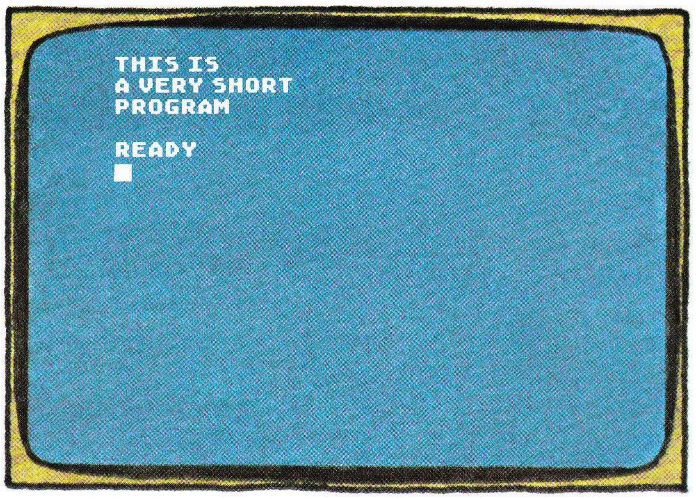


```
T:2+2
2+2
■
```

Figure 2-2

Figures 2-3 and 2-4 are examples of programs using the T: command. Notice that Figure 2-4 uses graphics characters that are created by holding down the CTRL key while typing the letters.

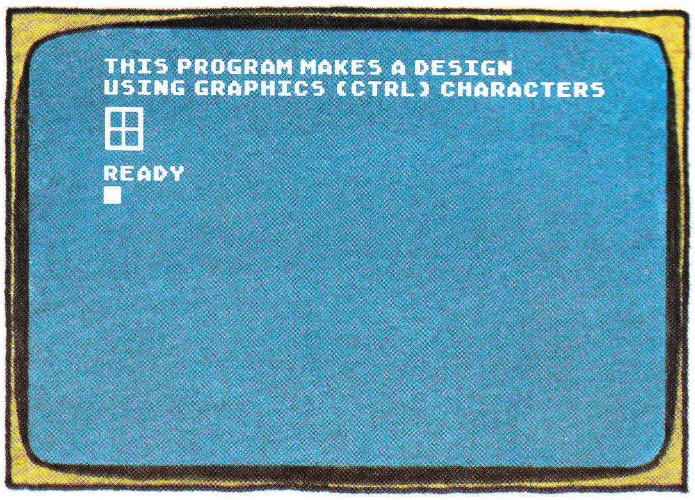
```
10 T:THIS IS
20 T:A VERY SHORT
30 T:PROGRAM
■
```



```
THIS IS
A VERY SHORT
PROGRAM

READY
■
```

Figure 2-3

```
10 T:THIS PROGRAM MAKES A DESIGN
20 T:USING GRAPHICS (CTRL)
CHARACTERS
30 T:
40 T: ⊞
50 T:
■
```



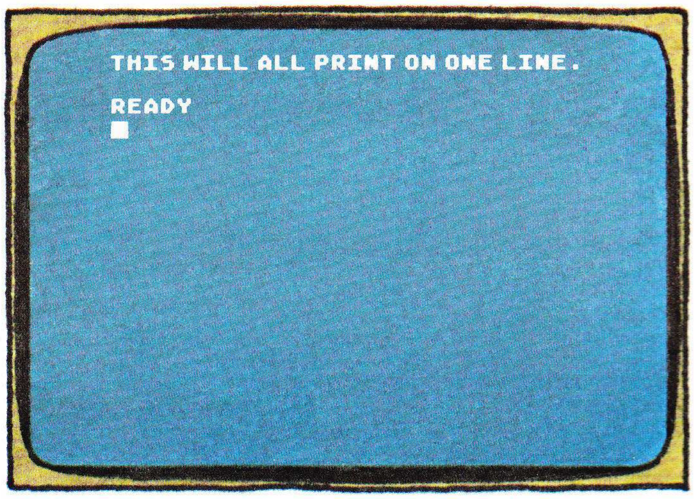
```
THIS PROGRAM MAKES A DESIGN
USING GRAPHICS (CTRL) CHARACTERS
⊞

READY
■
```

Figure 2-4

The backslash (\ ) has a special meaning to the T: command. Rather than printing the next output on a separate line, the backslash tells the computer to continue on the same line (see Figures 2-5 and 2-6).

Note: The computer will automatically move to the next line when it runs out of room.

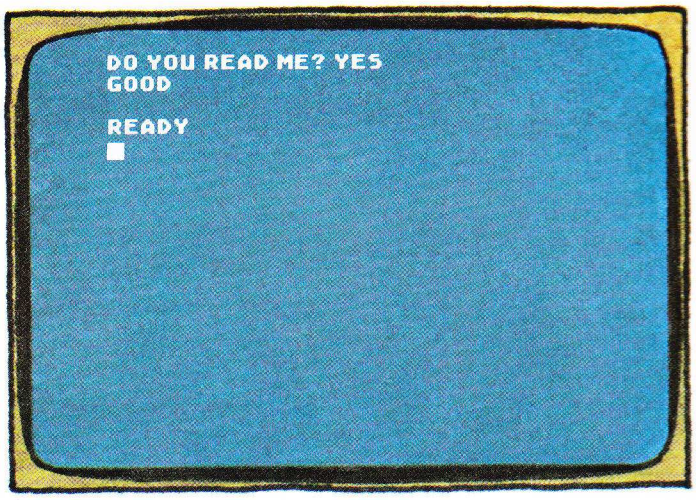
```
10 T:THIS WILL \
20 T:ALL PRINT \
30 T:ON ONE LINE.

RUN
■
```



```
THIS WILL ALL PRINT ON ONE LINE.

READY
■
```

Figure 2-5

```
10 T:DO YOU READ ME? \
20 A:
30 T:GOOD
■
```

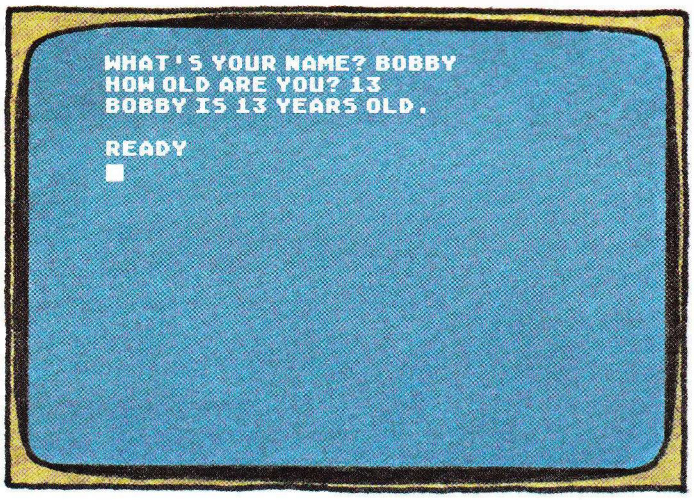


```
DO YOU READ ME? YES
GOOD

READY
■
```

Figure 2-6

When a T: command sees a string ($) or numeric (#) variable, it automatically replaces those variable names with their contents if they have been given a value (see Figure 2-7).

```
10  T:WHAT'S YOUR NAME? \
20  A:$NAME
30  T:HOW OLD ARE YOU? \
40  A:#A
50  T:$NAME IS #A YEARS OLD.

READY
■
```



```
WHAT'S YOUR NAME? BOBBY
HOW OLD ARE YOU? 13
BOBBY IS 13 YEARS OLD.

READY
■
```

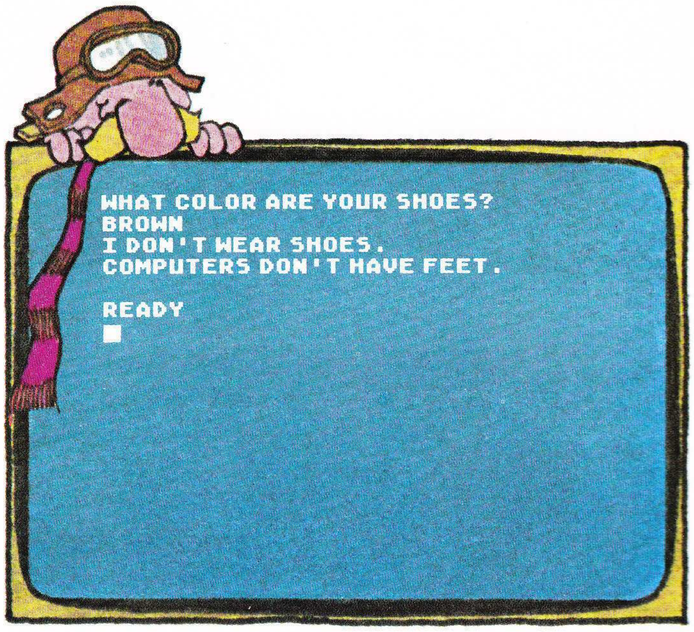
Figure 2-7

A:

Accept

The A: (ACCEPT) command tells the program to wait for information to be typed from the keyboard. Figure 2-8 is an example of a simple use of the A: command.

```
10 T:WHAT COLOR ARE YOUR SHOES?
20 A:
30 T:I DON'T WEAR SHOES.
40 T:COMPUTERS DON'T HAVE FEET.

READY
■
```

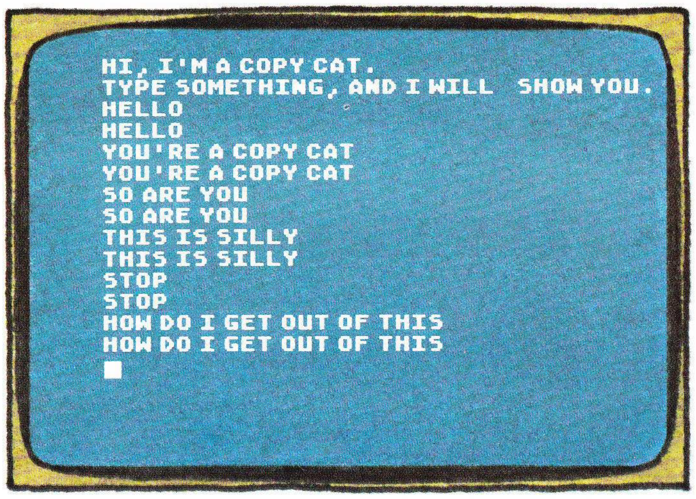


```
WHAT COLOR ARE YOUR SHOES?
BROWN
I DON'T WEAR SHOES.
COMPUTERS DON'T HAVE FEET.

READY
■
```

Figure 2-8

The A: command also lets you give a value to a string ($) or numeric (#) variable as shown in Figures 2-9 and 2-10.

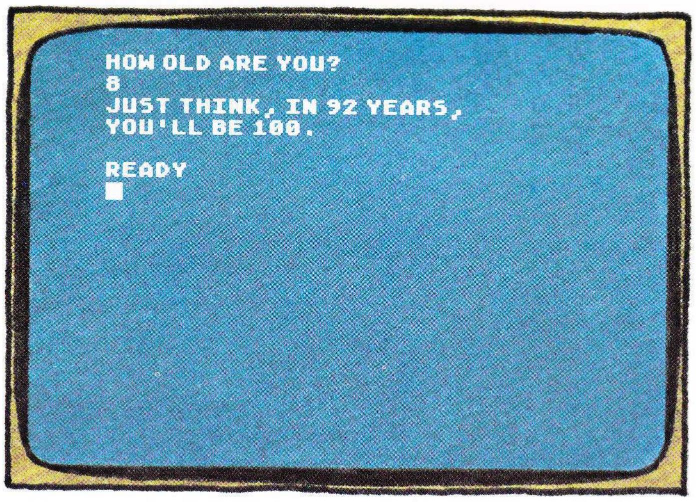
```
10 R:COPY CAT PROGRAM
20 T:HI, I'M A COPY CAT
30 T:TYPE SOMETHING, AND I WILL
SHOW YOU.
40 *AGAIN
50 A:$COPY
60 T:$COPY
70 J:*AGAIN

READY
■
```



```
HI, I'M A COPY CAT.
TYPE SOMETHING, AND I WILL  SHOW YOU.
HELLO
HELLO
YOU'RE A COPY CAT
YOU'RE A COPY CAT
SO ARE YOU
SO ARE YOU
THIS IS SILLY
THIS IS SILLY
STOP
STOP
HOW DO I GET OUT OF THIS
HOW DO I GET OUT OF THIS
■
```

Figure 2-9

```
10  R:AGE OLD PROGRAM
20  T:HOW OLD ARE YOU?
30  A:#Y
40  C:#L=100-#Y
50  T:JUST THINK, IN #L YEARS,
60  T:YOU'LL BE 100.

READY
■
```

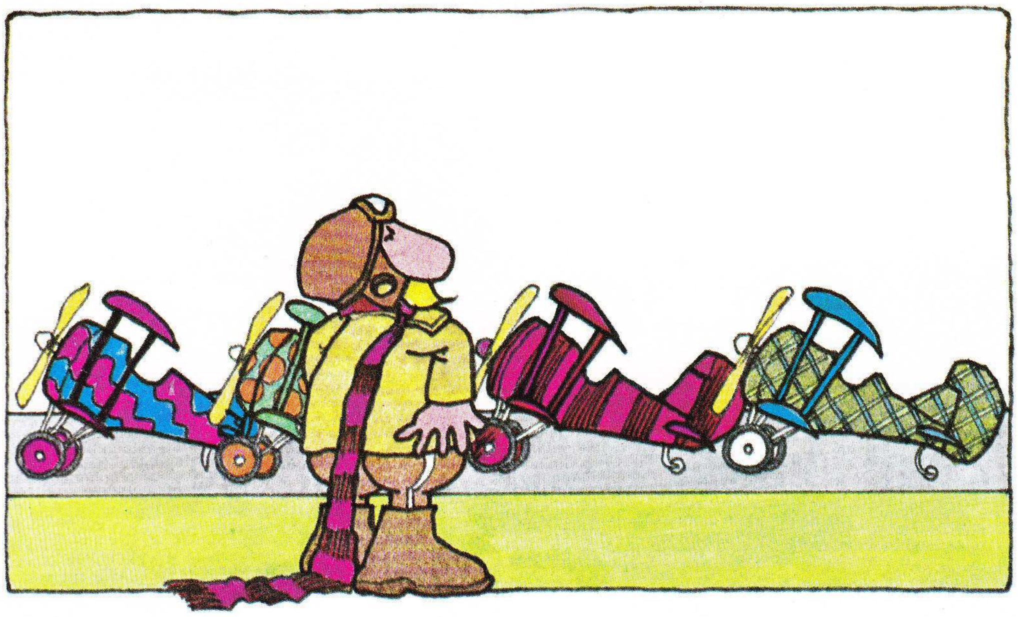


```
HOW OLD ARE YOU?
8
JUST THINK, IN 92 YEARS,
YOU'LL BE 100.

READY
■
```

Figure 2-10

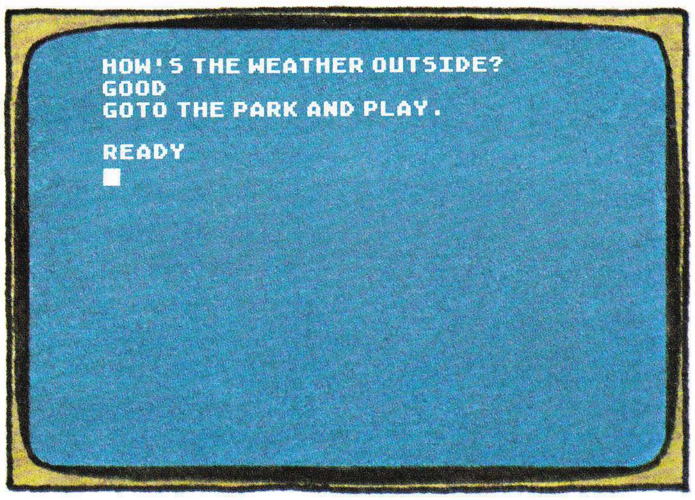
M:

Match

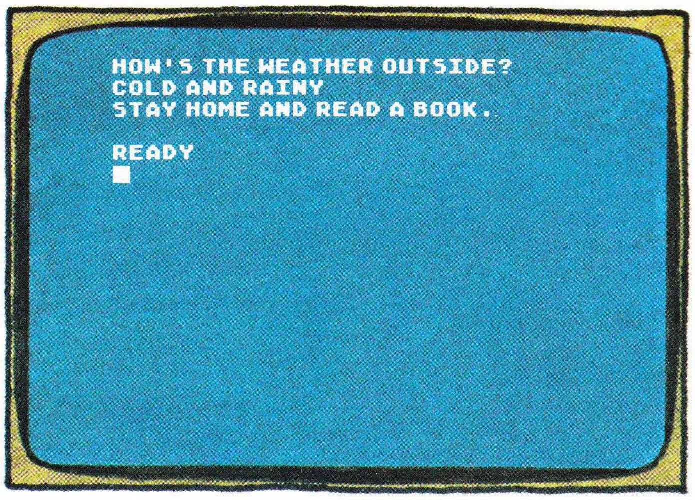


The M: command matches the answer entered by a user (during an A: command) with one or more answers expected by the program. Also, M: automatically creates a YES or NO condition — Yes (Y) there was a match, or NO (N) there was no match. Figure 2-11 shows how this works.

```
10 R:WEATHER TO DO
20 T:HOW'S THE WEATHER OUTSIDE?
30 A:
40 M:NICE,GOOD,HOT,GREAT
50 TY:GOTO THE PARK AND PLAY.
60 TN:STAY HOME AND READ A BOOK.
70 E:

READY
■
```



```
HOW'S THE WEATHER OUTSIDE?
GOOD
GOTO THE PARK AND PLAY.

READY
■
```



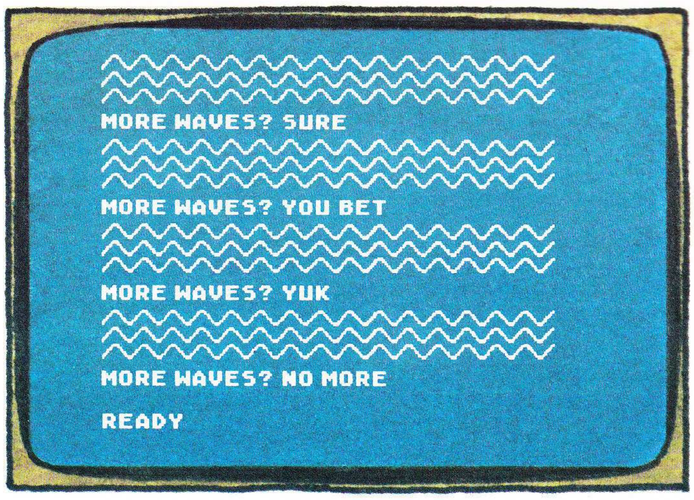
```
HOW'S THE WEATHER OUTSIDE?
COLD AND RAINY
STAY HOME AND READ A BOOK.

READY
■
```

Figure 2-11

In Figure 2-11, if the answer matches *nice*, *good*, *hot*, or *great*, the program types "Go to the park and play" (line 50). If the answer does not match, the program types "Stay home and read a book" (line 60).

In the list of match choices, if you put a space followed by a single letter, the computer will match anything starting with that letter. For example, in Figure 2-12 below, the M:(space)Y will match any word that starts with the letter Y; this would include YES, YUP, YEAH, YOU BET, or even YUK!

```
10  R:WAVES
20  *WATER
30  T: /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
40  T: /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
50  T: /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
60  T:MORE WAVES? \
70  A:
80  M:  Y,OK,O.K.,ALRIGHT,SURE
90  JY: *WATER
100  E:

READY
■
```



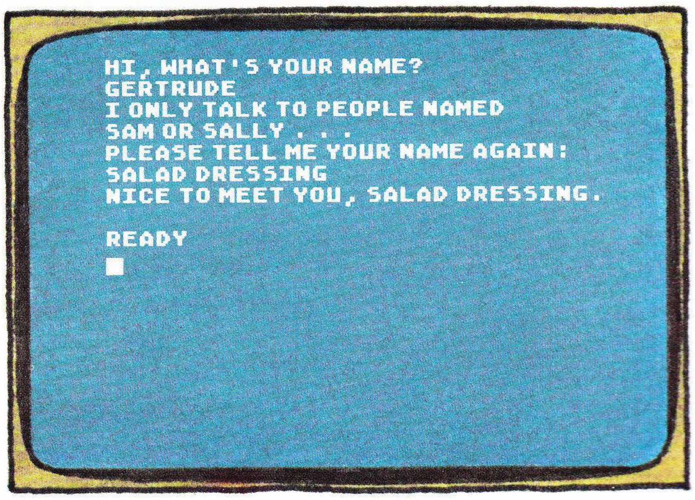
```
/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
MORE WAVES? SURE
/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
MORE WAVES? YOU BET
/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
MORE WAVES? YUK
/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
MORE WAVES? NO MORE

READY
```

Figure 2-12

Note: The JY: command in line 80 of Figure 2-12 is a conditional JUMP command. See J: (JUMP) or Section 6, Conditionals, for an explanation.

The computer will look to see if any of the choices in a match list are contained, exactly, in any part of an answer. The program in Figure 2-13 below will match on SAL or SAM as any part of the answer. This means that all of the following will create a match: SALLY, SALT, SALAD, SALUTE, SALAMI, SAMMY, SAMUEL, and SAME.

```
10  R:SAM'S & SALLY'S PROGRAM
20  T:HI, WHAT'S YOUR NAME?
30  *AGAIN A: $NAME
40  M:SAM,SAL
50  TN:I ONLY TALK TO PEOPLE NAMED
60  TN:SAM OR SALLY . . .
70  TN:PLEASE TELL ME YOUR NAME
AGAIN:
80  JN:*AGAIN
90  TY:NICE TO MEET YOU, $NAME.
100  E:

READY
■
```

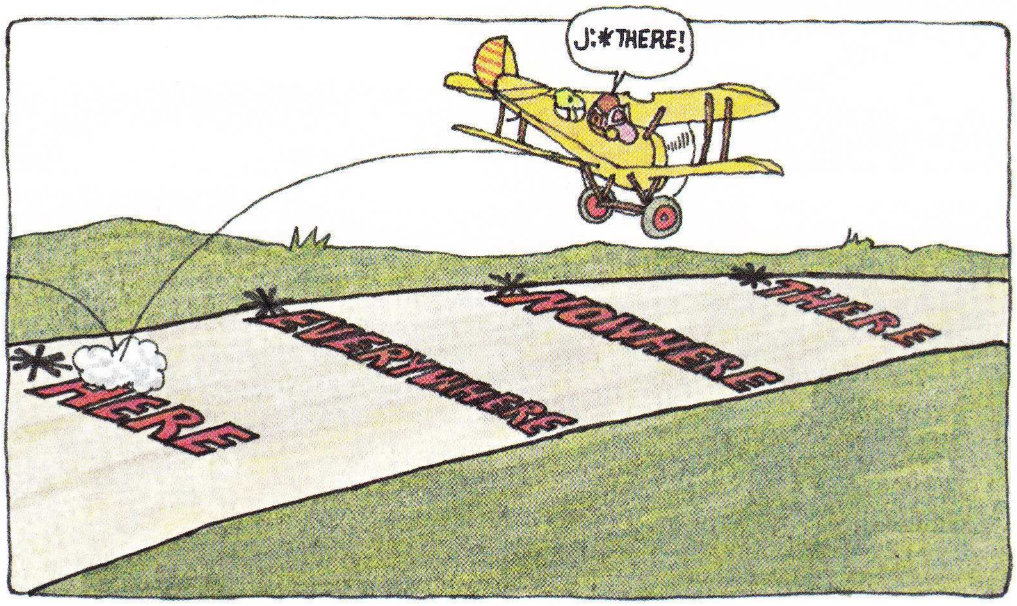


```
HI, WHAT'S YOUR NAME?
GERTRUDE
I ONLY TALK TO PEOPLE NAMED
SAM OR SALLY . . .
PLEASE TELL ME YOUR NAME AGAIN:
SALAD DRESSING
NICE TO MEET YOU, SALAD DRESSING.

READY
■
```

Figure 2-13

J:

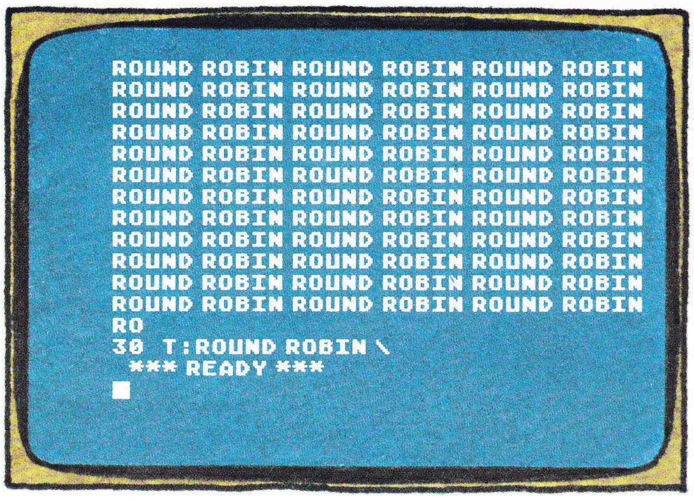
Jump



The J: (JUMP) command lets you jump from one spot in a program to another. The J: is ALWAYS followed by a ✻ LABEL indicating to which statement you want to jump. The destination of your J: command must also be labeled with the same ✻ LABEL.

Figure 2-14 shows how the J: command can be used to make a simple loop. The program keeps looping (repeating), filling the screen with *ROUND ROBIN.*

```
10  R:LOOPING PROGRAM
20  *OVER
30  T:ROUND ROBIN \
40  J: *OVER

READY
■
```



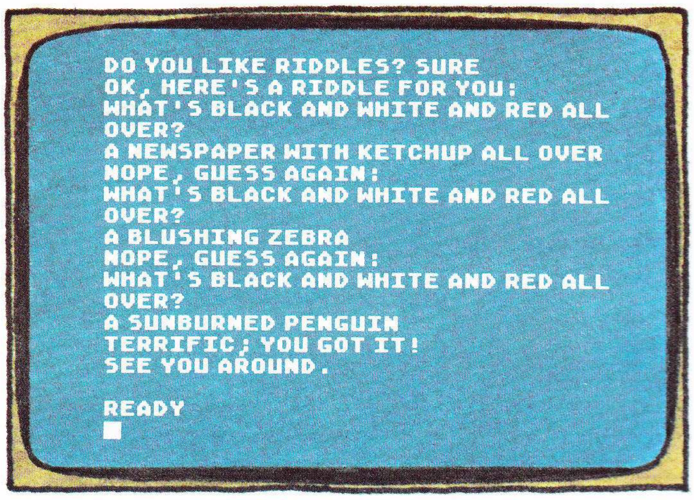
```
ROUND ROBIN ROUND ROBIN ROUND ROBIN
ROUND ROBIN ROUND ROBIN ROUND ROBIN
ROUND ROBIN ROUND ROBIN ROUND ROBIN
ROUND ROBIN ROUND ROBIN ROUND ROBIN
ROUND ROBIN ROUND ROBIN ROUND ROBIN
ROUND ROBIN ROUND ROBIN ROUND ROBIN
ROUND ROBIN ROUND ROBIN ROUND ROBIN
ROUND ROBIN ROUND ROBIN ROUND ROBIN
ROUND ROBIN ROUND ROBIN ROUND ROBIN
ROUND ROBIN ROUND ROBIN ROUND ROBIN
ROUND ROBIN ROUND ROBIN ROUND ROBIN
ROUND ROBIN ROUND ROBIN ROUND ROBIN
RO
30  T:ROUND ROBIN \
 *** READY ***
■
```

Figure 2-14

Note: Figure 2-14 is an example of a never-ending loop. To stop it, press the BREAK key.

✱ LABELS can stand alone as in line 20 of Figure 2-14, or they can be placed in front of another command, as in lines 70 and 130 in Figure 2-15. This screen also shows you how the J: command is used conditionally following an M: command. That is, JY: (JUMP only if there is a match), or JN: (JUMP if there is no match).

```
10 R:RIDDLE PROGRAM
20 T:DO YOU LIKE RIDDLES? \
30 A:
40 M:YES,YUP,YEAH,SURE,OK,
O.K.,ALRIGHT
50 JN:*BYE
60 T:OK, HERE'S A RIDDLE FOR YOU:
70 *AGAIN T:WHAT'S BLACK AND
WHITE AND RED ALL OVER?
80 A:
90 M:SUNBURNED PENGUIN
100 TN:NOPE, GUESS AGAIN
110 JN:*AGAIN
120 T:TERRIFIC; YOU GOT IT!
130 *BYE T:SEE YOU AROUND.
140 E:

READY
■
```

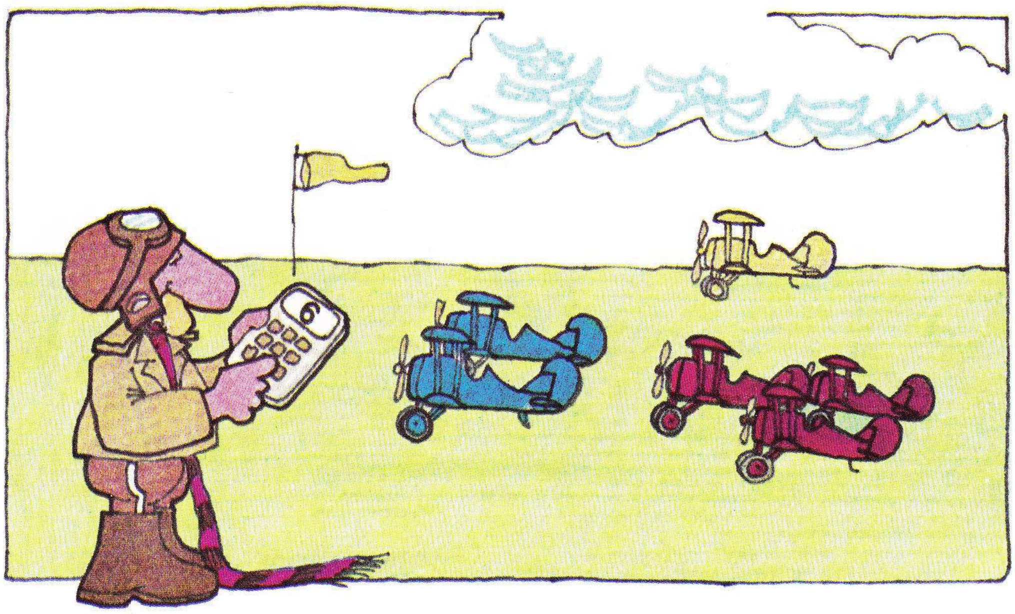


```
DO YOU LIKE RIDDLES? SURE
OK, HERE'S A RIDDLE FOR YOU:
WHAT'S BLACK AND WHITE AND RED ALL
OVER?
A NEWSPAPER WITH KETCHUP ALL OVER
NOPE, GUESS AGAIN:
WHAT'S BLACK AND WHITE AND RED ALL
OVER?
A BLUSHING ZEBRA
NOPE, GUESS AGAIN:
WHAT'S BLACK AND WHITE AND RED ALL
OVER?
A SUNBURNED PENGUIN
TERRIFIC; YOU GOT IT!
SEE YOU AROUND.

READY
■
```

Figure 2-15

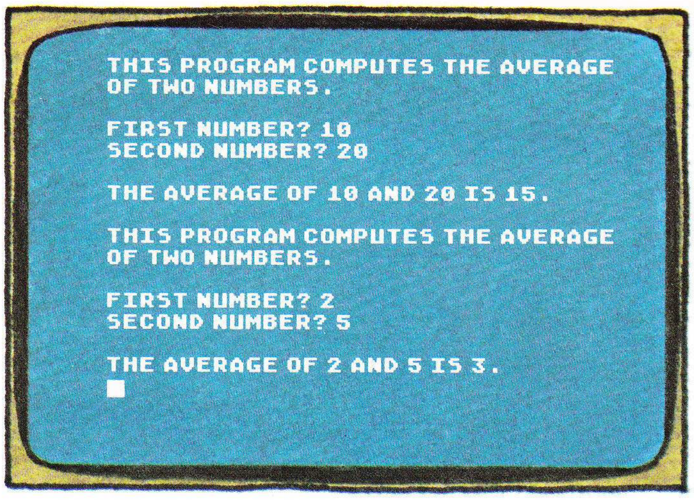
C:

Compute



Any time you want to do arithmetic in PILOT, you must use the C: (COMPUTE) command with numeric variables. As an example, the program in Figure 2-16 lets you input two numbers, and then it computes their average (#A).

```
10 R:AVERAGE PROGRAM
20 *DOAGAIN
30 T:THIS PROGRAM COMPUTES THE
AVERAGE
40 T:OF TWO NUMBERS
50 T:
60 T:FIRST NUMBER? \
70 A:#F
80 T:SECOND NUMBER? \
90 A:#S
100 C:#A=#F+#S/2
110 T:
120 T:THE AVERAGE OF #F AND #S IS
#A.
130 T:
140 J:*DOAGAIN

READY
■
```



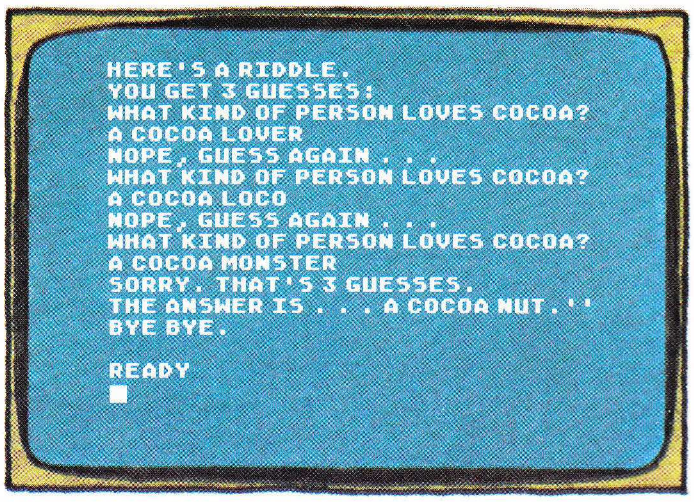
```
THIS PROGRAM COMPUTES THE AVERAGE
OF TWO NUMBERS.

FIRST NUMBER? 10
SECOND NUMBER? 20

THE AVERAGE OF 10 AND 20 IS 15.

THIS PROGRAM COMPUTES THE AVERAGE
OF TWO NUMBERS.

FIRST NUMBER? 2
SECOND NUMBER? 5

THE AVERAGE OF 2 AND 5 IS 3.
■
```

Figure 2-16

Note: PILOT will evaluate arithmetic expresions from left to right unless you have used parentheses to show a different grouping.

The C: command is often used to count things in a program. For example, you might want to give a person three chances at answering a riddle. A numeric variable such as #G is used to keep track of the number of guesses made so far. A numeric condition is used to find out whether or not #G has reached the limit (three in this case). Figure 2-17 below shows you how this looks in a program.

```
10  R:THREE GUESSES
20  C:#G=1
30  T:HERE'S A RIDDLE.
40  T:YOU GET 3 GUESSES:
50  *AGAIN T:WHAT KIND OF PERSON
LOVES COCOA?
60  A:
70  M:  COCONUT , COCOANUT , COCOA
NUT ,
80  TY:VERY GOOD; YOU'RE SMART!
90  JY:*BYE
100  J(#G=3): *NOMORE
110  T:NOPE, GUESS AGAIN
120  C:#G=#G+1
130  J:*AGAIN
140  *NOMORE T:SORRY, THAT'S 3
GUESSES.
150  T:THE ANSWER IS . . . .
160  PA:90
170  T:''A COCOA NUT.''
180  *BYE T:BYE BYE
190  E:

READY
■
```



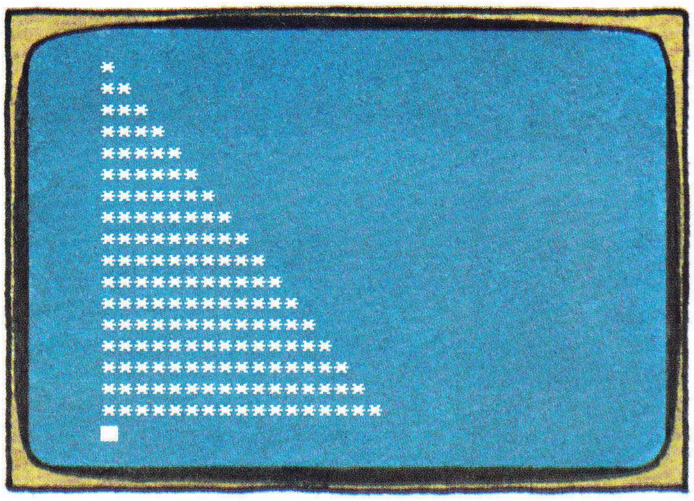
```
HERE'S A RIDDLE.
YOU GET 3 GUESSES:
WHAT KIND OF PERSON LOVES COCOA?
A COCOA LOVER
NOPE, GUESS AGAIN . . .
WHAT KIND OF PERSON LOVES COCOA?
A COCOA LOCO
NOPE, GUESS AGAIN . . .
WHAT KIND OF PERSON LOVES COCOA?
A COCOA MONSTER
SORRY. THAT'S 3 GUESSES.
THE ANSWER IS . . . A COCOA NUT.''
BYE BYE.

READY
■
```

Figure 2-17

Note: The numeric conditional in the J: command in line 100 is explained in Section 6.

You may also use the C: (COMPUTE) command to give a value to a string ($) variable within a program. For example, the program in Figure 2-18 keeps adding one more star (✻) to the value of the string variable $GROW until the value reaches 21 stars.

```
10  R:GROWING STARS
20  C:$GROW=
30  C:$STAR=*
40  C:#C=0
50  *LOOP
60  C:#C=#C+1
70  C:$GROW=$GROW$STAR
80  T:$GROW
90  J(#C<21):*LOOP
100  E:

READY
■
```



```
*
**
***
****
*****
******
*******
********
*********
**********
***********
************
*************
**************
***************
****************
*****************
■
```

Figure 2-18

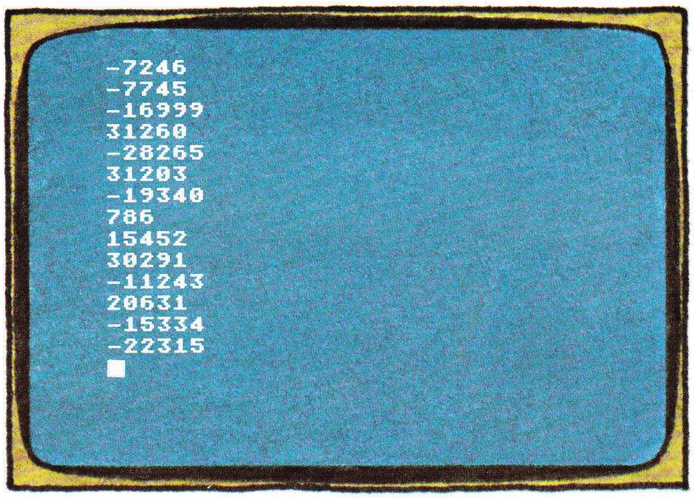
RANDOM NUMBERS

Besides the examples above, the C: (COMPUTE) command will let the computer select numbers at random. For example, if all the numbers allowed by PILOT were mixed up in a large fish bowl, the computer would reach into the bowl and choose one by chance. The random number is expressed as a **?** for the value of a numeric variable in a C: command. For example,

C:#R=?.

The program in Figure 2-19 generates random numbers and types them out.

```
10 R:RANDOM NUMBERS
20 *AGAIN
30 C:#R=?
40 T:#R
50 J:*AGAIN

READY
■
```

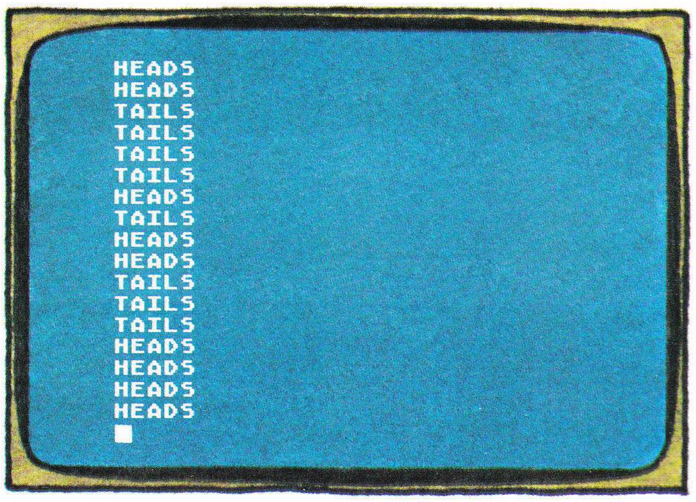


```
-7246
-7745
-16999
31260
-28265
31203
-19340
786
15452
30291
-11243
20631
-15334
-22315
■
```

Figure 2-19

You can tell the computer only to select random numbers from 0 through an upper limit. Simply place a backslash (\ ) and the upper limit *after* the question mark (?). For example, **C:#R = ? \ 10** will choose a random number from 0 through 9. The COIN TOSS program in Figure 2-20 randomly generates only 0's (HEADS) and 1's (TAILS).

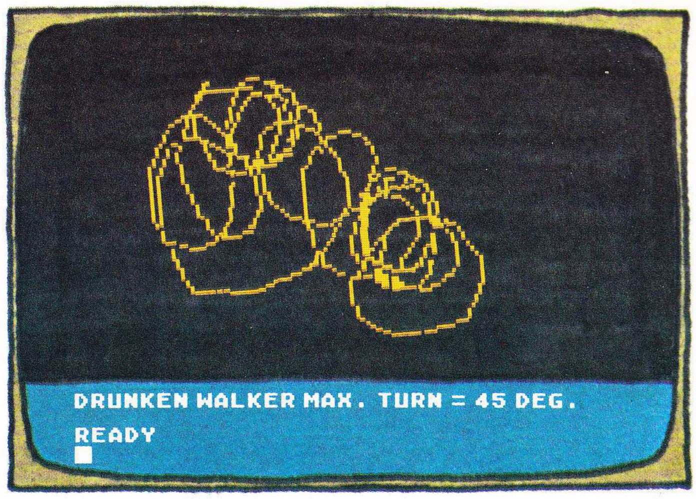
```
10 R:COIN FLIP
20 *MORE
30 C:#R=?\2
40 T(#R=0):HEADS
50 T(#R=1):TAILS
60 J:*MORE

READY
■
```



```
HEADS
HEADS
TAILS
TAILS
TAILS
TAILS
HEADS
TAILS
HEADS
HEADS
TAILS
TAILS
TAILS
HEADS
HEADS
HEADS
HEADS
■
```

A numeric variable can be used as the upper limit for selecting a random number. For example, the program in Figure 2-21 allows you to type in the maximum amount that a drunkard will turn at each step as he staggers for 200 paces.

```
10 *DRUNKWALKER
20 GR:CLEAR
30 T:ENTER MAXIMUM TURN (0-360):\
40 A:#R
50 T:} [CLEAR SCREEN]
60 T:DRUNKEN WALKER MAX. TURN =
#R D EG.
70 C:#C=0
80 *WALK
90 GR:DRAW 5
100 PA:5
110 GR:TURN ?\#R
120 C:#C=#C+1
130 J(#C<200):*WALK
140 E:

READY
■
```

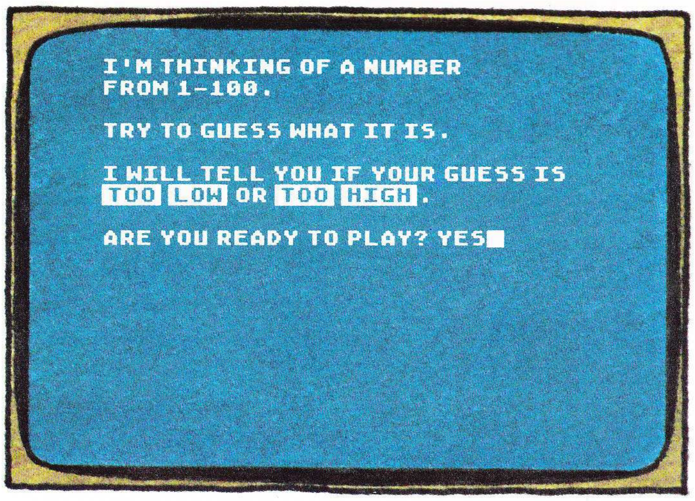


```
DRUNKEN WALKER MAX. TURN = 45 DEG.
READY
■
```

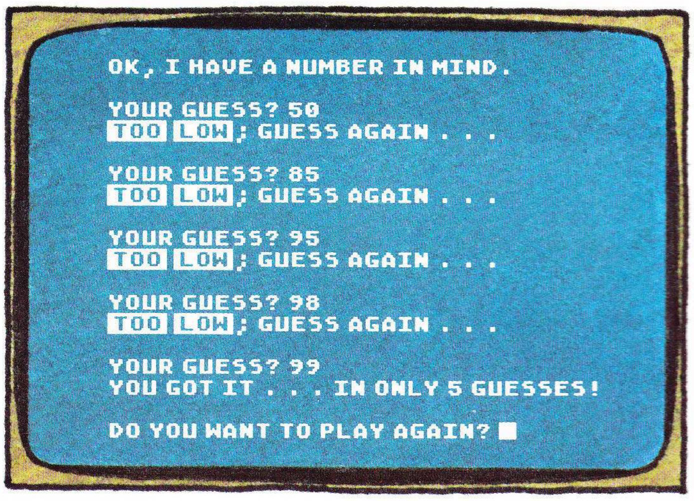
Figure 2-21

The GUESS MY NUMBER program in Figure 2-22 gives you a more complete example of how you can use the random number feature of PILOT.

```
10  R:GUESS MY NUMBER
20  T:I'M THINKING OF A NUMBER
30  T:FROM 1-100
40  T:
50  T:TRY TO GUESS WHAT IT IS.
60  T:
70  T:I WILL TELL YOU IF YOUR GUESS
80  T:TOO LOW OR TOO HIGH.
90  T:
100  T:ARE YOU READY TO PLAY?
110  A:
120  M:Y,SURE,OK,ALRIGHT
130  JN:*BYE
140  *BEGIN
150  T:↰ [CLEAR THE SCREEN]
160  C:#R=?\100+1
170  C:#C=0
180  T:OK, I HAVE A NUMBER IN MIND.
190  T:
200  *AGAIN
210  T:
220  T:YOUR GUESS? \
230  A:#G
240  C:#C=#C+1
250  T(#G<#R):TOO LOW; GUESS
AGAIN . . .
260  T(#G>#R):TOO HIGH, GUESS
AGAIN . .
270  J(#G<>#R):*AGAIN
280  T:YOU GOT IT . . . IN ONLY #C
GUESSES!
290  U:*BEEP
300  T:
310  T:DO  YOU WANT TO PLAY AGAIN? \
320  A:
330  M:Y,SURE,OK,ALRIGHT
340  JY:*BEGIN
350  *BYE T:OK, SEE YOU AROUND.
360  E:
370  *BEEP
380  C:#N=0
390  C:#S=10-#C
400  *BEEPAGAIN
410  SO:13
420  PA:8
430  SO:0
440  C:#N=#N+1
450  J(#N<#S):*BEEPAGAIN
460  SO:13,17,20,25
470  PA:60
480  SO:0
490  E:

READY
■
```



```
I'M THINKING OF A NUMBER
FROM 1-100.

TRY TO GUESS WHAT IT IS.

I WILL TELL YOU IF YOUR GUESS IS
TOO LOW OR TOO HIGH.

ARE YOU READY TO PLAY? YES■
```

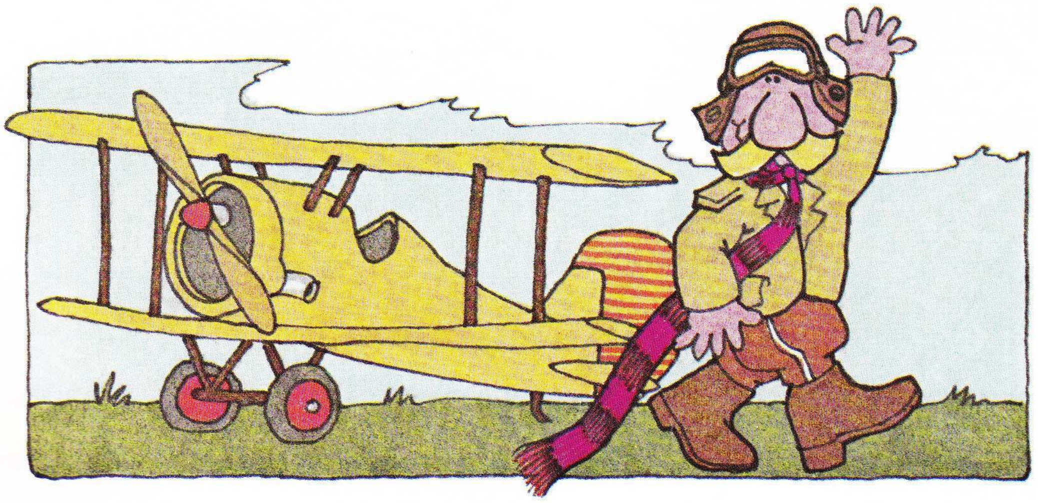


```
OK, I HAVE A NUMBER IN MIND.

YOUR GUESS? 50
TOO LOW; GUESS AGAIN . . .

YOUR GUESS? 85
TOO LOW; GUESS AGAIN . . .

YOUR GUESS? 95
TOO LOW; GUESS AGAIN . . .

YOUR GUESS? 98
TOO LOW; GUESS AGAIN . . .

YOUR GUESS? 99
YOU GOT IT . . . IN ONLY 5 GUESSES!

DO YOU WANT TO PLAY AGAIN? ■
```

Figure 2-22

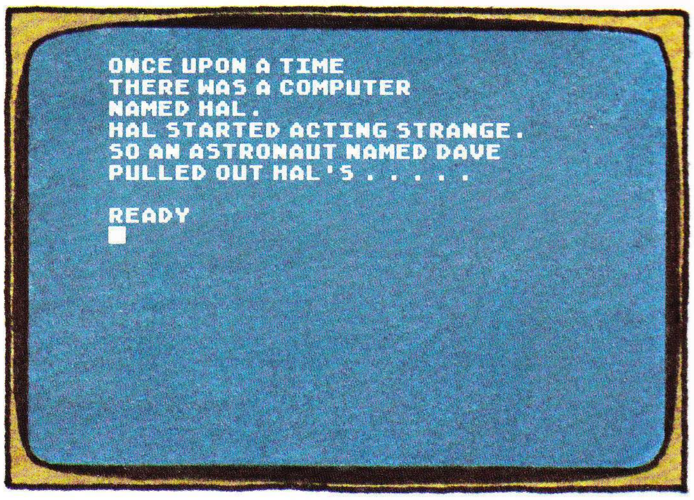
E:

End



The E: command marks the END of something. In Figure 2-23 below, it marks the END of your program.

```
10  R:SUDDEN ENDING
20  T:ONCE UPON A TIME
30  T:THERE WAS A COMPUTER
40  T:NAMED HAL.
50  T:HAL STARTING ACTING STRANGE.
60  T:SO AN ASTRONAUT NAMED DAVE
70  T:PULLED OUT HAL'S . . .
80  E:

READY
■
```

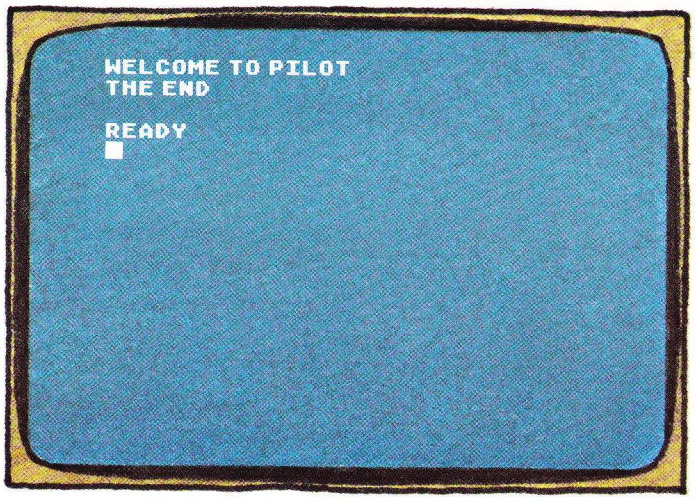


```
ONCE UPON A TIME
THERE WAS A COMPUTER
NAMED HAL.
HAL STARTED ACTING STRANGE.
SO AN ASTRONAUT NAMED DAVE
PULLED OUT HAL'S . . . . .

READY
■
```

Figure 2-23

The end of a program is not necessarily the last line of the program. In Figure 2-24, the E: command is used to mark the END of the program at line 70, and it is also used to mark the END of the ✻ TRUMPETS module at line 120.

```
10 R:FANFARE PROGRAM
20 T:WELCOME TO PILOT
30 U:*TRUMPETS
40 PA:10
50 U:*TRUMPETS
60 T:THE END
70 E:
80 *TRUMPETS
90 SO:1,5,9,13
100 PA:120
110 SO:0,0,0,0
120 E:

READY

■
```

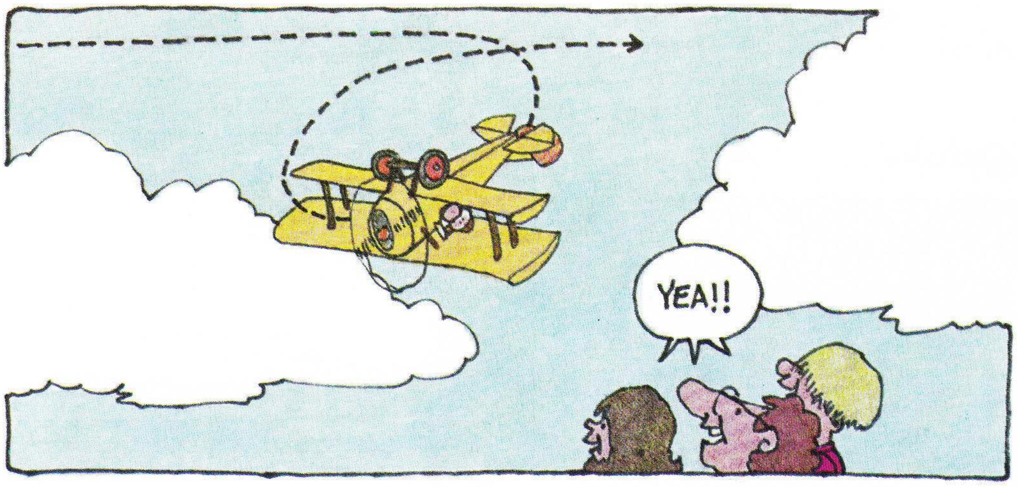


```
WELCOME TO PILOT
THE END

READY
■
```

Figure 2-24

U:

Use



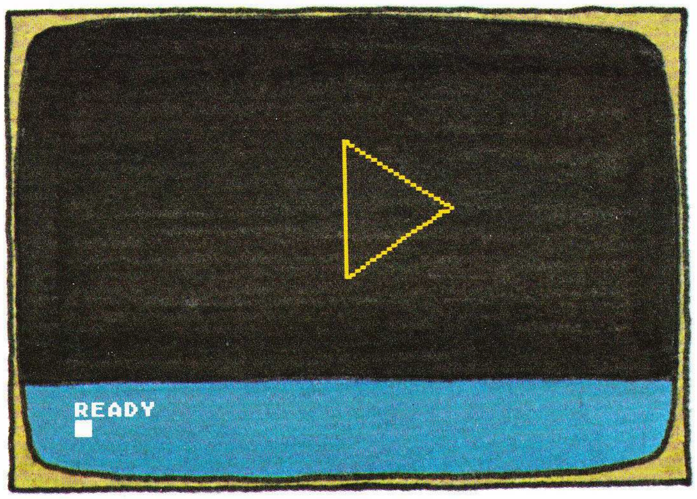
The U: command tells the computer to USE a module. A module is a program (sub-program) within a program.

Every module must begin with a ✱ LABEL and must end with an E:. Figure 2-25 shows a module that draws a triangle.

```
100 *TRIANGLE
110 GR:3(DRAW 30;TURN 120)
120 E:

READY
■
```

Figure 2-25

If you type U: ✱ LABEL in *Immediate* mode, you can check out a module without running the entire program. For example, Figure 2-26 is the result of typing U: ✱ TRIANGLE.



```
READY
■
```

Figure 2-26

Figure 2-27 shows the ✱ TRIANGLE module within a larger program. When the computer sees U: ✱ TRIANGLE, it jumps to the statement labeled ✱ TRIANGLE and uses that module. At the end of the module, the computer continues with the command following U: ✱ TRIANGLE.

```
10 R:WHEEL OF TRIANGLES
20 C:#T=0
30 GR:CLEAR
40 *AGAIN C:#T=#T+1
50 U:*TRIANGLE
60 GR:TURN 36
70 J(#T<10) :*AGAIN
80 E:
100 *TRIANGLE
110 GR:3(DRAW 30; TURN 120)
120 E:

READY
■
```

Figure 2-27

The program in Figure 2-27 uses the ✻ TRIANGLE module ten times to draw a wheel of triangles. The final result of this program is shown in Figure 2-28 below.

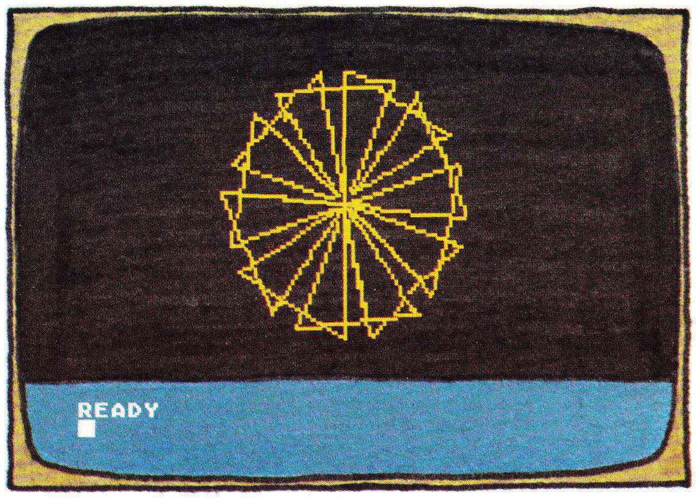


Figure 2-28

Figure 2-29 shows a program that uses several modules (✻ BOX, ✻ TRIANGLE, ✻ MOON, and ✻ STAR) to make a picture.

```
10 R:PICTURE
20 GR:CLEAR
30 GR:GOTO -25,-10
40 GR:PEN BLUE
50 U:*BOX
60 GR:TURNTO 30;GO 25
70 GR:PEN RED
80 U:*TRIANGLE
90 GR:GOTO 0,25
100 GR:PEN YELLOW
110 U:*MOON
120 GR:GOTO -40,25
130 U:*STAR
140 GR:GOTO -10,35
150 U:*STAR
160 GR:GOTO 35,20
170 U:*STAR
299 E:
400 *BOX
410 GR:TURNTO 90
420 GR:3(DRAW 25; TURN 90)
430 GR:FILL 25
440 E:
500 *TRIANGLE
510 GR:TURNTO 150
520 GR:2(DRAW 25; TURN 120);FILL 24
530 E:
600 *MOON
610 GR:TURNTO 70
620 GR:38(DRAW 1; TURN 5)
630 GR:TURNTO 50
640 GR:5(TURN -20; FILL 5)
650 E:
700 *STAR
710 GR:5(DRAW 10; TURN 144)
720 E:

READY
■
```

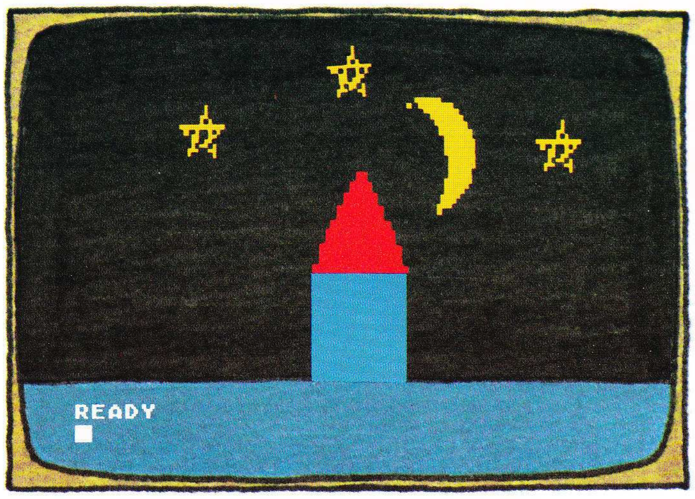


Figure 2-29

Note: It is possible to have modules used within modules up to eight layers deep. For example, the ✻ TRIANGLE module could itself use a ✻ FILL module; the ✻ FILL module could use a ✻ PENCOLOR module, and so on.

R:

Remark

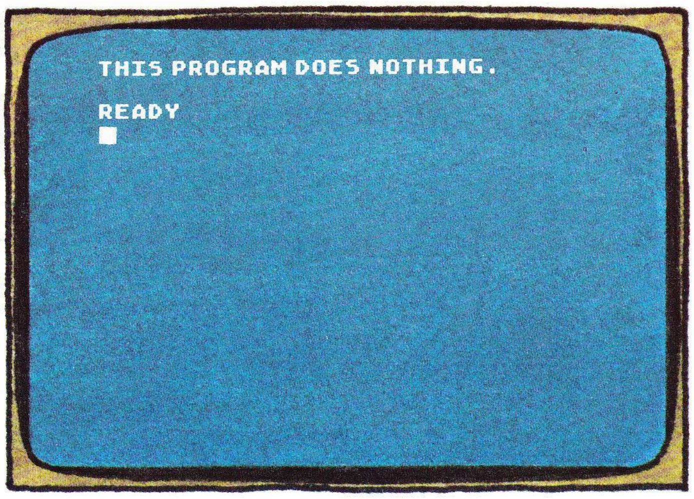
The R: (REMARK) command lets you say something about the program. However, any REMARKS made will be ignored by the computer when you RUN the program. The only time you see them is when you LIST the program onto the television screen. For example, Figure 2-30 is a program made up almost entirely of REMARKS.

```
10  R:***************************
20  R:*                         *
30  R:*                         *
40  R:*    THE NOTHING PROGRAM  *
50  R:*                         *
60  R:*                         *
70  R:***************************
80  T:[THIS IS ALSO A REMARK]
90  T:THIS PROGRAM DOES NOTHING.
100  R:THAT'S FOR SURE!
110  R:THIS IS THE END OF THE
PROGRAM.
120  E:

READY
■
```

Figure 2-30

Note: Square brackets [ ] are used to indicate a remark on the same line as a command as shown in line 80 above.

When you type **RUN** and press RETURN, Figure 2-31 is all you will see on the screen.

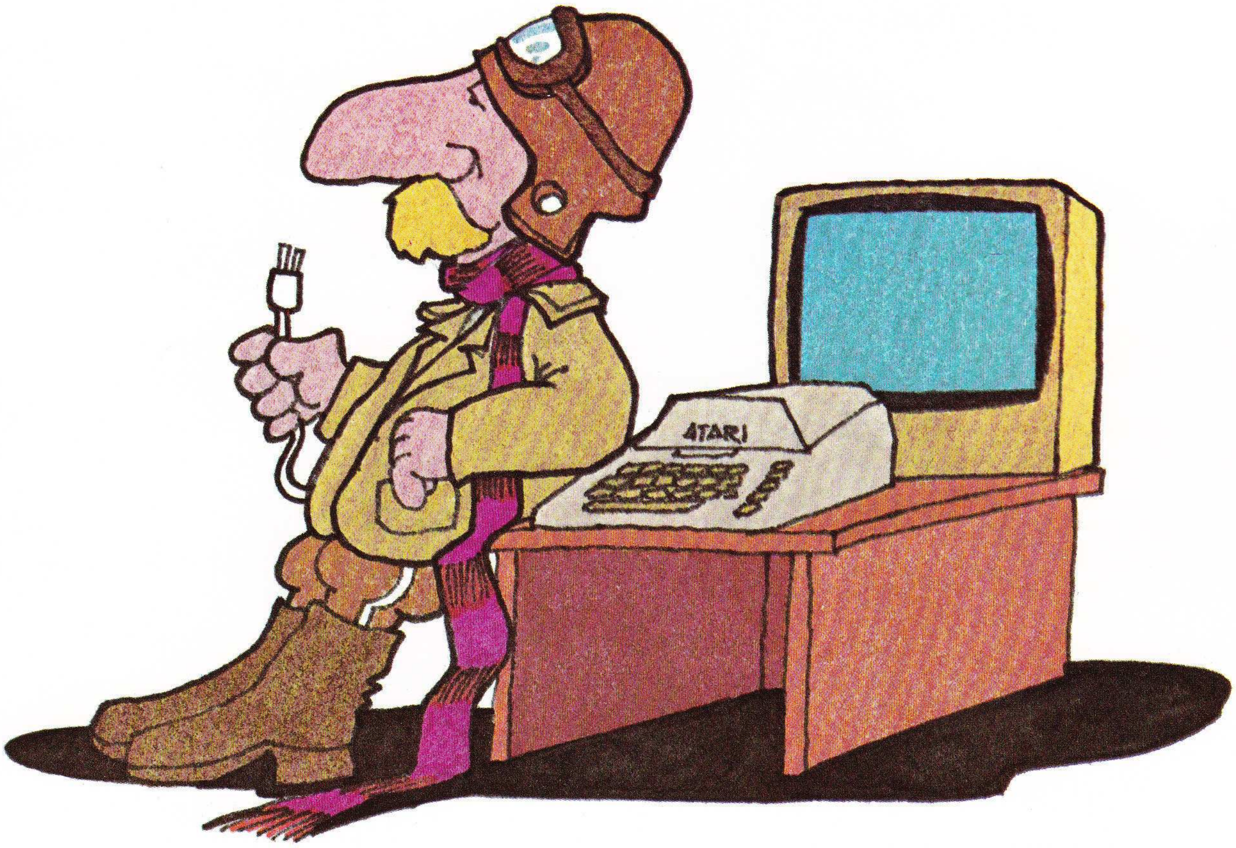


```
THIS PROGRAM DOES NOTHING.

READY
■
```

Figure 2-31

REMARKS are particularly helpful when you write longer programs because they will remind you how the program works long after you have written it.

3 EXECUTIVE COMMANDS



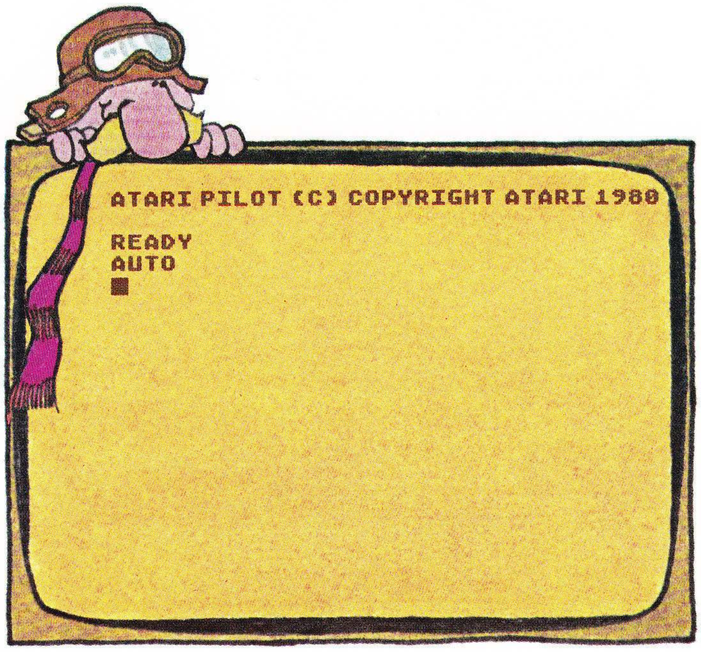
AUTO

Automatic Line Numbering

Use the AUTO command when you want to type a program into the computer memory. AUTO *automatically* numbers your program lines for you.

While you are in the AUTO mode, the computer will make everything you type part of your program. No Immediate mode commands can be entered until you exit from the AUTO mode by pressing the RETURN key.

When you type **AUTO** your screen turns yellow as in Figure 3-1. When you leave the AUTO mode, your screen turns blue.



```
ATARI PILOT (C) COPYRIGHT ATARI 1980

READY
AUTO
■
```

Figure 3-1

Remember that the computer enters the line numbers, which you cannot see until you exit from the AUTO mode and LIST the program.

Figure 3-2 shows a program entered in the AUTO mode.

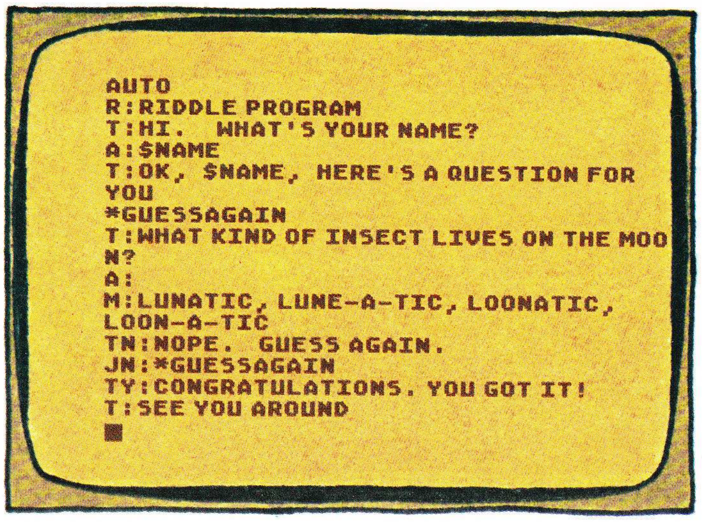

```
AUTO
R:RIDDLE PROGRAM
T:HI.  WHAT'S YOUR NAME?
A:$NAME
T:OK, $NAME, HERE'S A QUESTION FOR
YOU
*GUESSAGAIN
T:WHAT KIND OF INSECT LIVES ON THE MOO
N?
A:
M:LUNATIC, LUNE-A-TIC, LOONATIC,
LOON-A-TIC
TN:NOPE.  GUESS AGAIN.
JN:*GUESSAGAIN
TY:CONGRATULATIONS. YOU GOT IT!
T:SEE YOU AROUND
■
```

Figure 3-2

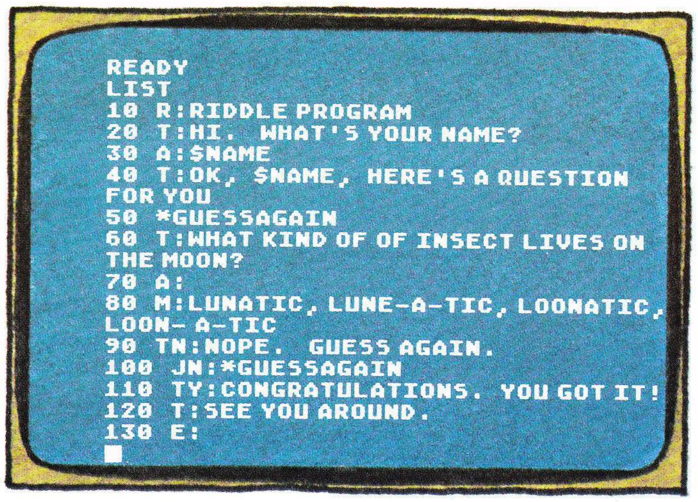
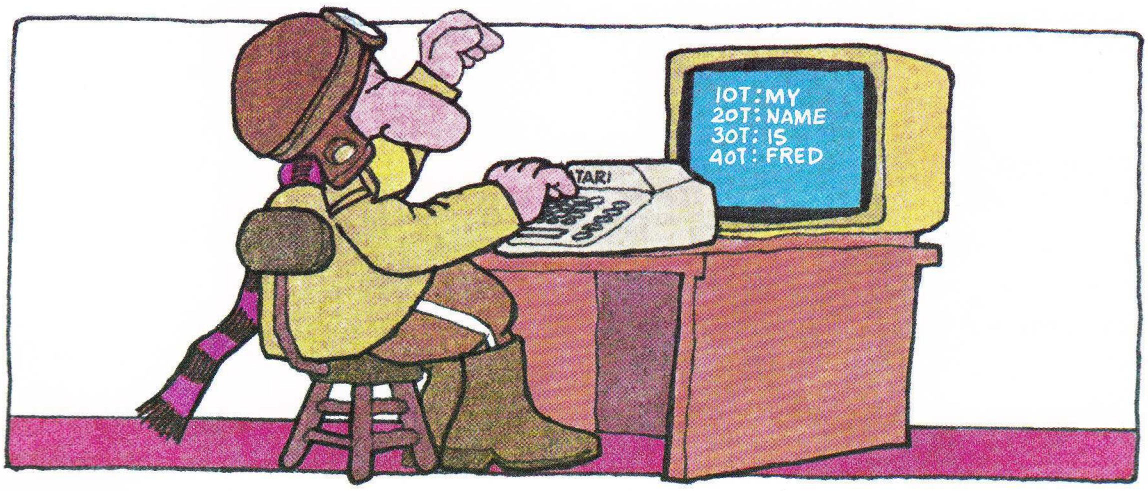

```
READY
LIST
10  R:RIDDLE PROGRAM
20  T:HI.  WHAT'S YOUR NAME?
30  A:$NAME
40  T:OK, $NAME, HERE'S A QUESTION
FOR YOU
50  *GUESSAGAIN
60  T:WHAT KIND OF OF INSECT LIVES ON
THE MOON?
70  A:
80  M:LUNATIC, LUNE-A-TIC, LOONATIC,
LOON- A-TIC
90  TN:NOPE.  GUESS AGAIN.
100  JN:*GUESSAGAIN
110  TY:CONGRATULATIONS. YOU GOT IT!
120  T:SEE YOU AROUND.
130  E:
■
```

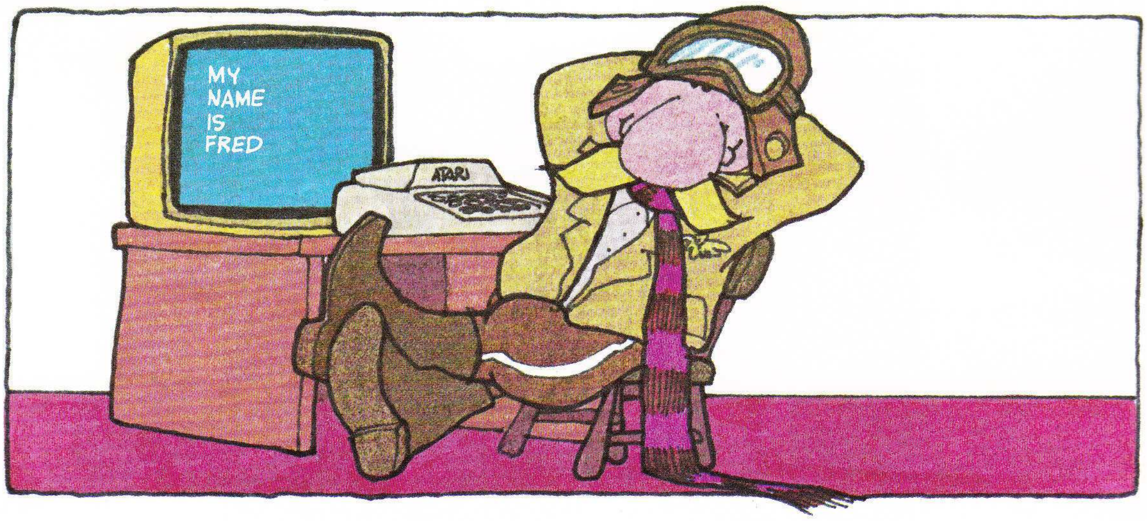
Figure 3-3

You can tell the AUTO command where to start numbering and by how much to increase each new line number. For example, AUTO 100,20 tells the computer to begin numbering at 100 and increase each line by 20 (if you leave off the second number, it will automatically increase by 10's).

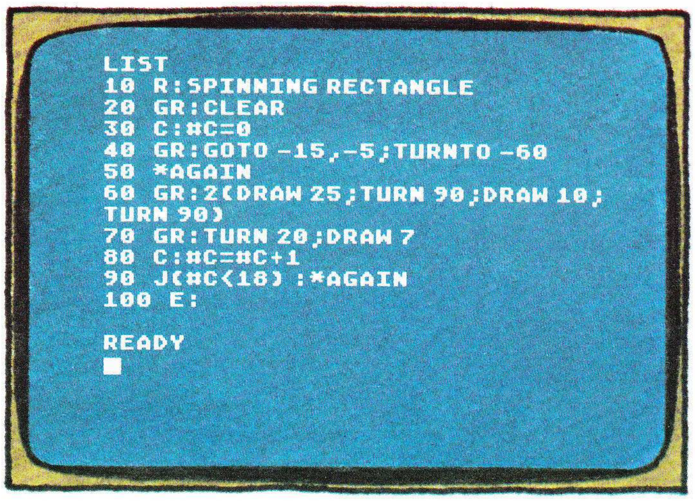
RUN

Run a Program



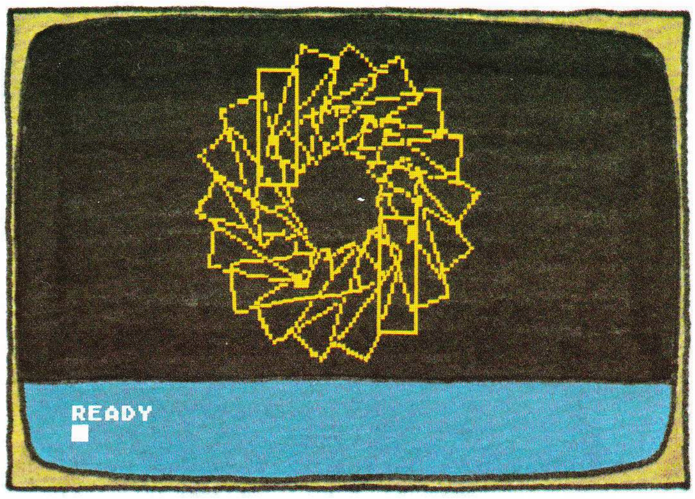


The RUN command tells the computer to execute the program currently in memory. Figure 3-4 shows a listed program and Screen 3-5 shows the program after RUN is typed.



```
LIST
10  R:SPINNING RECTANGLE
20  GR:CLEAR
30  C:#C=0
40  GR:GOTO -15,-5;TURNTO -60
50  *AGAIN
60  GR:2(DRAW 25;TURN 90;DRAW 10;
TURN 90)
70  GR:TURN 20;DRAW 7
80  C:#C=#C+1
90  J(#C<18) :*AGAIN
100  E:

READY
█
```

Figure 3-4

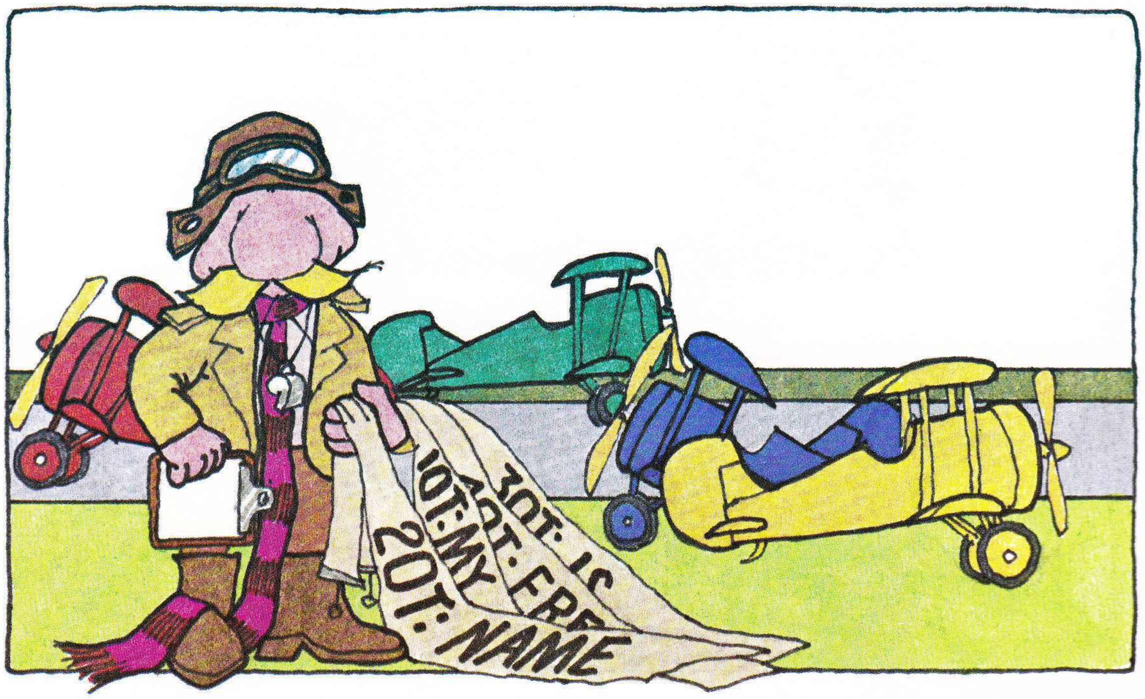


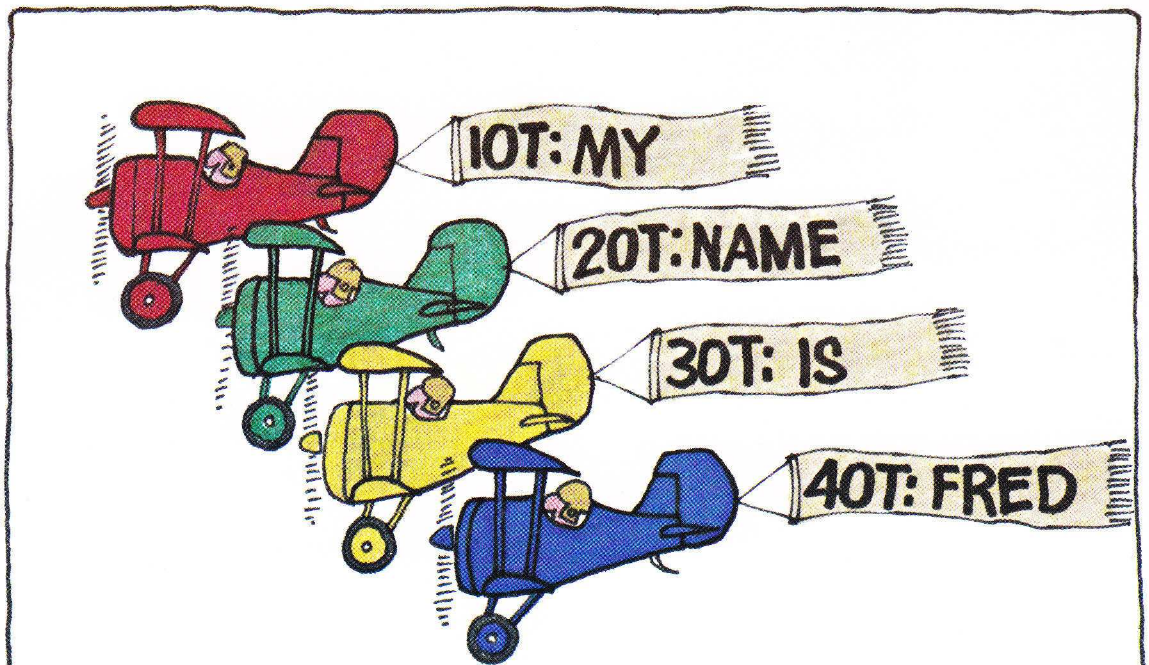
```
READY
█
```

Figure 3-5

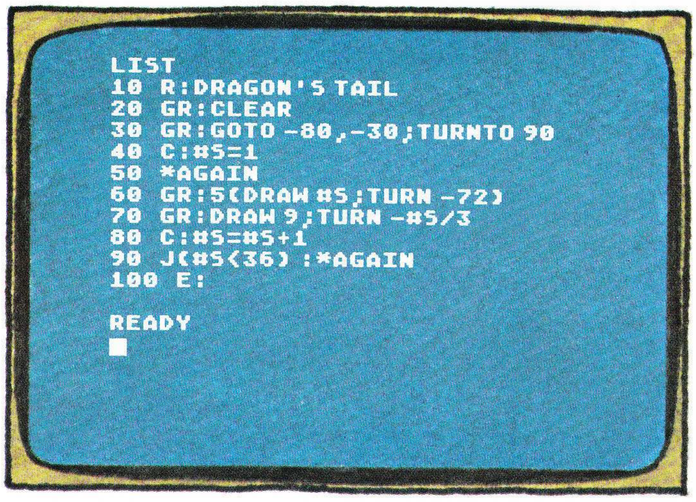
LIST

List a Program





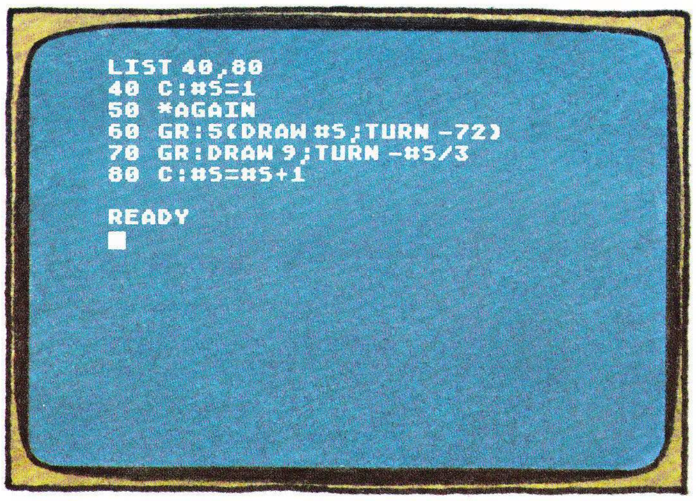
LIST tells the computer to display the program currently stored in memory. Use this command anytime you need to see the entire program, a few lines of the program, or just one line of the program. For example, Figure 3-6 shows the listing of an entire program.



```
LIST
10 R:DRAGON'S TAIL
20 GR:CLEAR
30 GR:GOTO -80,-30;TURNTO 90
40 C:#S=1
50 *AGAIN
60 GR:5(DRAW #S;TURN -72)
70 GR:DRAW 9;TURN -#S/3
80 C:#S=#S+1
90 J(#S<36) :*AGAIN
100 E:

READY
█
```

Figure 3-6

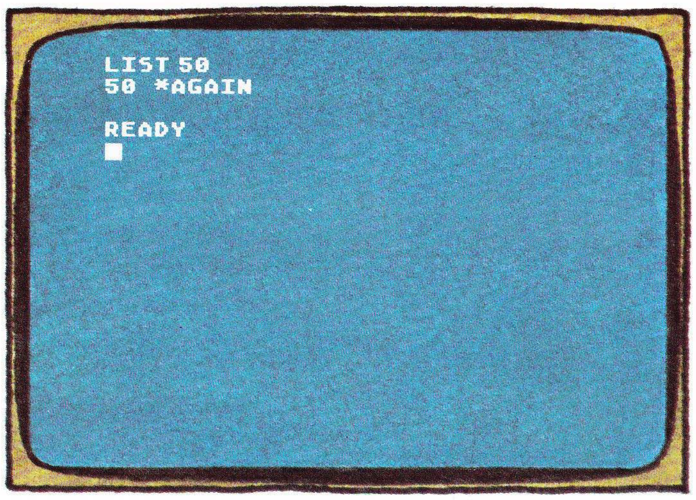
The command LIST 40,80 tells the computer to start listing at line 40 and end at line 80, as you see in Figure 3-7.



```
LIST 40,80
40 C:#S=1
50 *AGAIN
60 GR:5(DRAW #S;TURN -72)
70 GR:DRAW 9;TURN -#S/3
80 C:#S=#S+1

READY
█
```

Figure 3-7

The command LIST 50 tells the computer to LIST only line 50, as you see in Figure 3-8.



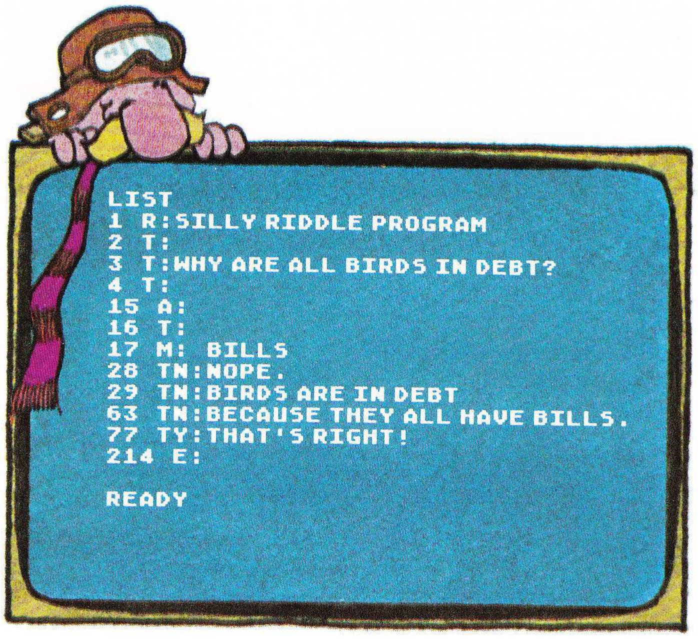
```
LIST 50
50 *AGAIN

READY
■
```

Figure 3-8

Note: To get a printed listing of your program on paper, you must SAVE your program to the printer (see the SAVE command).

REN
Renumber a Program

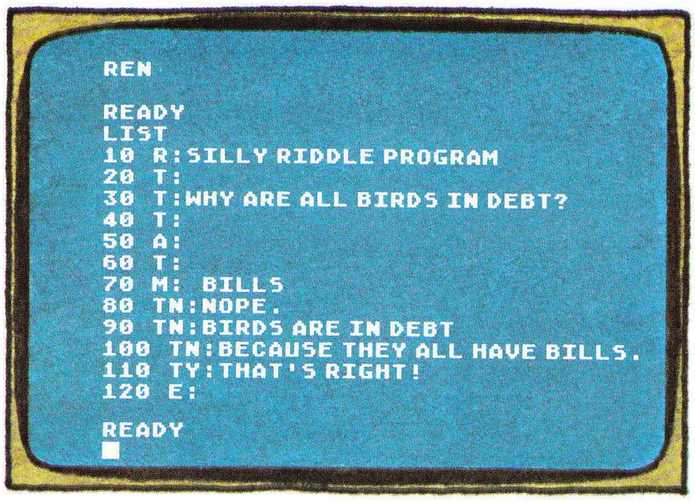
The REN command renumbers the program statements, beginning with the first line of the program. It renumbers the lines by 10's, starting at 10, so you can add more lines of program if necessary. Figure 3-9 shows a program before renumbering.



```
LIST
1 R:SILLY RIDDLE PROGRAM
2 T:
3 T:WHY ARE ALL BIRDS IN DEBT?
4 T:
15 A:
16 T:
17 M: BILLS
28 TN:NOPE.
29 TN:BIRDS ARE IN DEBT
63 TN:BECAUSE THEY ALL HAVE BILLS.
77 TY:THAT'S RIGHT!
214 E:

READY
```

Figure 3-9

Figure 3-10 shows the same program renumbered by 10's starting with 10.

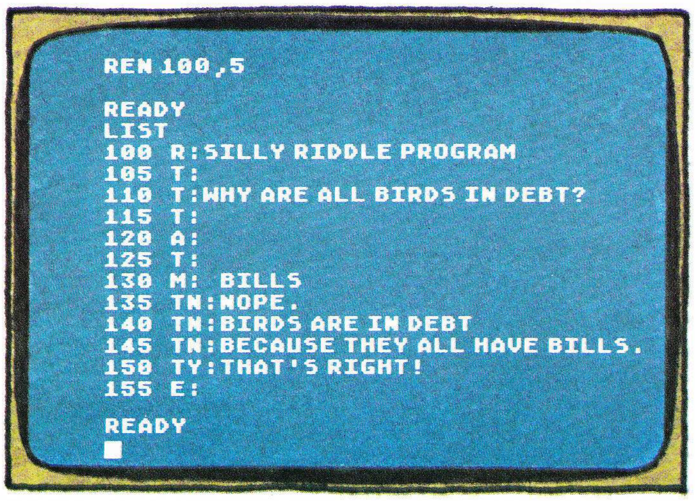


```
REN

READY
LIST
10 R:SILLY RIDDLE PROGRAM
20 T:
30 T:WHY ARE ALL BIRDS IN DEBT?
40 T:
50 A:
60 T:
70 M: BILLS
80 TN:NOPE.
90 TN:BIRDS ARE IN DEBT
100 TN:BECAUSE THEY ALL HAVE BILLS.
110 TY:THAT'S RIGHT!
120 E:

READY
█
```

Figure 3-10

REN will also let you begin renumbering at any number and increase each statement by any amount you want. For example, **REN 100,5** will renumber your program statements starting at 100 and increase each line by 5's.

Figure 3-11 shows the program in Figure 3-10 renumbered, starting at 100, increasing by 5's.

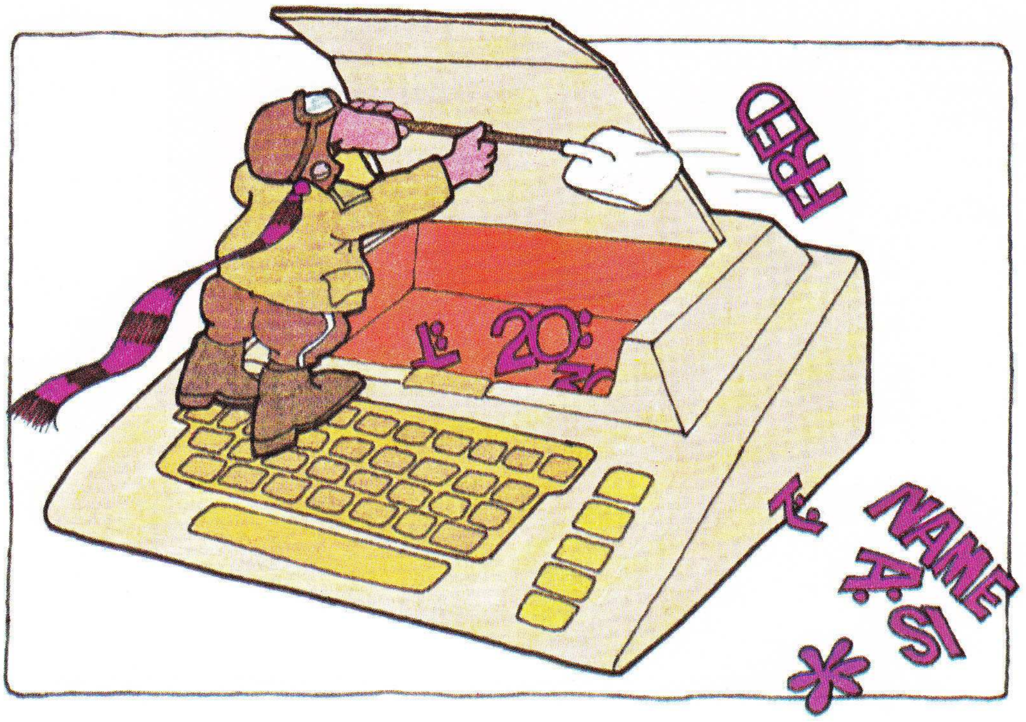


```
REN 100,5

READY
LIST
100  R:SILLY RIDDLE PROGRAM
105  T:
110  T:WHY ARE ALL BIRDS IN DEBT?
115  T:
120  A:
125  T:
130  M:  BILLS
135  TN:NOPE.
140  TN:BIRDS ARE IN DEBT
145  TN:BECAUSE THEY ALL HAVE BILLS.
150  TY:THAT'S RIGHT!
155  E:

READY
█
```

Figure 3-11

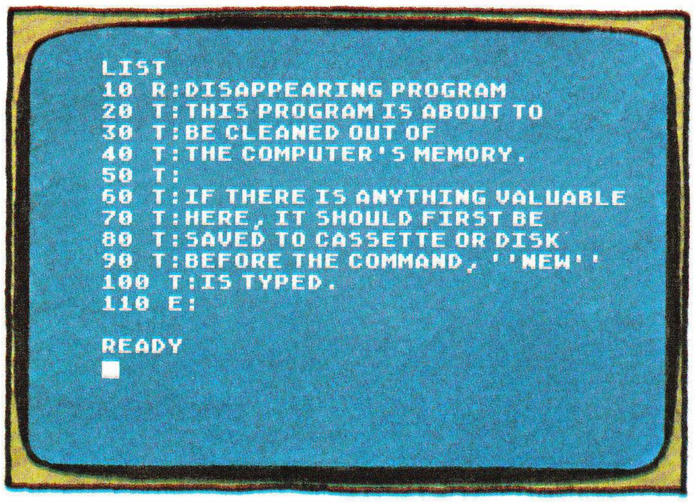
NEW

New Program



The NEW command lets you erase the program in the computer memory. You must be careful when you use this command. Make sure you do not want the program any more before typing **NEW**.

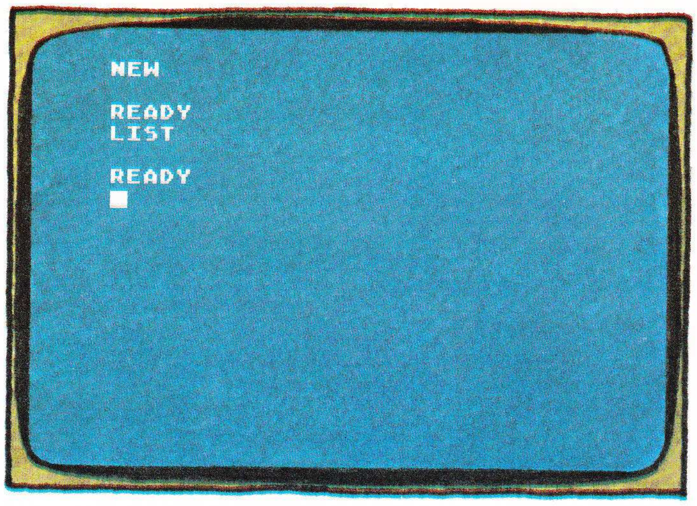
Figure 3-12 shows the listing of a program before **NEW** is typed.



```
LIST
10  R:DISAPPEARING PROGRAM
20  T:THIS PROGRAM IS ABOUT TO
30  T:BE CLEANED OUT OF
40  T:THE COMPUTER'S MEMORY.
50  T:
60  T:IF THERE IS ANYTHING VALUABLE
70  T:HERE, IT SHOULD FIRST BE
80  T:SAVED TO CASSETTE OR DISK
90  T:BEFORE THE COMMAND, ''NEW''
100  T:IS TYPED.
110  E:

READY
█
```

Figure 3-12

Figure 3-13 shows the attempted listing of a program after **NEW** is typed.



```
NEW

READY
LIST

READY
█
```

Figure 3-13

VNEW

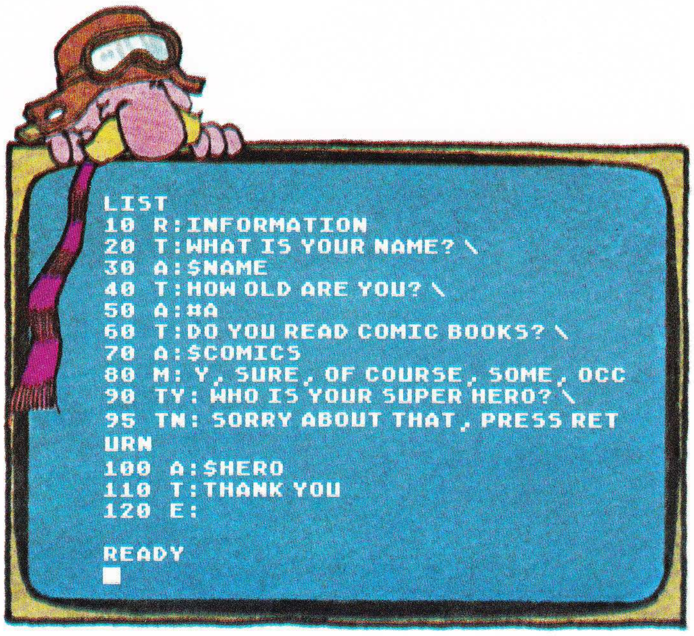
New Variables

The VNEW: command clears the numeric and/or string variables by using the command as follows:

VNEW:$	(clears the string variables)
VNEW:#	(clears the numeric variables)
VNEW:	(clears both string and numeric variables)

The VNEW: command does not erase the program from memory.

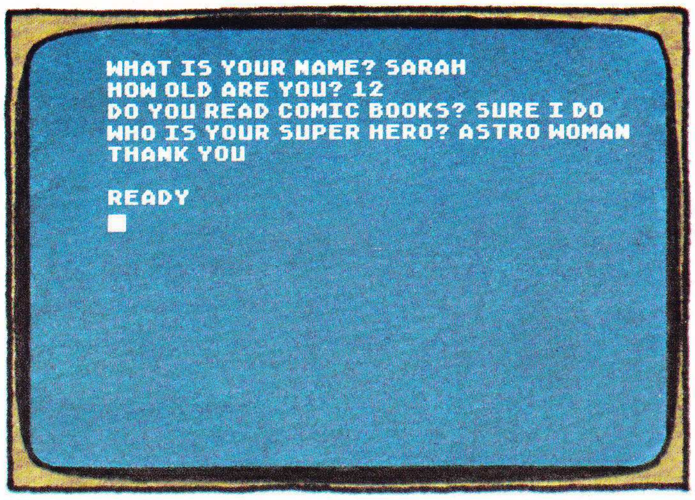
Figure 3-14 shows a program with string and numeric variables.



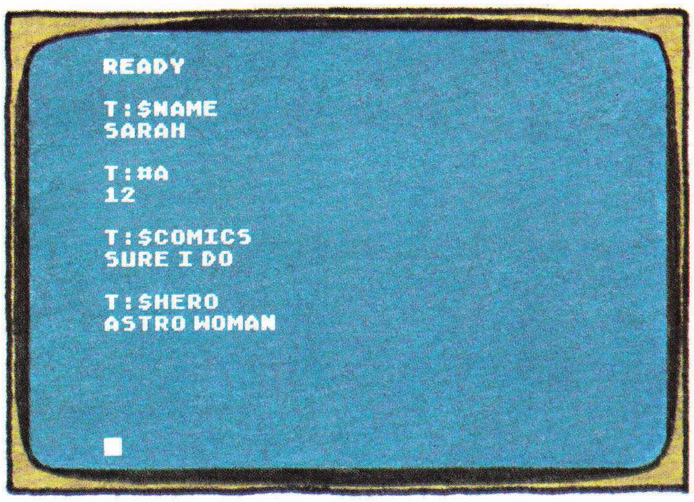
```
LIST
10 R:INFORMATION
20 T:WHAT IS YOUR NAME? \
30 A:$NAME
40 T:HOW OLD ARE YOU? \
50 A:#A
60 T:DO YOU READ COMIC BOOKS? \
70 A:$COMICS
80 M: Y, SURE, OF COURSE, SOME, OCC
90 TY: WHO IS YOUR SUPER HERO? \
95 TN: SORRY ABOUT THAT, PRESS RET
URN
100 A:$HERO
110 T:THANK YOU
120 E:

READY
█
```

Figure 3-14

Figure 3-15 shows the variables with values assigned.



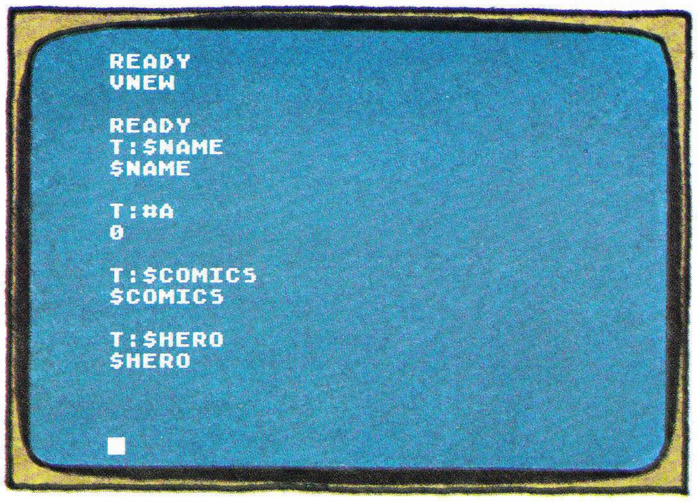
```
WHAT IS YOUR NAME? SARAH
HOW OLD ARE YOU? 12
DO YOU READ COMIC BOOKS? SURE I DO
WHO IS YOUR SUPER HERO? ASTRO WOMAN
THANK YOU

READY
■
```



```
READY

T:$NAME
SARAH

T:#A
12

T:$COMICS
SURE I DO

T:$HERO
ASTRO WOMAN



■
```

Figure 3-15

Figure 3-16 shows the variables after **VNEW** has been typed.



```
READY
VNEW

READY
T:$NAME
$NAME

T:#A
0

T:$COMICS
$COMICS

T:$HERO
$HERO
```

Figure 3-16

DUMP

Dump String Variables

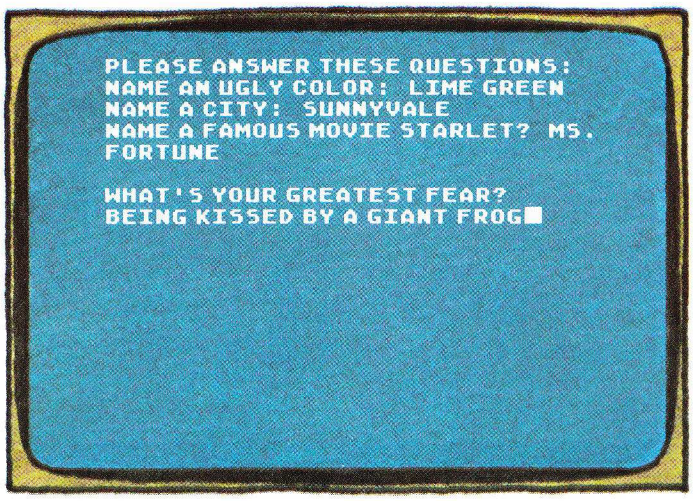
If you want to see the contents of all the string variables in the computer's memory, type **DUMP** and press the RETURN key. The DUMP command does not clear the memory, and therefore is helpful when you want to find errors in your program.

The program in Figure 3-17 is a listing of a program.

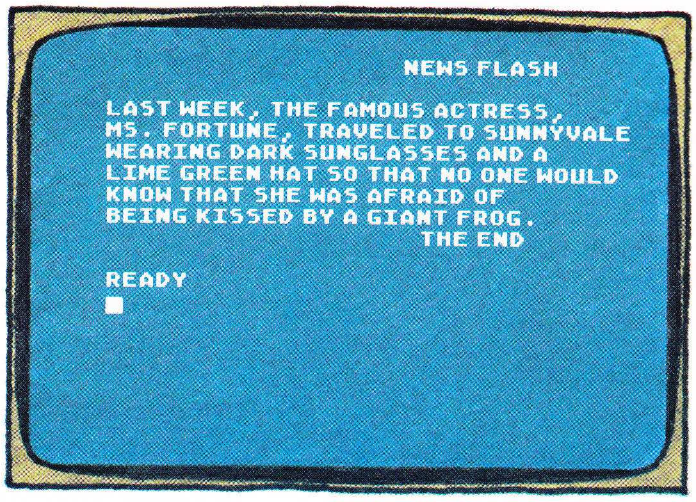
```
10  R:A MAD LIB
20  T:PLEASE ANSWER THESE
QUESTIONS:
30  T:NAME AN UGLY COLOR: \
40  A:$COLOR
50  T:NAME A CITY: \
60  A:$CITY
70  T:NAME A FAMOUS MOVIE STARLET?  \
80  A:$STARLET
90  T:WHAT'S YOUR GREATEST FEAR?
100  A:$FEAR
110  T:↰ [CLEAR SCREEN]
120  T:            NEWS FLASH
130  T:
140  T:LAST WEEK, THE FAMOUS
ACTRESS,
150  T:$STARLET, TRAVELED TO $CITY
160  T:WEARING DARK SUNGLASSES AND A
170  T:$COLOR HAT SO THAT NO ONE
WOULD
180  T:KNOW THAT SHE WAS AFRAID OF
190  T:$FEAR.
200  T:              THE END
210  E:

READY


■
```

Figure 3-17

Figure 3-18 is a RUN of that program.



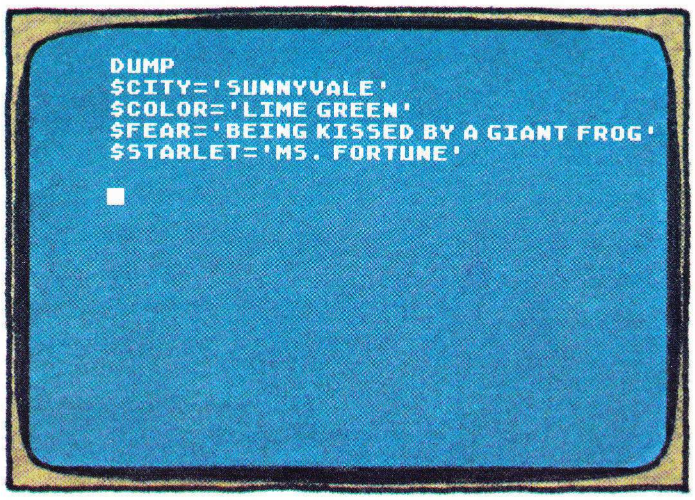
```
PLEASE ANSWER THESE QUESTIONS:
NAME AN UGLY COLOR:  LIME GREEN
NAME A CITY:  SUNNYVALE
NAME A FAMOUS MOVIE STARLET?  MS.
FORTUNE

WHAT'S YOUR GREATEST FEAR?
BEING KISSED BY A GIANT FROG■
```



```
                  NEWS FLASH

LAST WEEK, THE FAMOUS ACTRESS,
MS. FORTUNE, TRAVELED TO SUNNYVALE
WEARING DARK SUNGLASSES AND A
LIME GREEN HAT SO THAT NO ONE WOULD
KNOW THAT SHE WAS AFRAID OF
BEING KISSED BY A GIANT FROG.
                       THE END

READY
■
```

Figure 3-18

Figure 3-19 is a DUMP of the variables in that program.

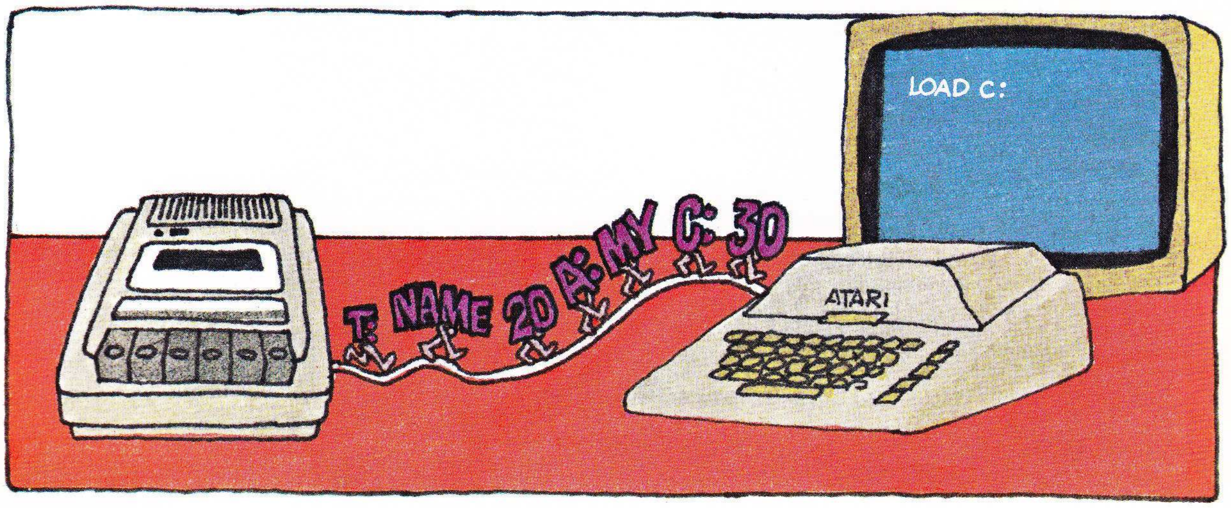


```
DUMP
$CITY='SUNNYVALE'
$COLOR='LIME GREEN'
$FEAR='BEING KISSED BY A GIANT FROG'
$STARLET='MS. FORTUNE'

■
```

Figure 3-19

LOAD

Load a Program



The LOAD command tells the computer to LOAD a program you have saved on cassette or diskette into the computer memory.

LOAD C:

Loading From Cassette

In order to LOAD a program from cassette (indicated by C:), type **LOAD C:** and press the RETURN key.

The computer will beep, signaling you to position the cassette tape and press the PLAY button on the cassette recorder. When you are ready, press RETURN on the computer console to start the tape loading. If you turn up the volume control on your television set, you will soon hear a series of beeps. This tells you that the cassette is loading properly.

LOAD D: FILENAME
Loading From Diskette

If you are loading a program saved on a diskette, you must know the name of the program. For example, if the name of the program you want to LOAD is COLORS.PLT, you would type **LOAD D:COLORS.PLT** and press the RETURN key. The *D:* tells the computer you are loading from the disk drive. Again, if the volume control is turned up on the television set, you will begin hearing a series of fast beeps telling you that a program is being loaded.

MERGING PROGRAMS

The LOAD command does NOT clear out memory before loading a program. This allows you to merge two programs—that is, load one program into memory on top of another one. When you merge two programs, the program loaded last will write over and replace any statements having *the same* line numbers as the first program.

Note: Unless you want to merge two programs, you should always type **NEW** to clear memory before loading a program.

Figures 3-20 and 3-21 show two different programs, Program One and Program Two.

```
5  R:PROGRAM ONE
10  R:CITYSCAPE
20  GR:CLEAR
30  U:*BUILDINGS
40  U:*SKY
50  T:        THE CITY AT NIGHT
80  E:
100  *BUILDINGS
110  GR:GOTO 75,-31;CLEAR
120  C:#D=0
130  C:#L=1
140  *NEXTBUILD
150  C:#C=?\3+1 [RANDOM COLOR]
160  J(#C=#L):*NEXT BUILD
170  GR(#C=1):PEN RED
180  GR(#C=2):PEN YELLOW
190  GR(#C=3):PEN BLUE
200  C:#X=?\15+5 [RANDOM BLDG
WIDTH]
210  C:#Y=?\30+5 [RANDOM BLDG
HEIGHT]
220  GR:TURNTO 0;DRAW #Y;TURNTO
180;GO #Y
230  GR:TURNTO -90;GO #X;TURNTO 0;
FILL #Y
240  C:#D=#D+#X
250  C:#L=#C
260  GR:TURNTO 180;GO #Y
270  J(#D<150):*NEXTBUILD
300  *SKY
310  GR:PEN YELLOW
320  C:#N=0
330  *MORE
340  C:#X=?\158-79
350  C:#Y=?\50+5
360  GR:GOTO #X,#Y
370  GR:5(DRAW 4;TURN 144)
380  C:#N=#N+1
390  J(#N<20):*MORE
400  E:

READY
■
```

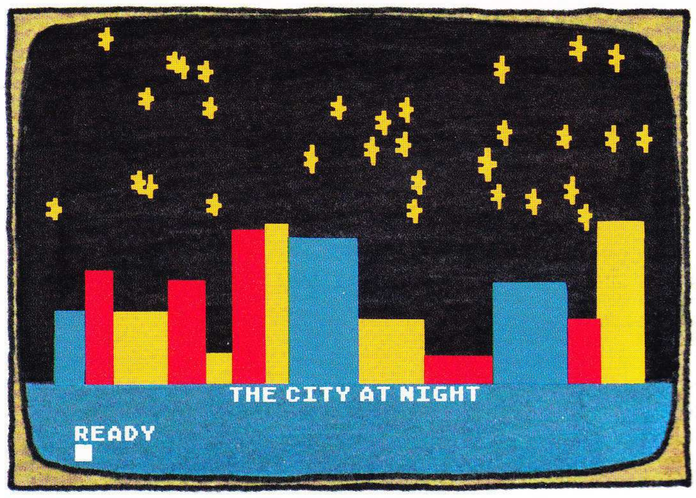
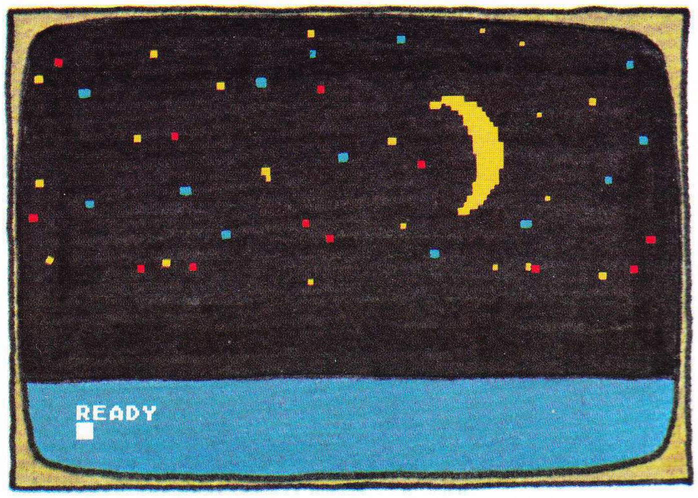


Figure 3-20

```
5  R:PROGRAM TWO
300  *SKY
310  GR:GOTO 30,40
320  U:*MOON
330  C:#S=0
340  *MORESTARS
350  C:#X=?\158-79
360  C:#Y=?\50+5
370  C:#C=?\3+1
380  GR(#C=1):PEN RED
390  GR(#C=2):PEN BLUE
400  GR(#C=3):PEN YELLOW
410  GR:GOTO #X,#Y
420  C:#S=#S+1
430  J(#S<70):*MORESTARS
440  E:
450  *MOON
460  GR:PEN YELLOW
470  GR:TURNTO 70
480  GR:38(DRAW 1;TURN 5)
490  GR:TURNTO 50
500  GR:5(TURN -20;FILL 5)
510  E:

READY
■
```

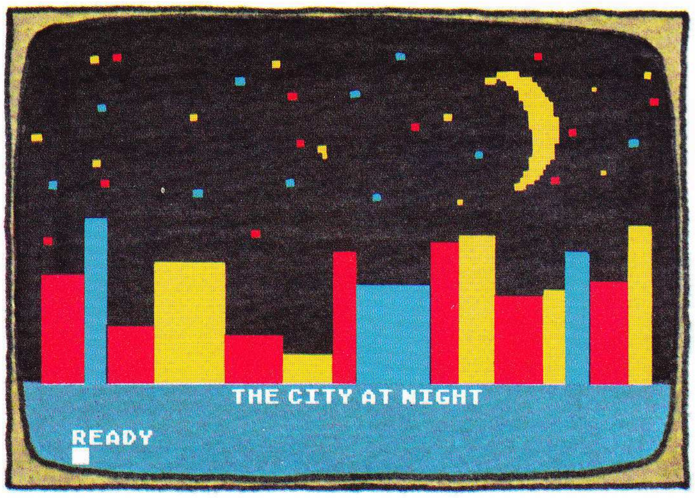


```
READY
■
```

Figure 3-21

Figure 3-22 shows what happens if you LOAD Program Two on top of Program One.

```
5  PROGRAM TWO
10  R:CITYSCAPE
20  GR:CLEAR
30  U:*BUILDINGS
40  U:*SKY
50  T:          THE CITY AT NIGHT
80  E:
100  *BUILDINGS
110  GR:GOTO 75;-31;CLEAR
120  C:#D=0
130  C:#L=1
140  *NEXTBUILD
150  C:#C=?\3+1 [RANDOM COLOR]
160  J(#C=#L):*NEXTBUILD
170  GR(#C=1):PEN RED
180  GR(#C=2):PEN YELLOW
190  GR(#C=3):PEN BLUE
200  C:#X=?\15+5 [RANDOM BLDG
WIDTH]
210  C:#Y=?\30+5 [RANDOM BLDG
HEIGHT]
220  GR:TURNTO 0;DRQW #Y;TURNTO
180;GO #Y
230  GR:TURNTO -90;GO #X;TURNTO 0;
FILL #Y
240  C:#D=#D+#X
250  C:#L=#C
260  GR:TURNTO 180;GO #Y
270  J(#D<150):*NEXTBUILD
300  *SKY
310  GR:GOTO 30,40
320  U:*MOON
330  C:#S=0
340  *MORESTARS
350  C:#X=?\158-79
360  C:#Y=?\50+5
370  C:#C=?\3+1
380  GR(#C=1):PEN RED
390  GR(#C=2):PEN BLUE
400  GR(#C=3):PEN YELLOW
410  GR:TOTO #X,#Y
420  C:#S=#S+1
430  J(#S<70):*MORESTARS
440  E:
450  *MOON
460  GR:PEN YELLOW
470  GR:TURNTO 70
480  GR:38(DRAW 1;TURN 5)
490  GR:TURNTO 50
500  GR:5(TURN -20;FILL 5)
510  E:

READY
■
```

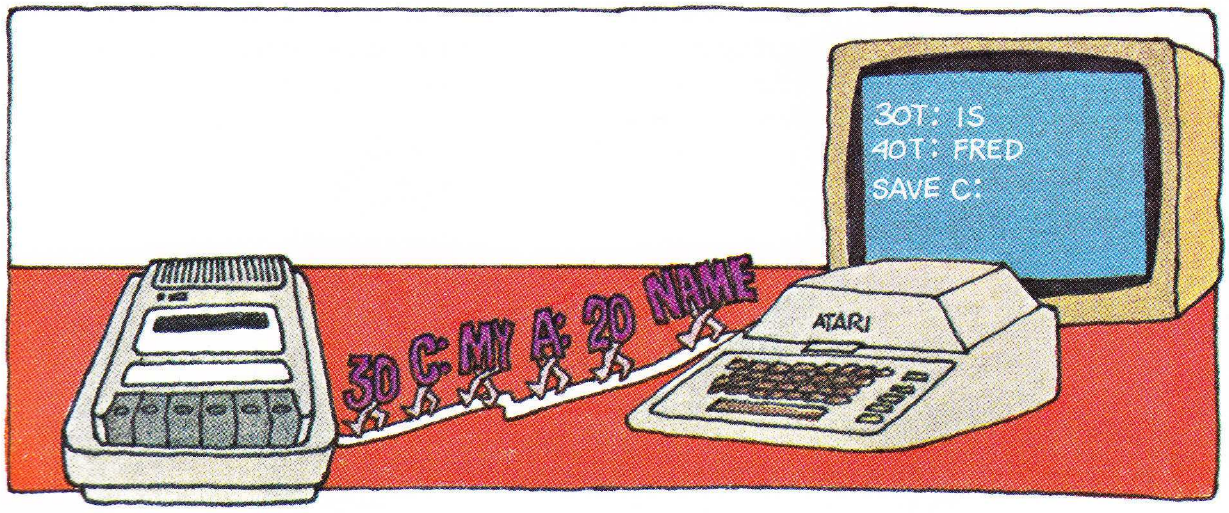


```
THE CITY AT NIGHT
READY
■
```

Figure 3-22

SAVE

Save a Program



The SAVE command lets you save all or any part of a program you have written. In order to SAVE a program, you must have either a cassette or disk drive attached to your computer. You can get a printout by saving your program to the printer.

SAVE C:

Save to Cassette

To SAVE a program to cassette, type **SAVE C:**. The computer will beep twice, which tells you to position the cassette type, and press the PLAY and RECORD buttons (at the same time). Then press the RETURN key on the computer console to begin saving your program. As this occurs, you will hear a high-pitched hum and a series of low beeps. When you SAVE to cassette in PILOT, you cannot give the program a name. You must make a note as to the location of the program on the tape by the numbers shown in the counter window on the recorder. Then when you want to LOAD the program back into memory, you must first move the tape until that number shows in the counter window.

If you want to SAVE only part of your program, you need to type the line numbers you want saved. For example, typing **SAVE C:10,50** will SAVE lines 10 through 50 to the cassette.

SAVE D:FILENAME

Save to Diskette

If you are saving to a diskette, you must give the program a name of up to seven characters. You can extend the program name by adding a period and three more characters. This is helpful if you have more than one version of a program. For example, if Version 3 of your COLOR program was in memory and you wanted to SAVE it on a diskette, you might type:

```
SAVE D:COLORS.V03
```

The disk drive will start humming, and if your television volume control is turned up slightly, you will hear the beeps telling you that the program is being saved.

If you are saving only part of the program to diskette, you need to type in the lines you want saved after the program name. For example, typing **SAVE D:MODULE1 10,50** will SAVE only lines 10 through 50 on the diskette under the program name, "MODULE1."

SAVE P:

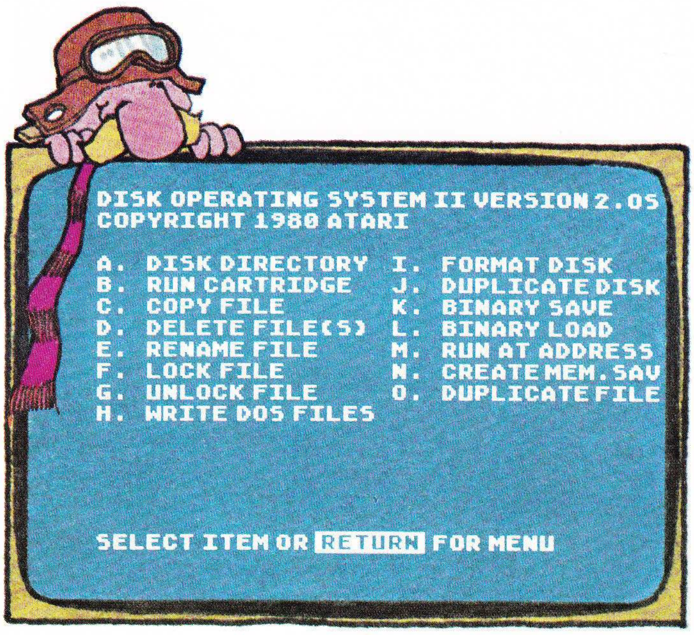
Save to Printer

To get a listing of part or all of your program printed on paper instead of the television screen, you must SAVE your program to the printer. To get a listing of your whole program, turn on the printer, and type **SAVE P:**. If you only want a listing of lines 20 through 50, type **SAVE P:20,50**.

DOS

Disk Operating System

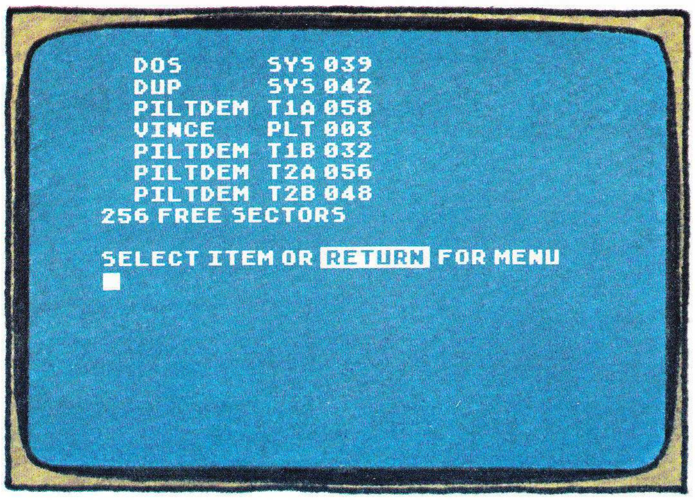
When you want to work with your disk files for any reason, type **DOS** and press RETURN. This will bring up the DOS Menu (Figure 3-23) to your television screen.



```
DISK OPERATING SYSTEM II VERSION 2.0S
COPYRIGHT 1980 ATARI

A. DISK DIRECTORY    I. FORMAT DISK
B. RUN CARTRIDGE     J. DUPLICATE DISK
C. COPY FILE         K. BINARY SAVE
D. DELETE FILE(S)    L. BINARY LOAD
E. RENAME FILE       M. RUN AT ADDRESS
F. LOCK FILE         N. CREATE MEM.SAV
G. UNLOCK FILE       O. DUPLICATE FILE
H. WRITE DOS FILES

SELECT ITEM OR RETURN FOR MENU
```

Figure 3-23

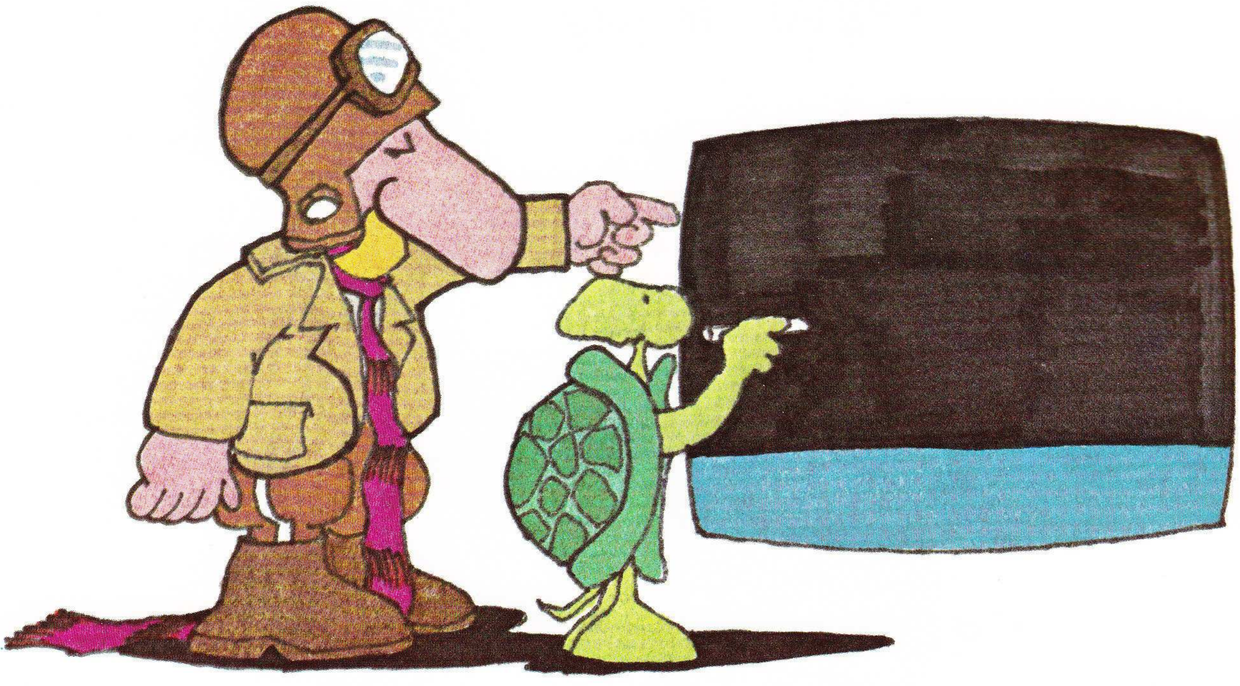
Select the function you want to perform from the DOS Menu, type the letter for that function, and press RETURN. For example, if you want to see what files are in the Disk Directory, you would type the letter **A** (see DOS Menu) and press RETURN twice. You will then see a listing of all the files in the directory (Figure 3-24). Our sample Disk Directory is listed on the following page; yours will have different filenames.



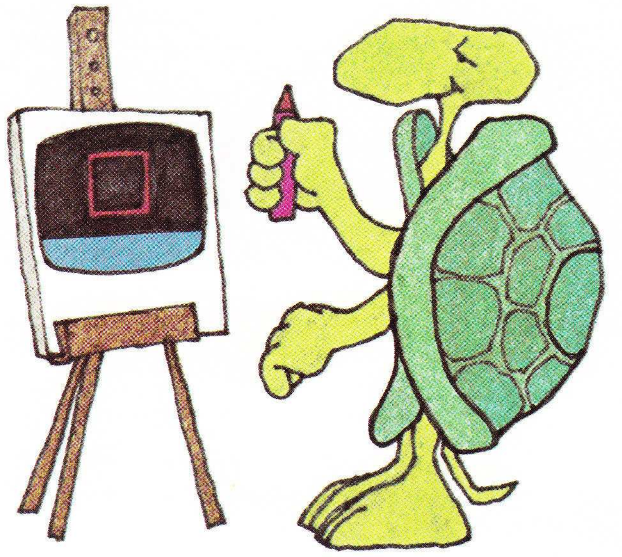
```
  DOS      SYS 039
  DUP      SYS 042
  PILTDEM  T1A 058
  VINCE    PLT 003
  PILTDEM  T1B 032
  PILTDEM  T2A 056
  PILTDEM  T2B 048
256 FREE SECTORS

SELECT ITEM OR RETURN FOR MENU
█
```

Figure 3-24

4 GRAPHICS



GR:
Graphics



The GR: (GRAPHICS) command tells the computer you are going into the Graphics mode to draw a picture. Figure 4-1 shows you what the basic Graphics screen looks like.

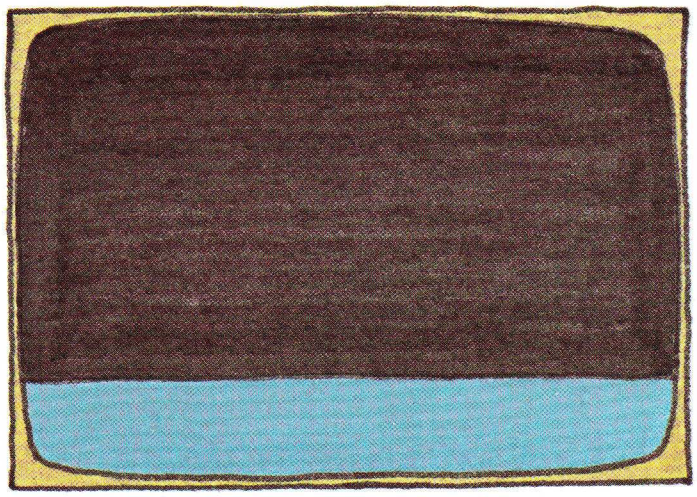


Figure 4-1

The black background is for drawing, and the blue strip at the bottom of the screen (the *text window*) is for displaying messages or writing commands.

TURTLE GRAPHICS

The method by which you draw is called *turtle graphics*. Imagine that the screen is the playground of an invisible turtle who draws on the screen with a tiny pen. The GRAPHICS commands tell the turtle where to go and what to do with the pen.

The GR: command does not draw anything by itself; it must be used with Graphics Subcommands. For example, the PEN Subcommand tells the turtle either to lift the PEN or to draw in one of four colors. The Graphics Subcommands are explained in more detail on the pages to follow.

Commanding the turtle to move is like giving someone directions to the nearest gas station: either you give directions **where** (go to Fourth and Main St.), or you give specific directions **which way and how far** (go 3 blocks to Fourth, take a right and go 4 blocks to Main St.).

WHERE TO GO – X AND Y COORDINATES

The Graphics screen is really an invisible grid of numbered lines which looks like a piece of graph paper. The center of the screen is a point (called the *origin*) where the X and Y axes cross.

The X axis goes horizontally from the left (-79) to right (+79). The Y axis goes vertically from the bottom of the screen (-47) to the top of the screen (+47). The bottom of the screen includes the text window.

Every point on the screen is named by two numbers (X,Y). These two numbers are called the **X and Y coordinates** of the point. They tell the distance horizontally and vertically from the point to the origin. Figure 4-2 shows the screen layout with some labeled points.

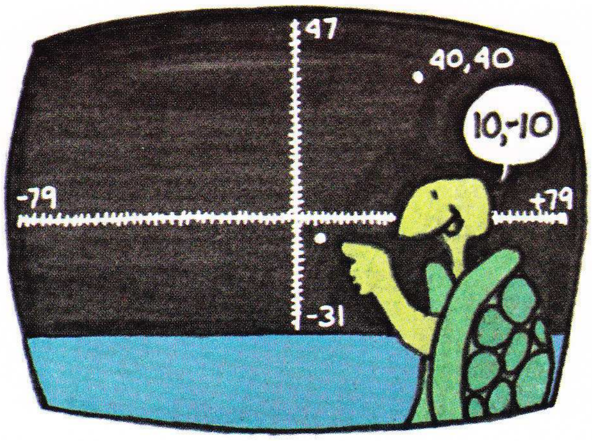


Figure 4-2

The Graphics Subcommands GOTO, DRAWTO, and FILLTO use coordinates to tell the turtle where to go.

WHAT DIRECTION AND HOW FAR TO GO

The turtle is always at the center of an invisible circle which moves with him wherever he goes. This circle is divided into 360 one-degree angles, with the 0-degree angle pointing up.

The turtle is always pointing in some direction within this circle, and can turn to the right (+) or left (-) in 360 different angles. Figure 4-3 shows some different headings. The Graphics Subcommands TURNTO and TURN, change the turtle's direction.

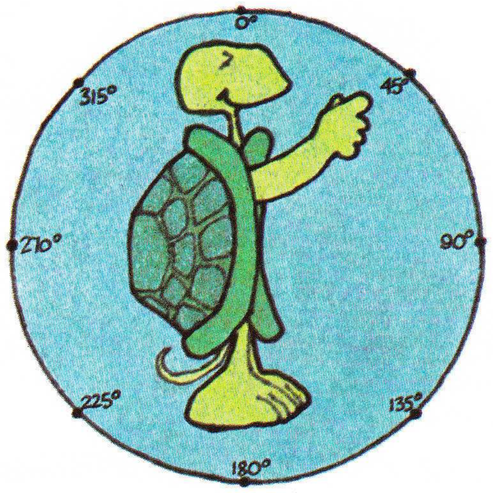


Figure 4-3

Each step the turtle takes, either forward or backward, is called a **turtle unit**. The units are the same size as the individual units making up the invisible grid; therefore, if the turtle moved from the origin to the right edge of the screen, he would move 80 turtle units. The Graphics Subcommands GO, DRAW, and FILL tell the turtle to move forward or backwards in whatever direction he is facing.

INITIAL LOCATION, HEADING, AND PEN COLOR

When you first enter the Graphics mode, the turtle is located at 0,0 (the origin). He is facing 0 degree (straight up) and the PEN color is yellow.

SHORTCUTS FOR GRAPHICS PROGRAMMING

You can program more than one GRAPHICS subcommand in the same program statement. **Each subcommand MUST be separated by a semicolon (;).** For example:

```
GR:GOTO 0,0; DRAW 10; TURN 45; PEN RED; DRAW 5
```

If you want a series of subcommands repeated a specific number of times, enclose the series of subcommands in parentheses and precede the left parenthesis by the number of times you want that series repeated. For example, the program in Figure 4-4 shows the long way to draw a box. The program in Figure 4-5 shows a short way to to draw the same box.

```
10 R:LONG WAY BOX
20 GR:CLEAR
25 GR:GOTO -15,-5
30 GR:DRAW 30
40 GR:TURN 90
50 GR:DRAW 30
60 GR:TURN 90
70 GR:DRAW 30
80 GR:TURN 90
90 GR:DRAW 30
100 GR:TURN 90
110 E:

READY
■
```

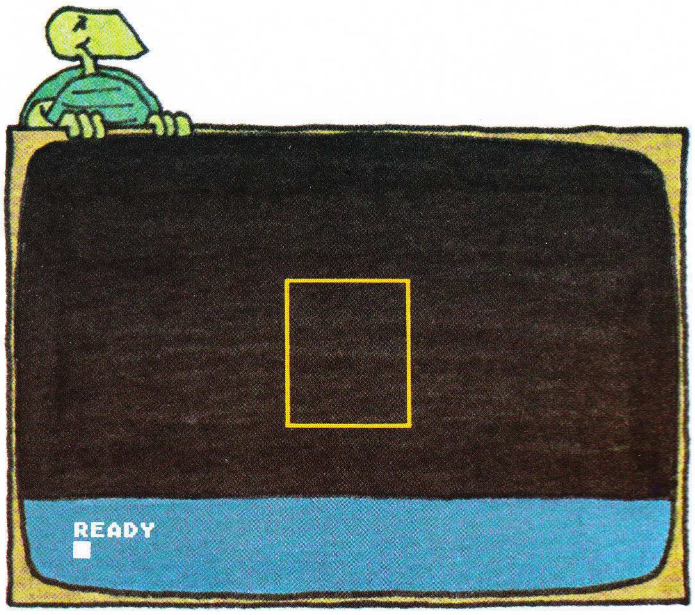


Figure 4-4

```
10 R:SHORT WAY BOX
20 GR:CLEAR
30 GR:GOTO -15,-5
40 GR:4(DRAW 30;TURN 90)
50 E:

READY
■
```

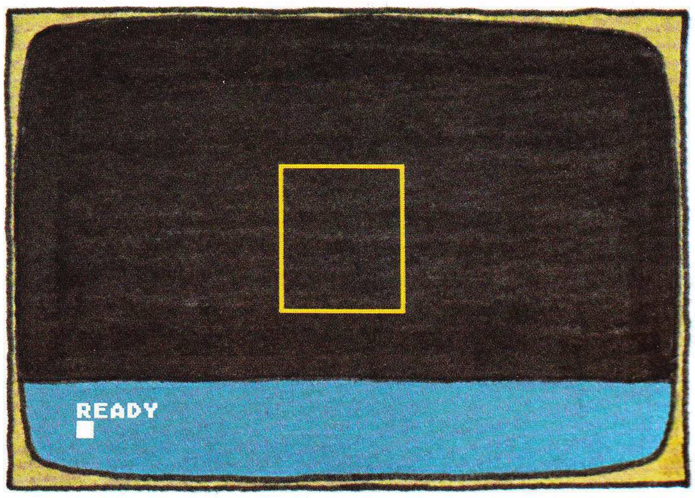


Figure 4-5

GR:CLEAR

Clear the Graphics Screen

GR:CLEAR erases what is on the screen. When you enter a new GR: command the picture begins at the point you left off with the last drawing.

Figures 4-6 through 4-8 show you how GR:CLEAR can be used in a program. The program draws a pentagon, pauses for you to look at it, then clears the screen and draws a star. The program continues to go back and forth between the pentagon and the star, giving you a flashing effect.

```
10  R:THE PENTAGON AND THE STAR
20  GR:CLEAR
30  *LOOP
40  U:*PENTAGON
50  PA:5
60  GR:CLEAR
70  U:*STAR
80  PA:5
90  GR:CLEAR
100  J:*LOOP
110  *PENTAGON
120  GR:GOTO -15,0
130  GR:TURNTO -18
140  GR:5(DRAW 20;TURN 72)
150  E:
160  *STAR
170  GR:GOTO -15,0
180  GR:TURNTO 19
190  GR:5(DRAW 31;TURN 144)
200  E:


READY
■
```

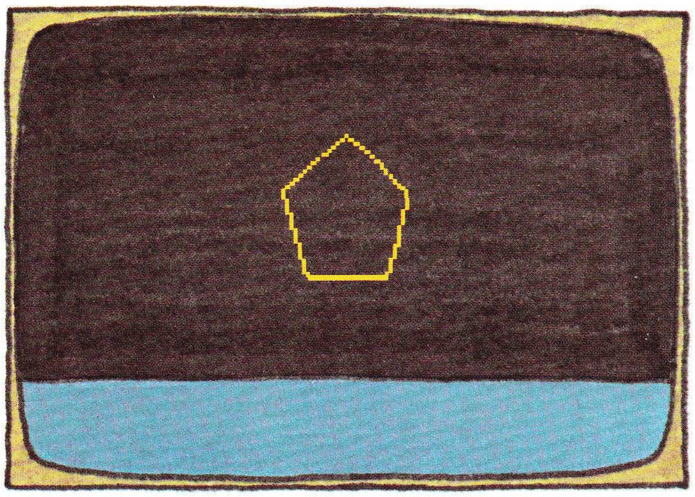


Figure 4-6

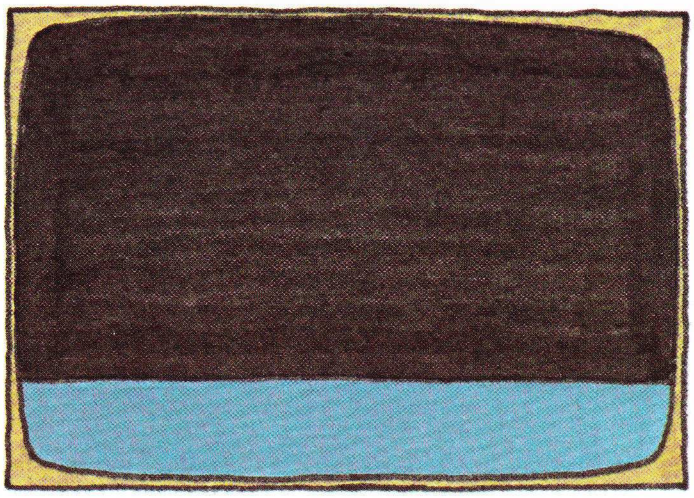


Figure 4-7

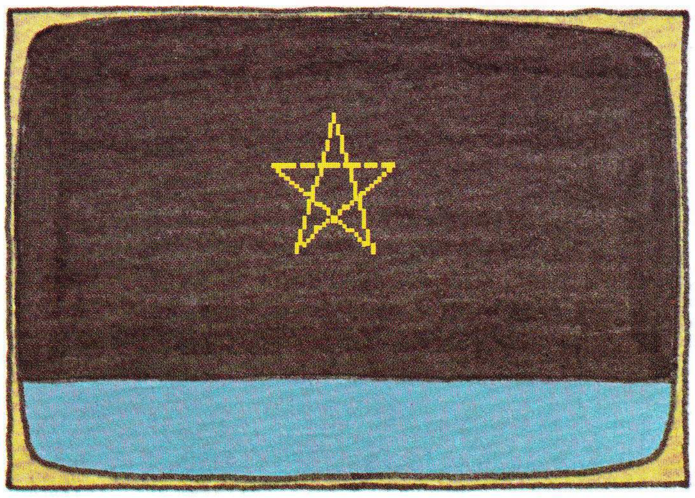
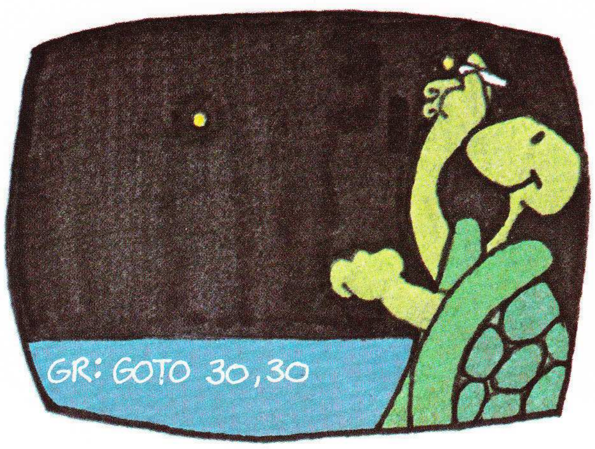


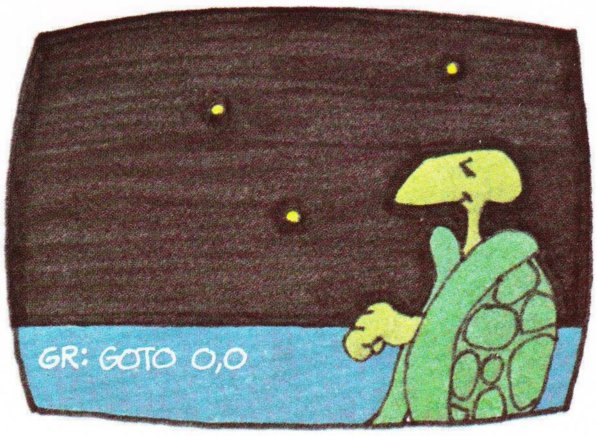
Figure 4-8

GR:GOTO

Go to the Point X,Y

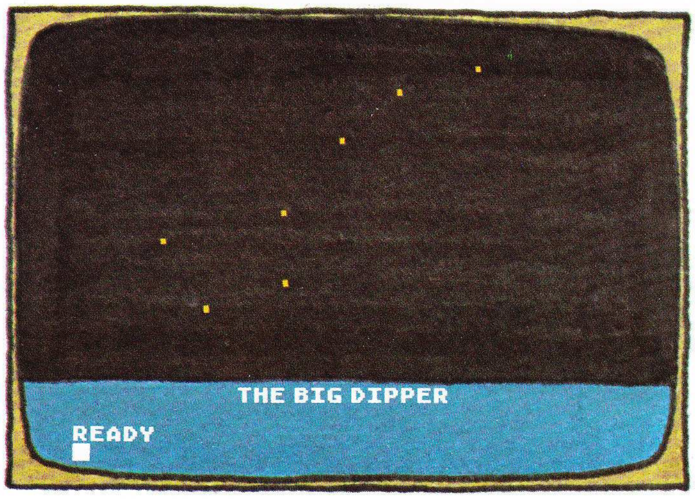
GR:GOTO -20,20





The command GR:GOTO X,Y tells the turtle to pick up the pen, move to the point with coordinates X and Y, and put the pen down there. The pen leaves a dot on the screen at the point (X,Y). Figure 4-9 is a short program that makes "The Big Dipper" using GOTO commands.

```
10 R:THE BIG DIPPER
15 GR:CLEAR
20 GR:GOTO 40,40
30 GR:GOTO 20,35
40 GR:GOTO 5,25
50 GR:GOTO -10,10
60 GR:GOTO -10,-5
70 GR:GOTO -30,-10
80 GR:GOTO -40,5
90 T:         THE BIG DIPPER
100 E:

READY
```

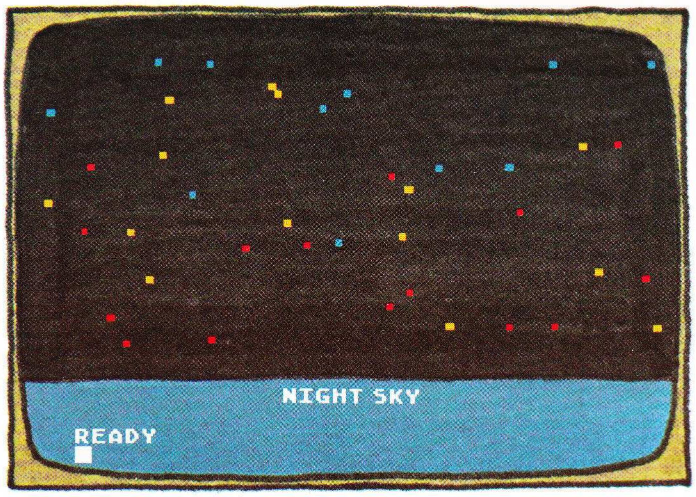


```
THE BIG DIPPER
READY
```

Figure 4-9

The X and Y coordinates in the GOTO command may either be numbers, numeric variables, or numeric expressions. The program in Figure 4-10 uses numeric variables #X and #Y as coordinates. These variables are given random values so the dots appear at random on the screen. (The color of the dots is also chosen at random.)

```
10 R:NIGHT SKY
20 GR:CLEAR
30 T:           NIGHT SKY
40 C:#S=0
50 *MORE
60 C:#X=?\158-79
70 C:#Y=?\47
80 C:#C=?\3+1
90 GR(#C=1):PEN RED
100 GR(#C=2):PEN BLUE
110 GR(#C=3):PEN YELLOW
120 GR:GOTO #X,#Y
130 C:#S=#S+1
140 J(#S<100):*MORE
150 E:

READY
```

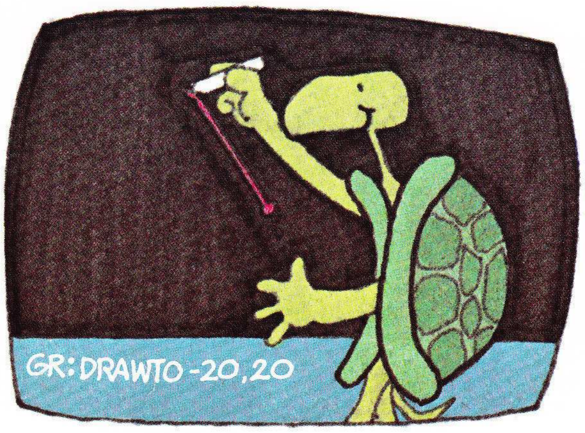


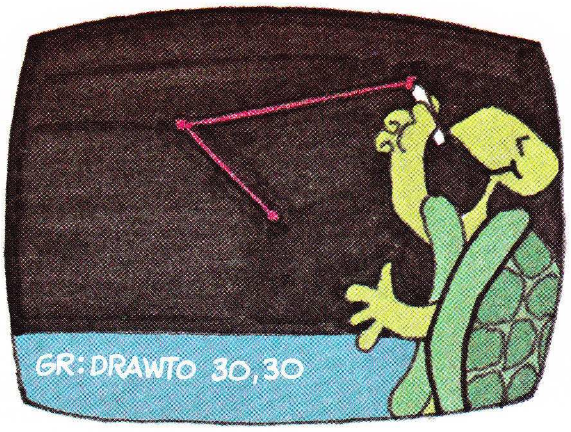
```
NIGHT SKY
READY
```

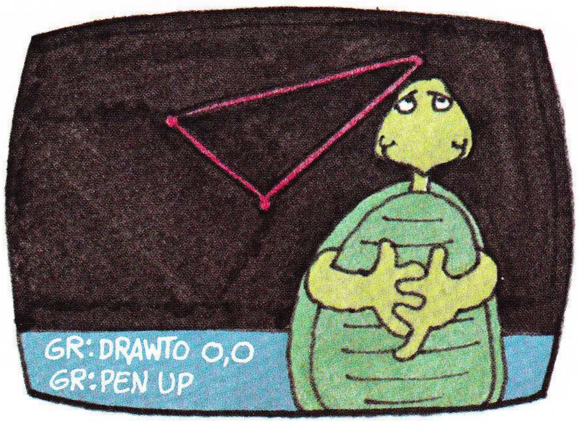
Figure 4-10

GR:DRAWTO

Draw to the Point X,Y

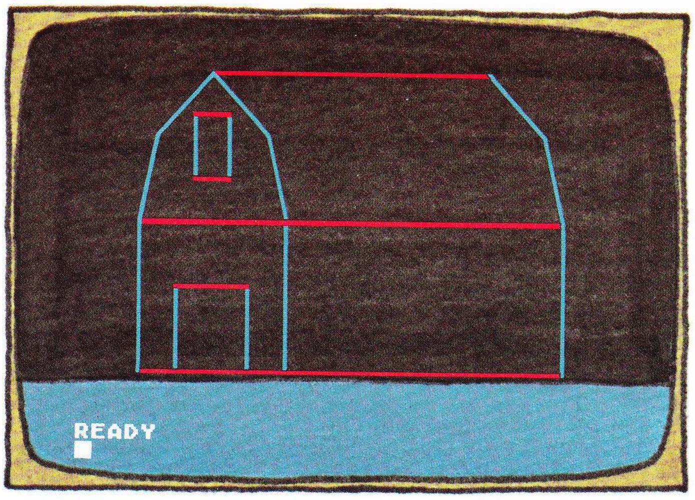






The command GR:DRAWTO X,Y tells the turtle to put the pen down and draw a straight line to the point with coordinates X and Y. The line is drawn in the pen's current color, and the turtle ends up at point X,Y. For example, the program in Figure 4-11 uses DRAWTO to make the outline of a barn.

```
10 R:BARN
20 GR:CLEAR
30 GR:PEN RED
40 GR:GOTO -30,40
50 GR:DRAWTO 45,40
60 GR:DRAWTO 60,25
70 GR:DRAWTO 65,5
80 GR:DRAWTO 65,5
90 GR:DRAWTO 65,-30
100 GR:DRAWTO -10,-30
110 GR:DRAWTO -10,5
120 GR:DRAWTO -15,25
130 GR:DRAWTO -30,40
140 GR:DRAW TO -45,25
150 GR:DRAWTO -50,5
160 GR:DRAWTO -50,-30
170 GR:DRAWTO -10,-30
180 GR:GOTO 65,5
190 GR:DRAWTO -50,5
200 GR:GOTO -35,30
210 GR:DRAWTO -25,30
220 GR:DRAWTO -25,15
230 GR:DRAWTO -35,15
240 GR:DRAWTO -35,30
250 GR:GOTO -40,-30
260 GR:DRAWTO -40,-10
270 GR:DRAWTO -20,-10
280 GR:DRAWTO -20,-30
290 E:

READY
■
```

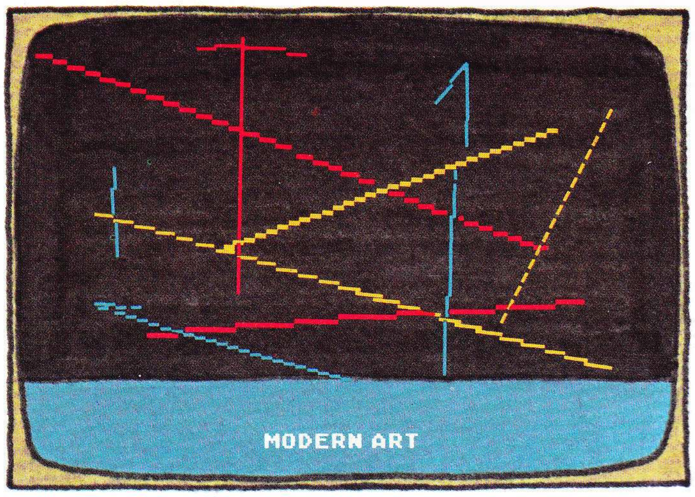


```
READY
■
```

Figure 4-11

The X and Y coordinates in the GOTO command may either be numbers, numeric variables, or numeric expressions. The "Modern Art 1" program in Figure 4-12 uses numeric variables #X and #Y as coordinates. These variables are given random values so the lines are drawn at random. The colors of the lines are also chosen at random.

```
10 R:MODERN ART I
20 *BEGIN
30 GR:CLEAR
40 T:
50 T:           MODERN ART
60 C:#L=?\200+7
70 C:#S=0
80 *MORE
90 C:#X=?\158-79
100 C:#Y=?\94-47
110 C:#C=?\4
120 GR(#C=0) :PEN UP
130 GR(#C=1) :PEN RED
140 GR(#C=2) :PEN BLUE
150 GR(#C=3) :PEN YELLOW
160 GR:DRAWTO #X,#Y
170 C:#S=#S+1
180 J(#S<#L) :*MORE
190 PA:1000
200 J:*BEGIN

READY
■
```



```
MODERN ART
```

Figure 4-12

GR:FILLTO

Fill to the Point X,Y

The command GR:FILLTO X,Y tells the turtle to draw a line to the point with coordinates X and Y. As he moves, the black background between the turtle and any figure or line directly to his right is filled with the current pen color. Figure 4-13 shows how FILLTO is used to fill parts of a barn in red.

```
10  R:RED BARN
20  GR:CLEAR
30  GR:PEN RED
40  GR:GOTO -30,40
50  GR:DRAWTO 45,40
60  GR:DRAWTO 60,25
70  GR:DRAWTO 65,5
80  GR:DRAWTO 65,5
90  GR:DRAWTO 65,-30
100  GR:DRAWTO -10,-30
110  GR:FILLTO -10,5
120  GR:FILLTO -15,25
130  GR:FILLTO -30,40
140  GR:DRAWTO -45,25
150  GR:DRAWTO -50,5
160  GR:DRAWTO -50,-30
170  GR:DRAWTO -10,-30
180  GR:GOTO 65,5
190  GR:PEN ERASE
200  GR:DRAWTO -10,5
210  GR:PEN RED
220  GR:DRAWTO -50,5
230  GR:GOTO -35,30
240  GR:DRAWTO -25,30
250  GR:DRAWTO -25,15
260  GR:DRAWTO -35,15
270  GR:FILLTO -35,30
280  GR:GOTO -20,-30
290  GR:DRAWTO -20,-10
300  GR:DRAWTO -40.-10
310  GR:FILLTO -40,-30
320  E:

READY
■
```

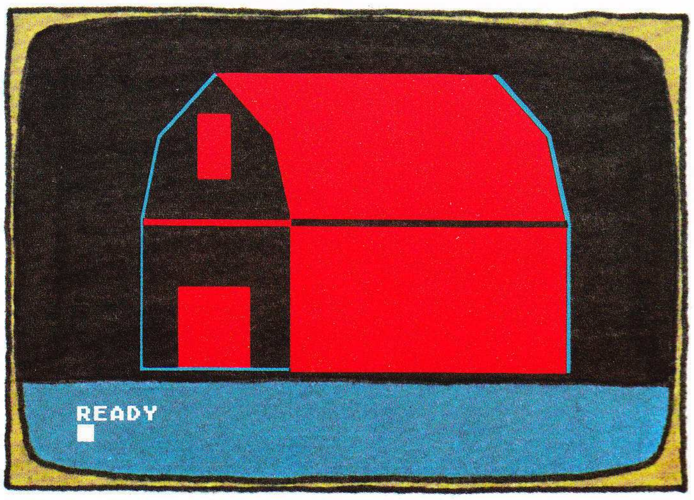


Figure 4-13

If there is nothing to block the color filling in before it reaches the right edge of the screen, the filling "wraps around," and color continues to fill in from the left edge until a barrier is reached. When there is nothing blocking the fill anywhere in its path, the line being filled from becomes its own barrier. This is shown in Figure 4-14.

```
10 R:FILLING UP SPACE
20 GR:GOTO 30,45;TURNTO 180;PEN
YELLOW
30 U:*PARTBOX
40 GR:GOTO -40,15;TURNTO 90;PEN
YELLOW
50 U:*PARTBOX
60 GR:GOTO -50,-20;TURNTO 30;PEN
YELLOW
70 U:*TRIANGLE
80 GR:GOTO 15,0;TURNTO 90;PEN
YELLOW
90 U:*BOX
100 GR:GOTO 50,5;TURNTO -30;PEN
YELLOW
120 U:*TRIANGLE
200 GR:GOTO -50,-30
210 GR:PEN RED
220 GR:FILLTO 50,30
230 GR:PEN YELLOW;GOTO 51,31
299 E:
400 *BOX
410 GR:4(DRAW 10;TURN 90)
420 E:
500 *PARTBOX
510 GR:3(DRAW 20;TURN 90)
520 E:
600 *TRIANGLE
610 GR:3(DRAW 15;TURN 120)
620 E:

READY
■
```

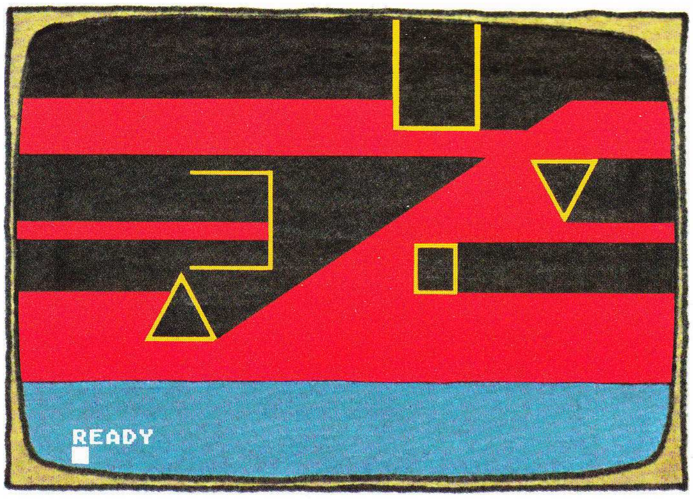
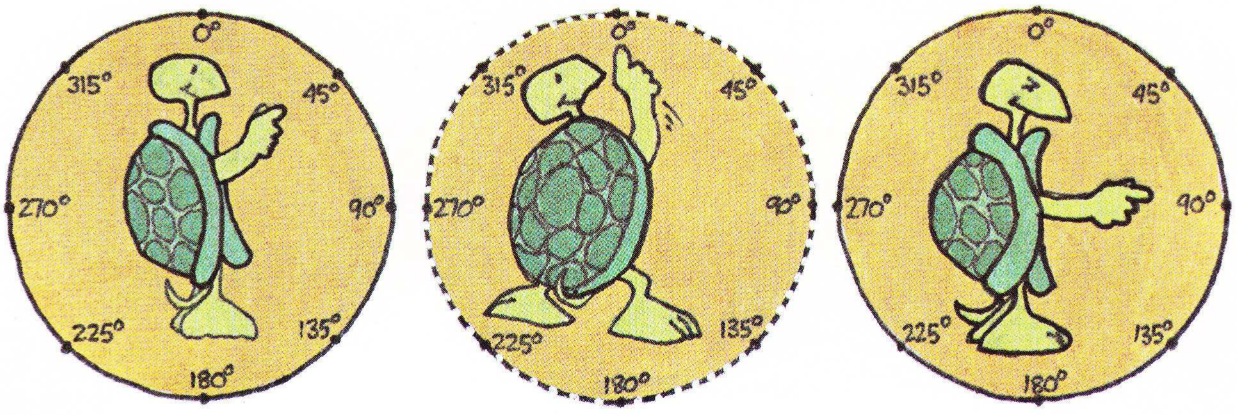


Figure 4-14

FILLTO, like GOTO and DRAWTO, may use either numbers, numeric variables, or numeric expressions as coordinates.

GR:TURNTO

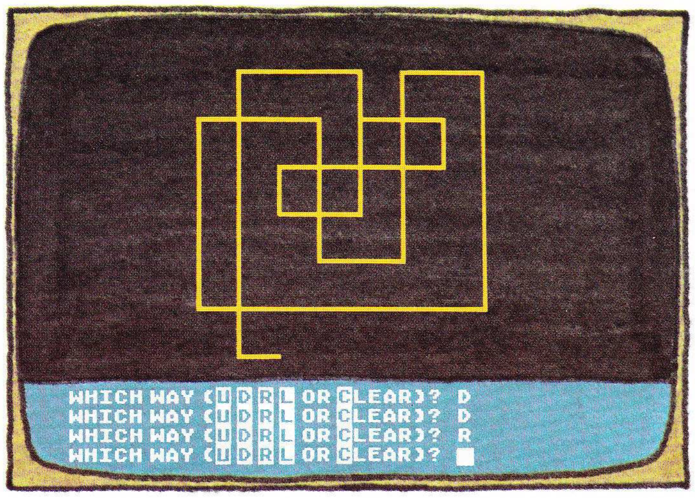
Turn to N Degrees



The command GR:TURNTO N tells the turtle to turn in place until he is heading toward the angle N degrees. The top of the screen is 0 degree. Positive angles (+) are measured clockwise from 0 degree; negative angles (-) are measured counterclockwise.

Since the turtle is invisible on the screen, you cannot see him turn, but you can see the direction of his new heading by executing a GO, DRAW, or FILL command after the TURNTO. Figure 4-15 shows a program that uses TURNTO to let you make a design on the screen with right angles and 10-unit lines.

```
10  R:SKETCHING PROGRAM
20  T:THIS PROGRAM LETS YOU DRAW A
30  T:DESIGN ON THE GRAPHICS
SCREEN
40  T:USING THE COMMANDS:
50  T:
60  T:UP
70  T:DOWN
80  T:RIGHT
90  T:LEFT
100  T:CLEAR
110  T:
120  T:PRESS RETURN TO BEGIN . . .
130  A:
140  *BEGIN
150  GR:CLEAR
160  T:STARTING X LOCATION? \
170  A:#X
180  T:STARTING Y LOCATION? \
190  A:#Y
200  GR:GOTO #X,#Y
210  *COMMAND
220  T:WHICH WAY (U D R L or CLEAR)? \
230  A:
240  M:  U, D, R, L, C
250  JM:*UP,*DOWN,*RIGHT,*LEFT,
*BEGIN
260  *UP GR:TURNTO 0
270  J:*LINE
280  *DOWN GR:TURNTO 180
290  J:*LINE
300  *RIGHT GR:TURNTO 90
310  J:*LINE
320  *LEFT GR:TURNTO -90
330  *LINE GR:DRAW 10
340  J:*COMMAND

READY
■
```

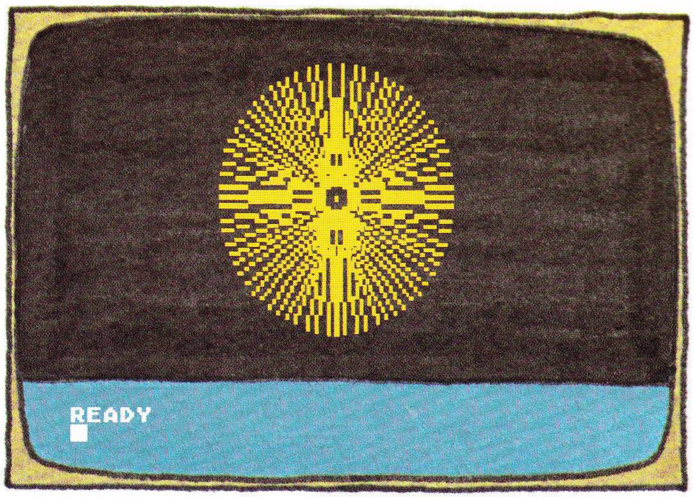


```
WHICH WAY (U D R L OR CLEAR)? D
WHICH WAY (U D R L OR CLEAR)? D
WHICH WAY (U D R L OR CLEAR)? R
WHICH WAY (U D R L OR CLEAR)? ■
```

Figure 4-15

The directional angle, N, in the GR:TURNTO N command may be either a number, a numeric variable, or a numeric expression. The program in Figure 4-16 uses TURNTO with the numeric variable, #T, to create a starburst pattern with lines coming out from the origin every 5 degrees.

```
10 R:STARBURST
20 C:#T=0
25 GR:CLEAR
30 *LOOP
40 GR:GOTO 0,0
50 GR:TURNTO #T
60 GR:GO 3;DRAW 25
70 C:#T=#T+5
80 J(#T<360):*LOOP
90 E:

READY
■
```

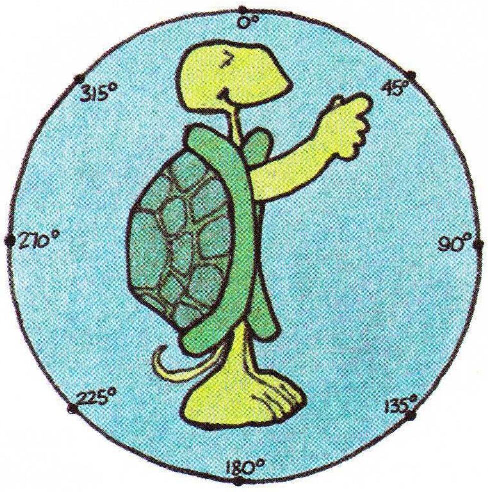


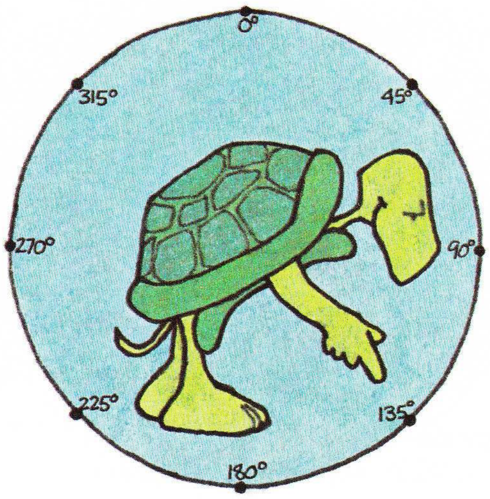
```
READY
■
```

Figure 4-16

GR:TURN

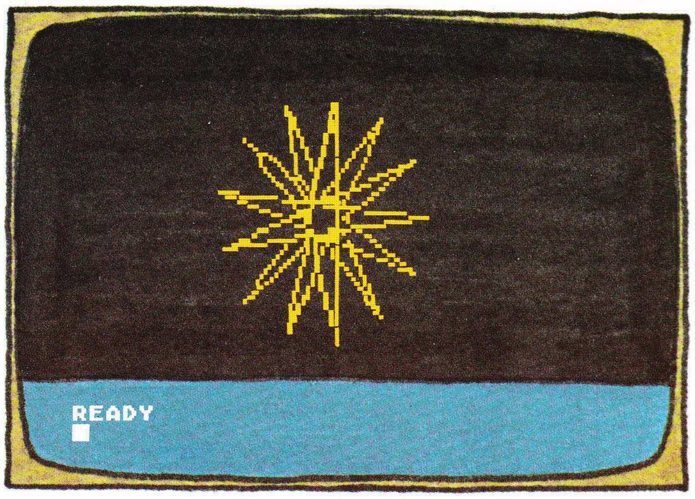
Turn N Degrees





The command GR: TURN N tells the turtle to turn N degrees clockwise (+N) or counterclockwise (-N) from the direction he is currently facing. Since the turtle is invisible on the screen, you cannot see him turn, but you can see the direction of his new heading by executing a GO, DRAW, or FILL command after the TURN. Figure 4-17 shows a program that makes a 13-pointed star with a 166-degree turn at each point.

```
10 R:13-POINTED STAR
20 C:#C=0
30 GR:CLEAR
40 GR:GOTO -10,-5
50 GR:TURNTO 14
60 *LOOP
70 GR:DRAW 50
80 GR:TURN 166
90 C:#C=#C+1
100 J(#C<14) :*LOOP
110 E:

READY
■
```



```
READY
■
```

Figure 4-17

The directional angle N, in the TURN (N degrees) command can be either a number, a numeric variable, or a numeric expression. The program in Figure 4-18 draws first a triangle, then a square, then a pentagon, and so on, all on top of each other. The amount the turtle turns in line 90 is 360 degrees divided by the number of sides in the polygon he is currently drawing.

```
10 R:EUCLID
20 C:#S=3
30 GR:CLEAR
40 *BEGIN
50 C:#C=0
60 GR:GOTO 10,-30
70 GR:TURNTO -90
80 *LOOP
90 GR:DRAW 20;TURN 360/#S
100 C:#C=#C+1
110 J(#C<#S):*LOOP
120 C:#S=#S+1
130 J(#S<21):*BEGIN
140 E:

READY
■
```

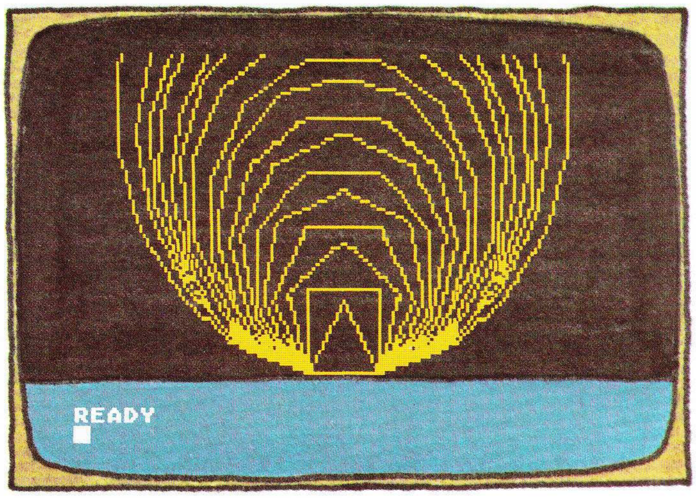
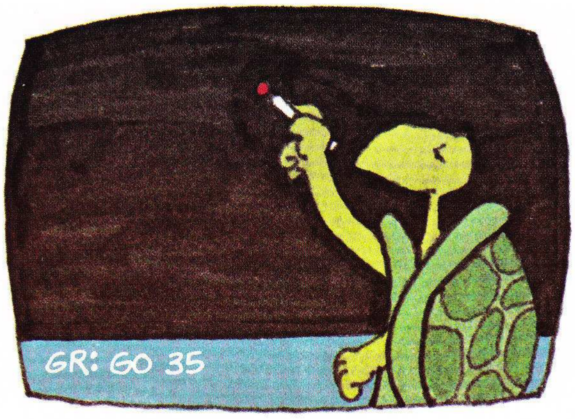
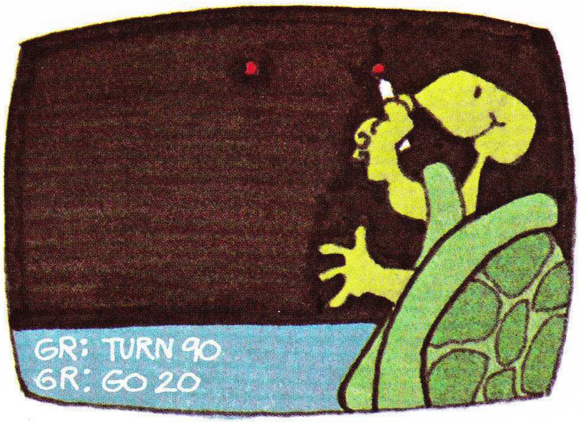


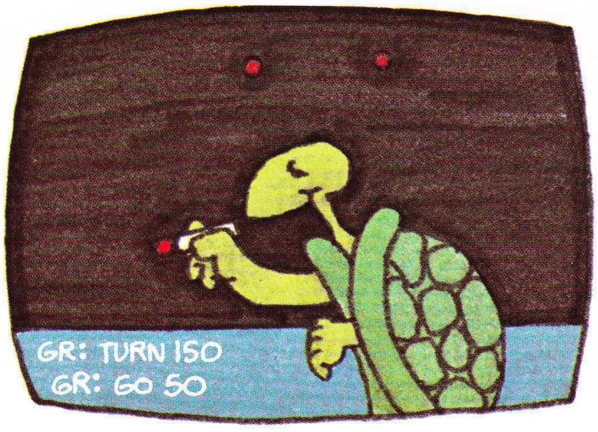
Figure 4-18

GR:GO

Go N Units

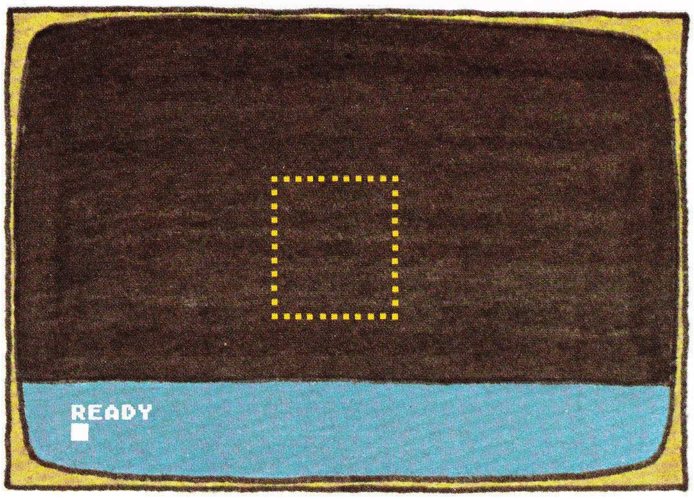






The GR:GO N command tells the turtle to pick up the pen, move N units in the direction he is facing, and put the pen down there. The turtle leaves a dot on the screen at the location where he ends up. The program in Figure 4-19 uses the GO command to draw a box with dotted lines.

```
10 R:DOTTED BOX
20 GR:CLEAR
30 GR:4(10(GO 3) ;TURN 90)
40 E:

READY
■
```

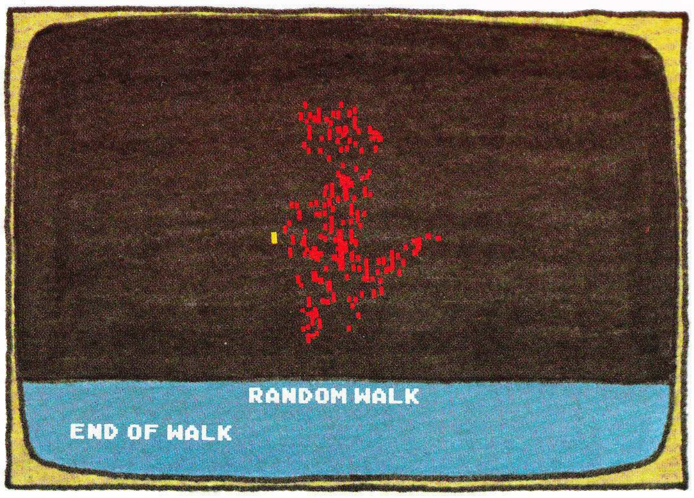


```
READY
■
```

Figure 4-19

The number of units, N, in the GR:GO N command can be either a number, numeric variable, or a numeric expression. The program in Figure 4-20 tells the turtle to take a "random walk." That is, he starts at the origin, turns a random number of degrees, goes a random number of units, turns at random again, moves at random again, and so on. At the end of this walk, all the places he stopped and turned are marked by dots on the screen.

```
10 R:RANDOM WALK
20 *BEGIN
30 C:#C=0
40 GR:CLEAR
50 T:         RANDOM WALK
60 GR:PEN YELLOW;GOTO 0,0;PEN RED
70 *LOOP
80 GR:GO ?\5
90 GR:TURN ?\360
100 C:#C=#C+1
110 J(#C<500) :*LOOP
120 T:
130 T:END OF WALK
140 PA:900
150 J:*BEGIN

READY
■
```

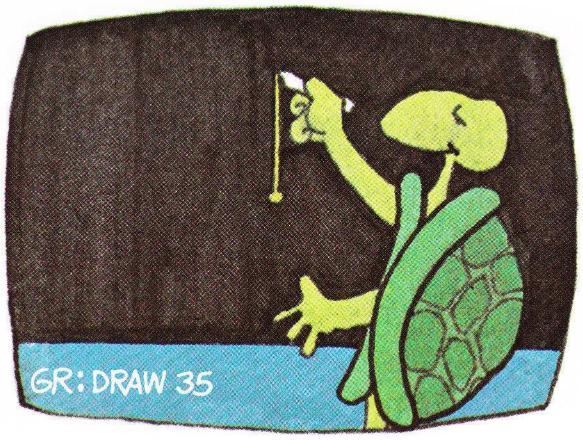


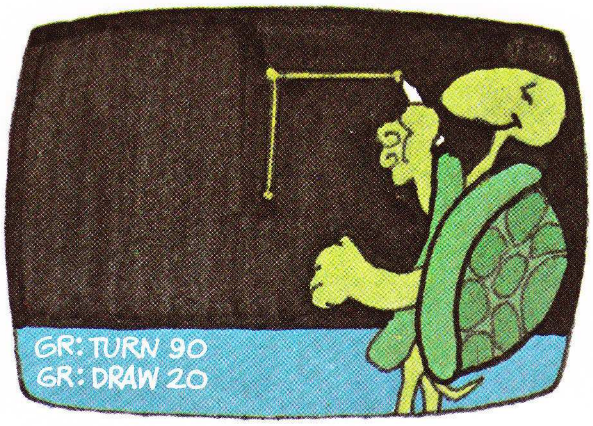
```
           RANDOM WALK
END OF WALK
```

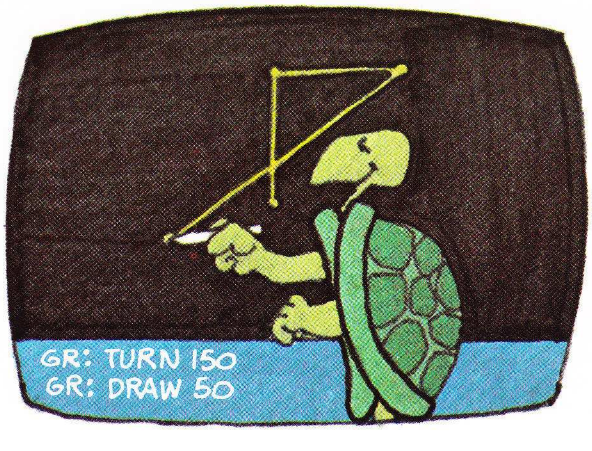
Figure 4-20

GR:DRAW

Draw N Units

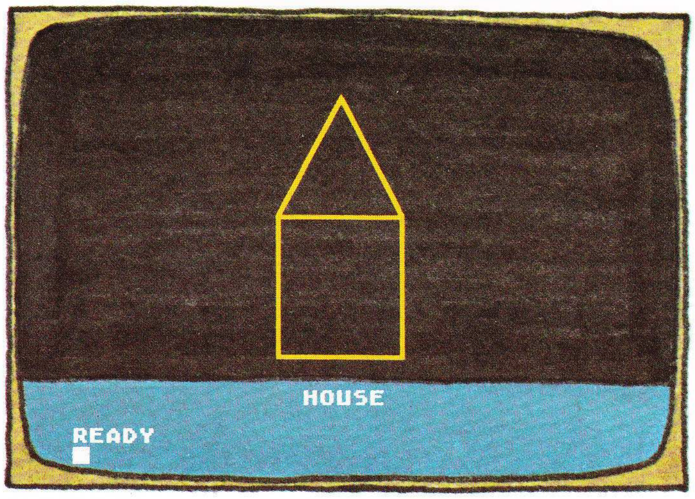






The command GR:DRAW N tells the turtle to draw a line, N units long, moving in the direction he is currently facing. The line is drawn in the current pen color. The program in Figure 4-21 uses the DRAW and TURN commands to make a simple house.

```
10 R:SIMPLE HOUSE
20 GR:CLEAR
30 T:             HOUSE
40 GR:GOTO -15,10
50 GR:TURNTO 90
60 GR:DRAW 30;TURN 90
70 GR:DRAW 30;TURN 90
80 GR:DRAW 30;TURN 90
90 GR:DRAW 30;TURN 30
100 GR:DRAW 30;TURN 120
110 GR:DRAW 30;TURN 120
120 E:

READY
■
```

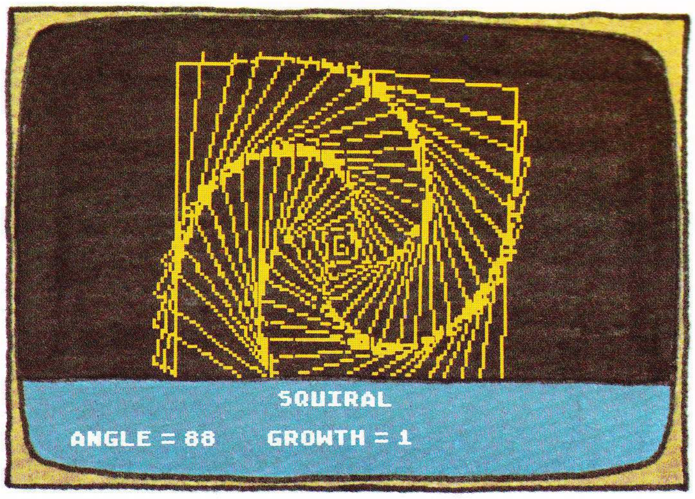


```
            HOUSE
READY
■
```

Figure 4-21

The number of units, N, in the GR:DRAW N command can be either a number, a numeric variable, or a numeric expression. Figure 4-22 shows a program that draws spirals at any angle you choose. Each time the turtle turns, the side of the spiral grows longer by the amount you type into the computer.

```
10 R:SQUIRAL
20 GR:CLEAR
30 *BEGIN
40 T:            SQUIRAL
50 T:
60 T:TURNING ANGLE (1-179)? \
70 A:#A
80 T:SPEED OF GROWTH (1-10)? \
90 A:#I
100 GR:CLEAR;GOTO 0,0;TURNTO 0
110 T:                     SQUIRAL
120 T:
125 T:ANGLE = #A         GROWTH =#I
130 C:#S=0
140 *MORE
150 GR:DRAW #S;TURN #A
160 C:#S=#S+#I
170 J(#S<200):*MORE
180 T:AGAIN? \
190 A:
200 M: Y,SURE,OK,ALRIGHT
210 JY:*BEGIN
220 GR:QUIT
230 E:

READY
■
```



```
           SQUIRAL
ANGLE = 88     GROWTH = 1
```

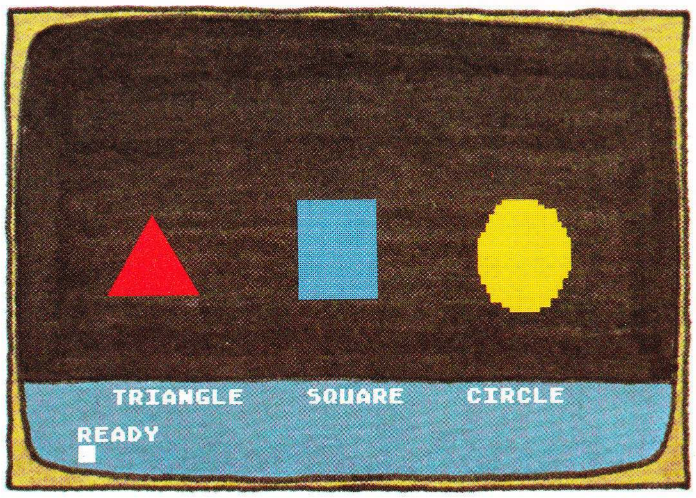
Figure 4-22

GR:FILL

Fill N Units

The GR:FILL N command tells the turtle to draw a line, N units long, moving in the direction he is currently facing. As he moves, the black background between the turtle and any figure or line directly to his right is filled in with the current pen color. Figure 4-23 shows how FILL can be used to draw a solid triangle, square, or circle.

```
10 R:SOLIDS
20 GR:CLEAR
30 T:   TRIANGLE  SQUARE  CIRCLE
40 GR:GOTO -50,8;TURNTO 150;PEN
RED
50 U:*TRIANGLE
60 GR:GOTO -15,10;TURNTO 90;PEN
BLUE
70 U:*SQUARE
80 GR:GOTO 40,10;TURNTO 90;PEN
YELLOW
90 U:*CIRCLE
100 E:
200 *TRIANGLE
210 GR:2(DRAW 20;TURN 120)
220 GR:FILL 19
230 E:
300 *SQUARE
310 GR:3(DRAW 20;TURN 90)
320 GR:FILL 20
330 E:
400 *CIRCLE
410 GR:18(DRAW 2;TURN 10)
420 GR:18(FILL 2;TURN 10)

READY
■
```



```
     TRIANGLE     SQUARE     CIRCLE
READY
■
```

Figure 4-23

Note: To color in a polygon (such as a triangle, square, or circle) using the FILL command, you must first DRAW the right half of the polygon and then FILL the left half. Sometimes, the filled side must be shorter than the drawn side so that the color will be completely contained within the figure. For example, in Figure 4-23, the two *drawn* sides of the triangle are 20 units long, whereas the *filled* side is only 19 units long.

If there is nothing to block the color filling in before it reaches the right edge of the screen, the filling "wraps around," and color continues to fill in from the left edge until a barrier is reached. When there is nothing blocking the fill anywhere in its path, the line from which the filling takes place becomes its own barrier. The "Sunset" program in Figure 4-24 shows how FILL can be used to fill large areas of the screen with color.

```
10 R:SUNSET
15 GR:CLEAR
20 U:*SUN
30 U:*REDCLOUDS
40 U:*SKY
45 U:*SUNBATH
50 E:
100 *SUNBATH
110 GR:GOTO 0,-13,TURNTO 180;PEN
YELLOW
120 GR:FILL 26
130 E:
200 *SKY
210 GR:GOTO 0,10;TURNTO 0;PEN
BLUE
220 GR:FILL 37
230 E:
300 *SUN
310 GR:GOTO 0,10;TURNTO 90
320 GR:36(DRAW 1;TURN 5)
330 GR:36(FILL 1;TURN 5)
340 E:
400 *REDCLOUDS
410 GR:GOTO 0,10;TURNTO 90;PEN
RED
420 GR:36(FILL 1;TURN 5)
430 E:

READY
■
```

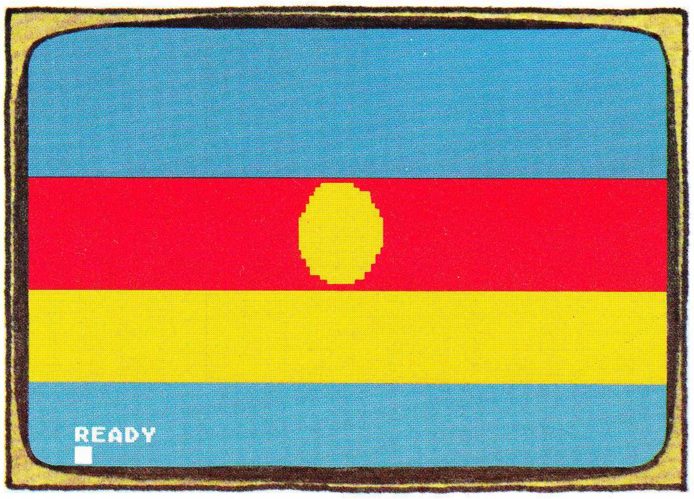
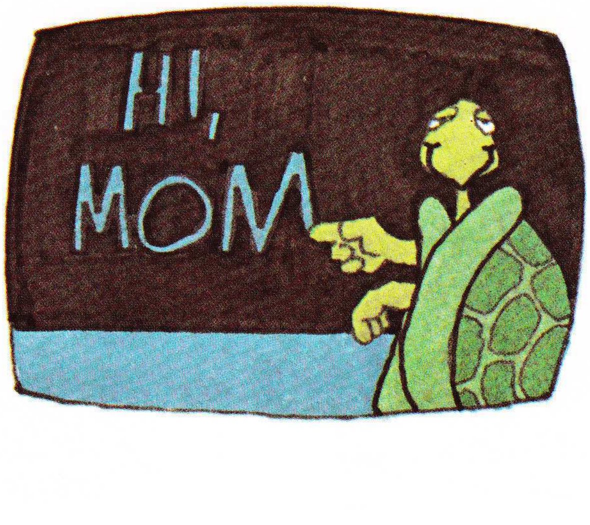


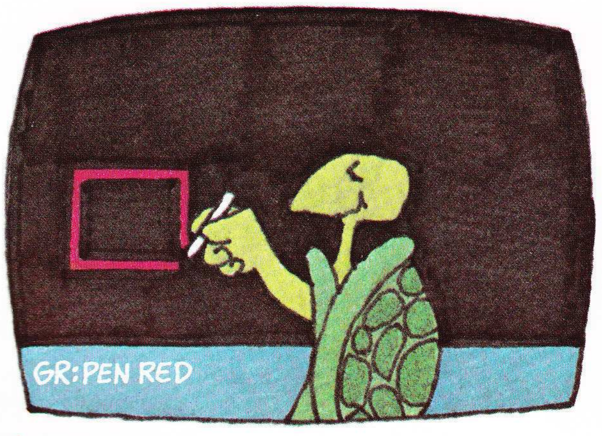
Figure 4-24

The number of units, N, in the GR:FILL N command can be either a number, a numeric variable, or a numeric expression.

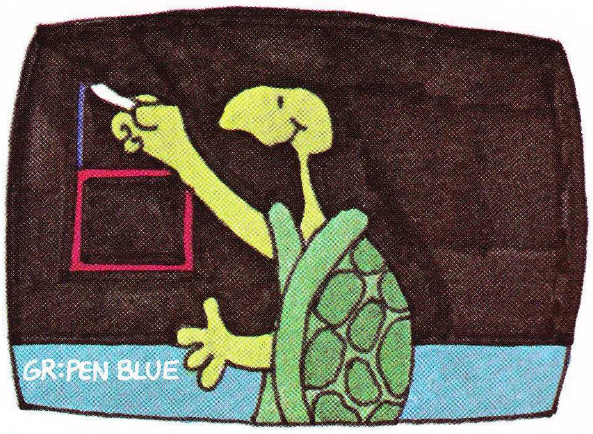


GR:PEN

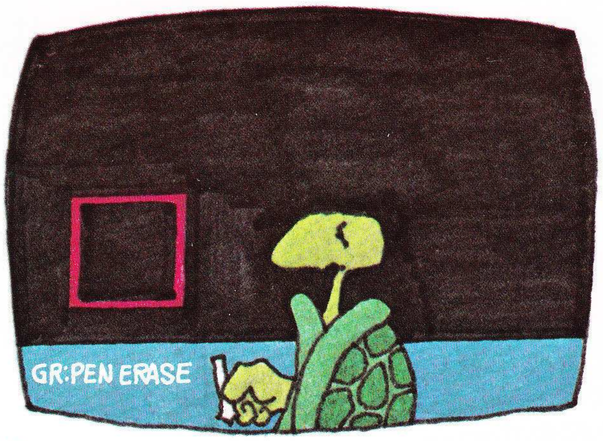
Choose a Pen Color



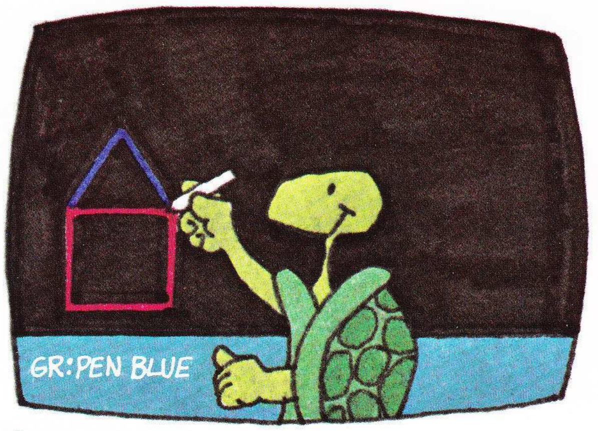
1.



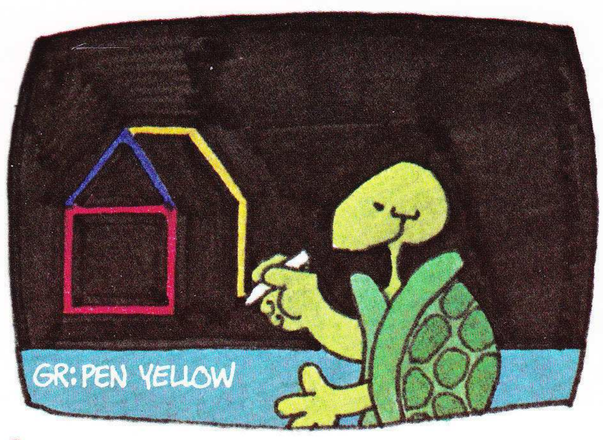
2.



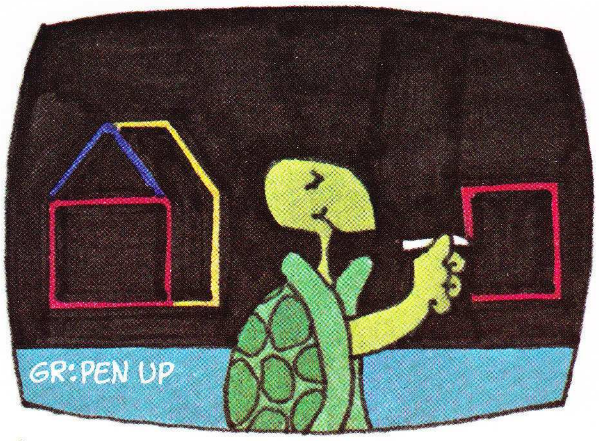
3.



4.



5.



6.

The GR:PEN COLOR command tells the turtle to choose a pen color — yellow, red, or blue. For example, the program shown in Figure 4-25 draws polka dots in yellow, red, and blue in random positions on the screen.

```
10 R:POLKA DOTS
20 *BEGIN
30 C:#N=0
40 GR:CLEAR
50 T:          POLKA DOTS
60 *POLKA
70 C:#X=?\150-75
80 C:#Y=?\62-31
90 C:#C=?\3+1
100 GR(#C=1):PEN YELLOW
110 GR(#C=2):PEN RED
120 GR(#C=3):PEN BLUE
130 GR:GOTO #X,#Y
140 U:*DOT
150 C:#N=#N+1
160 J(#N<50):*POLKA
170 PA:120
180 J:*BEGIN
190 E:
200 *DOT
210 GR:TURNTO 90
220 GR:10(DRAW 1;TURN 18)
230 GR:10(FILL 1;TURN 18)
240 E:

READY
■
```

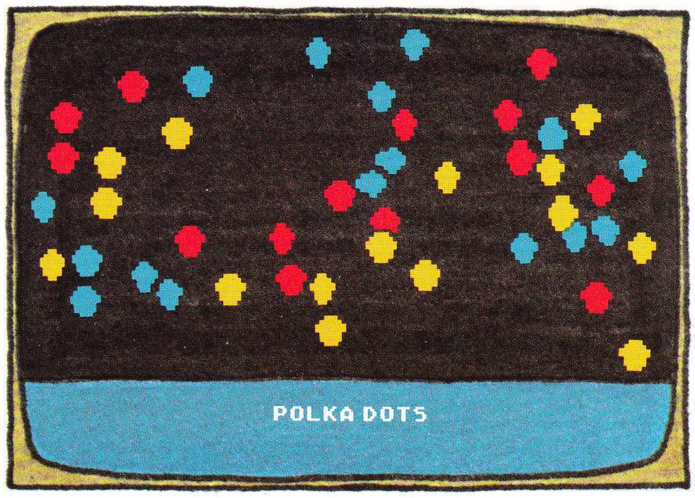


Figure 4-25

GR:PEN ERASE

Set Pen Color to Erase

You can also tell the turtle to draw in the black background color by using the command PEN ERASE. This command lets you "erase" anything you have already drawn by redrawing it in the black background color. Figure 4-26 shows how the PEN ERASE command is used repeatedly to draw a star, erase it, and redraw it in a different place. This gives the effect that the star is moving.

```
10 R:SHOOTING STAR
20 *BEGIN
30 C:#C=0
40 GR:CLEAR
50 T:          SHOOTING STARS
60 GR:GOTO -75,40
70 GR:TURNTO 120
80 *LOOP
90 GR:PEN YELLOW
100 U:*STAR
110 GR:PEN ERASE
120 U:*STAR
130 GR:GO 5
140 C:#C=#C+1
150 J(#C<32):*LOOP
160 PA:120
170 J:*BEGIN
180 *STAR
190 GR:5(DRAW 10;TURN 144)
200 E:

READY
```

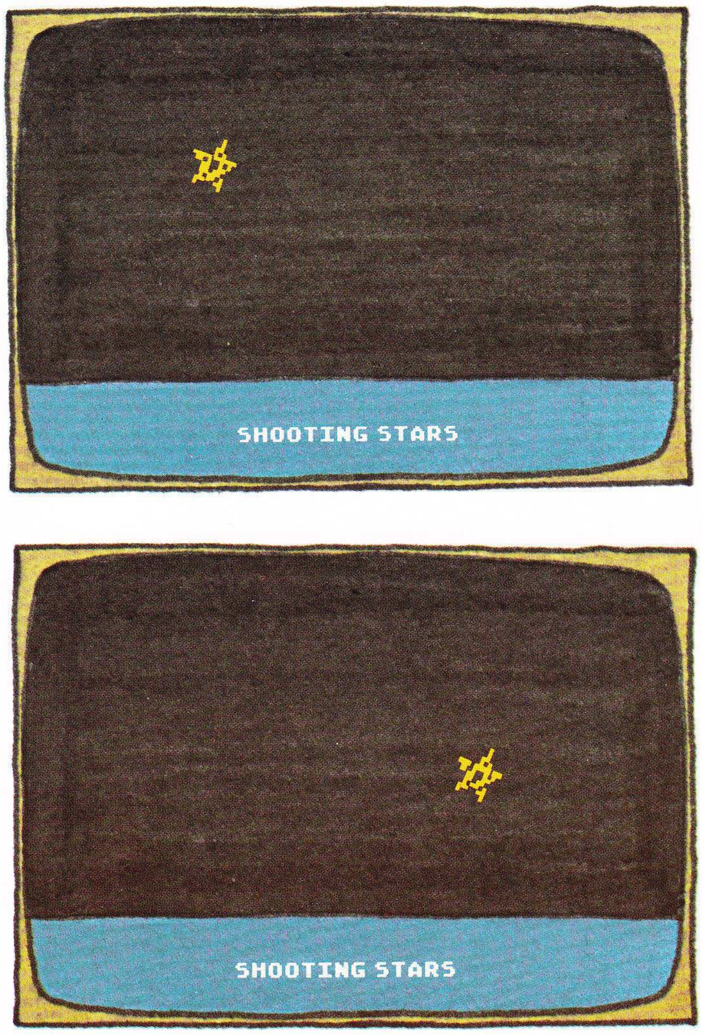
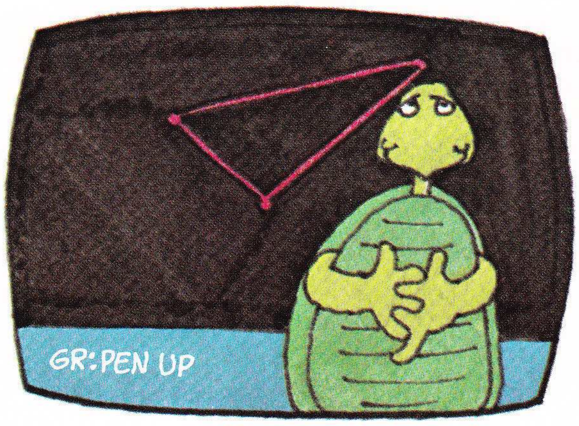


Figure 4-26

GR:PEN UP

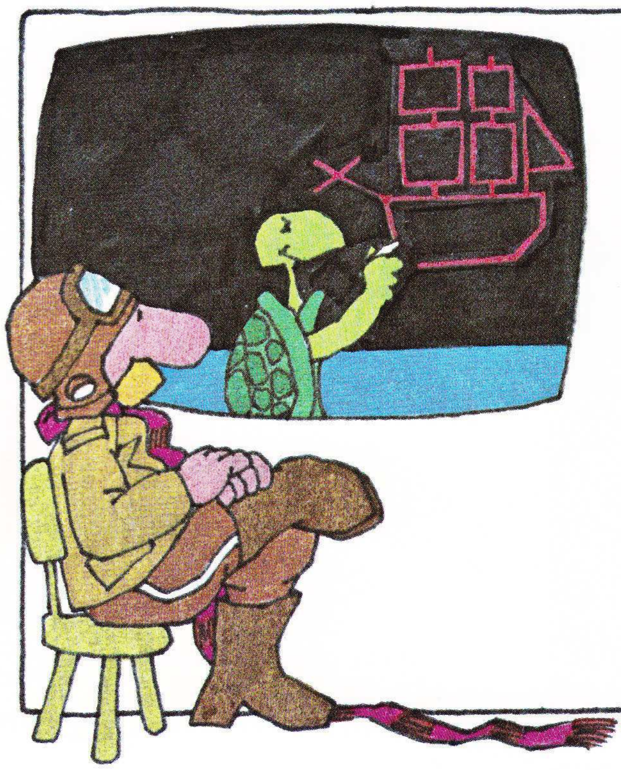
Lift the Pen

The PEN UP command tells the turtle to pick up the pen off the screen. While the pen is up, all movement by the turtle will be invisible. To put the pen back "down," you choose a pen color.

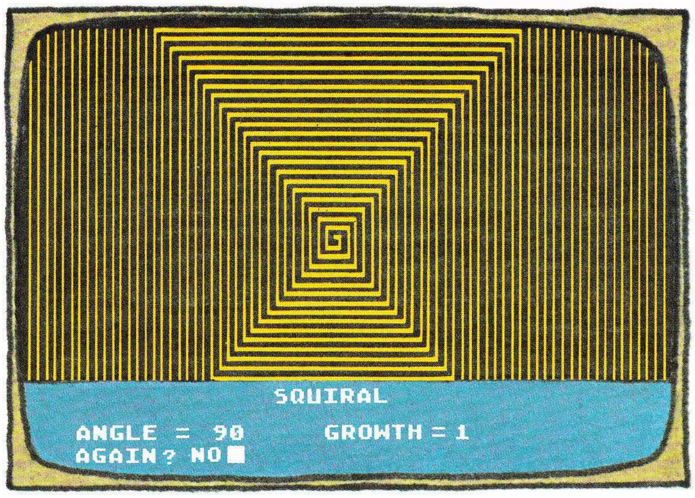


GR:QUIT

Quit the Graphics Mode

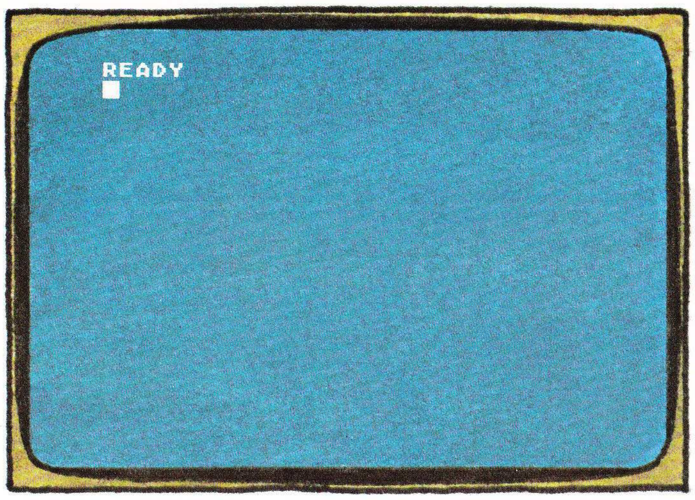


The command GR:QUIT lets you leave the Graphics mode and return to Text mode. Figure 4-27 shows you the screen *before* the GR:QUIT command, and Figure 4-28 shows you the screen *after* the GR:QUIT command.



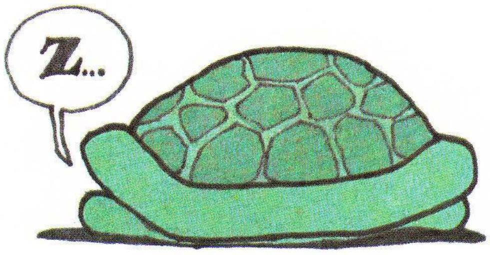
```
SQUIRAL
ANGLE = 90       GROWTH = 1
AGAIN? NO
```

Figure 4-27



READY

Figure 4-28



5 SOUND AND PAUSE

SO:
Sound

The SO: command lets you create musical tones on your computer. You can program up to four tones to play at the same time. Each separate tone is then called a *voice*. When a group of people get together to sing, they divide into several *voices*: bass, baritone, alto, and soprano. When you program SO: commands, you can program several voices to play together. When different voices play a sound at the same time, it is called a *chord*.

The tones you can play in PILOT range from 0 to 31. If you enter the command **SO:13**, you will hear a single tone equal to middle C on a piano. The command **SO:1** produces a tone equal to the note C, one octave below middle C. The command **SO:31** produces the note F sharp, one octave above middle C. **SO:0** turns a tone OFF.

Figure 5-1 shows a piano keyboard with the key numbers corresponding to the PILOT SOUND commands.

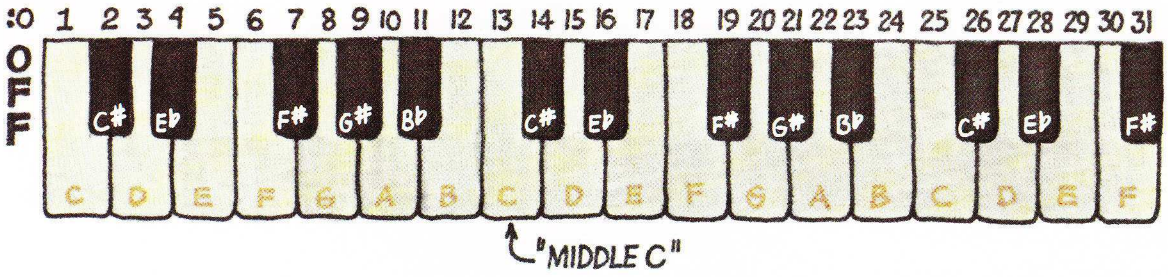
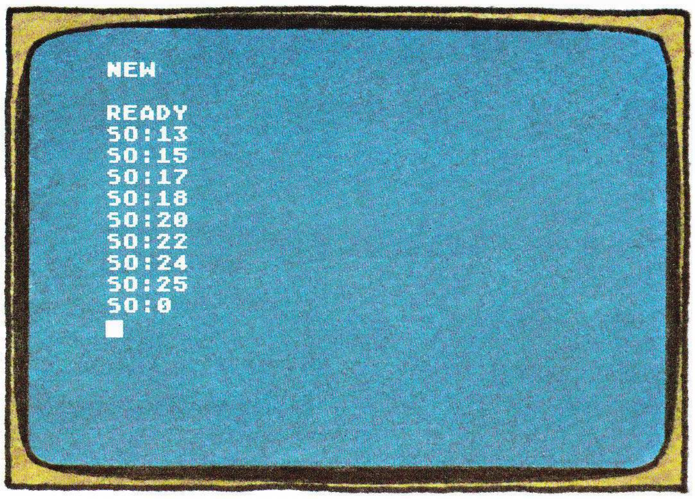


Figure 5-1

Typing the commands in Figure 5-2 one at a time will produce a scale.



```
NEW

READY
SO:13
SO:15
SO:17
SO:18
SO:20
SO:22
SO:24
SO:25
SO:0
```

Figure 5-2

A scale is a series of notes played one after another. Think of a ladder on which you take one step at a time to go up or down. A scale is similar, only with musical notes being played in a specific order. This particular scale covers the octave between middle C and C above middle C. An octave is a range of notes from one note to another note of the same name (higher or lower).

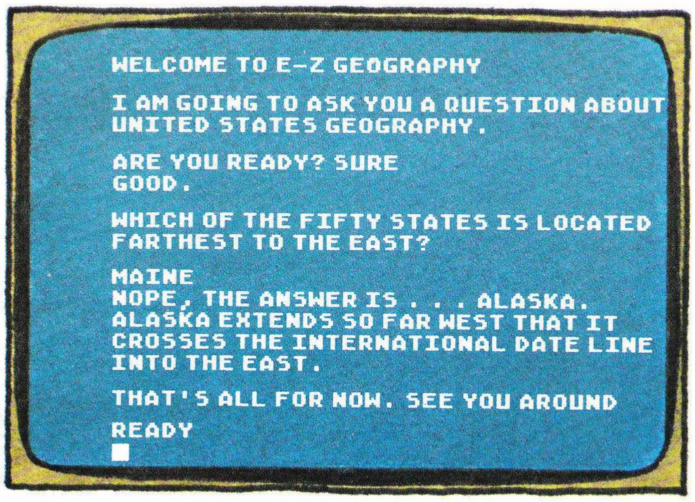
The program in Figure 5-3 repeatedly moves up and down the scale shown on page 86. When this program is RUN, the computer moves through the SOUND commands so quickly that all the tones blur together, and this scale sounds like a siren. The PA: commands in this program increase the length of time the tones are sounded. Without them, this scale sounds like rumbling static. The PA: command is explained later in this section.

```
10 R:SIREN
20 *BEGIN
30 SO:13
35 PA:1
40 SO:15
45 PA:1
50 SO:17
55 PA:1
60 SO:18
65 PA:1
70 SO:20
75 PA:1
80 SO:22
85 PA:1
90 SO:24
95 PA:1
100 SO:25
105 PA:1
110 SO:24
115 PA:1
120 SO:22
125 PA:1
130 SO:20
135 PA:1
140 SO:18
145 PA:1
150 SO:17
155 PA:1
160 SO:15
165 PA:1
170 SO:13
180 SO:0
190 J:*BEGIN

READY
■
```

Figure 5-3

The SO: command lets you create sound effects and tunes in your PILOT programs. Figure 5-4 shows how you might use SOUND to reward correct answers in a simple geography quiz program.

```
10  R:E-Z GEOGRAPHY
20  T:WELCOME TO E-Z GEOGRAPHY
30  T:
40  T:I AM GOING TO ASK YOU A
QUESTION
50  T:ABOUT UNITED STATES
GEOGRAPHY
60  T:
70  T:ARE YOU READY? \
80  A:
90  T:GOOD
100  T:
110  PA:60
120  T:WHICH OF THE FIFTY STATES IS
130  T:LOCATED FARTHEST TO THE
EAST?
140  T:
150  A:$STATE
160  M: ALASKA
170  TY:EXTRAORDINARY! YOU GOT IT!
180  UY:*DAHDAH
190  JY:*THATSALL
200  T:NOPE, THE ANSWER IS . . . \
210  PA:30
220  T:ALASKA
230  T:ALASKA EXTENDS SO FAR WEST
THAT
240  T:IT CROSSES THE
INTERNATIONAL
50  T:DATE LINE INTO THE EAST.
260  T:
270  *THATSALL T:
280  T:THAT'S ALL FOR NOW. SEE YOU
AROUND.
290  E:
300  *DAHDAH
310  C:#C=0
320  *LOOP
330  SO:13,17,20,25
340  PA:15
350  SO:0,0,0,0
360  C:#C=#C+1
370  J(#C<5):*LOOP
380  E:

READY
■
```



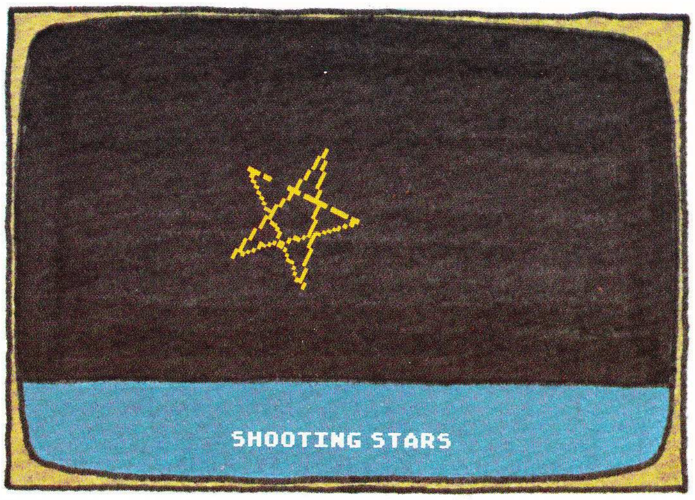
```
WELCOME TO E-Z GEOGRAPHY

I AM GOING TO ASK YOU A QUESTION ABOUT
UNITED STATES GEOGRAPHY.

ARE YOU READY? SURE
GOOD.

WHICH OF THE FIFTY STATES IS LOCATED
FARTHEST TO THE EAST?

MAINE
NOPE, THE ANSWER IS . . . ALASKA.
ALASKA EXTENDS SO FAR WEST THAT IT
CROSSES THE INTERNATIONAL DATE LINE
INTO THE EAST.

THAT'S ALL FOR NOW. SEE YOU AROUND

READY
■
```

Figure 5-4

You may use either numbers or numeric variables to tell the computer which tones to play in a SO: command. Values greater than 31 are reduced to numbers between 0 and 31.

Figure 5-5 shows how the SO: command can be used with a numeric variable to make sound effects for the shooting star program of Figure 4-26.

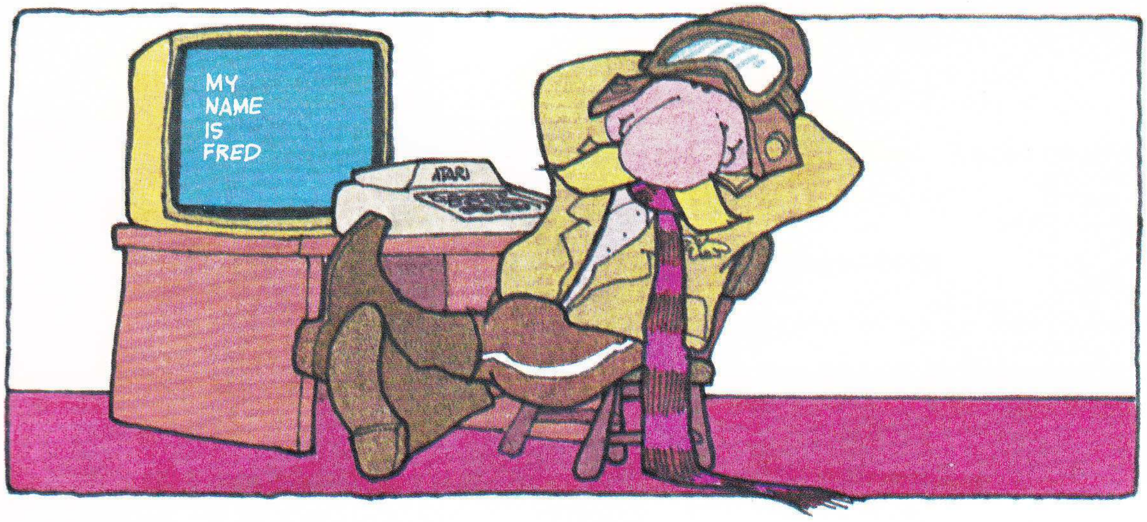
```
1   GR:QUIT
10  R:SHOOTING STAR WITH SOUND
EFFECTS
20  *BEGIN
30  C:#C=0
40  GR:CLEAR
50  T:          SHOOTING STARS
60  GR:GOTO -75,40
70  GR:TURNTO 120
80  C:#S=?\20+1
90  *LOOP
100  C:#P=31-#C
110  SO:#P
120  GR:PEN YELLOW
130  U:*STAR
140  GR:PEN ERASE
150  U:*STAR
160  GR:GO 5
170  C:#C=#C+1
180  J(#C<32):*LOOP
190  PA:240
200  J:*BEGIN
210  *STAR
220  GR:5(DRAW #S; TURN 144)
230  E:

READY
■
```



```
SHOOTING STARS
```

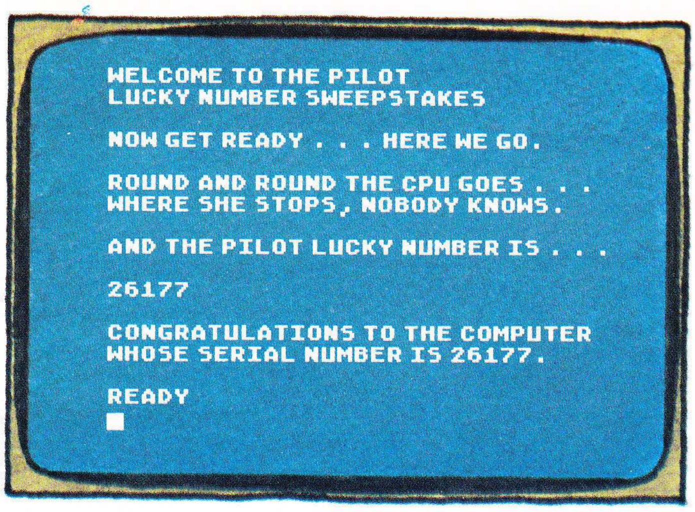
Figure 5-5

PA:
Pause N Units



Use the PA:N command when you want a program to PAUSE for any reason during the RUN of a program. For example, PA: may be used to freeze what is on the screen for a few moments before clearing the the screen. PA: also allows you to write text on the screen a little at a time or to pause for dramatic effect: "The correct answer is.... (pause)...An Embarrassed Zebra." The program in Figure 5-6 pauses for dramatic effect before announcing a lucky number in the PILOT Lucky Number Drawing.

```
10  R:LUCKY NUMBER
20  T:WELCOME TO THE PILOT
30  T:LUCKY NUMBER SWEEPSTAKES
40  T:
50  PA:90
60  T:NOW GET READY . . . \
70  PA:90
80  T:HERE WE GO.
90  T:
100  PA:90
110  T:ROUND AND ROUND THE CPU GOES
. . .
120  T:WHERE SHE STOPS, NOBODY
KNOWS.
130  T:
140  PA:90
150  C:#R=?\30000
160  T:AND THE PILOT LUCKY NUMBER
IS . . .
170  T:
180  PA:180
190  T:#R
200  T:
210  PA:180
220  T:CONGRATULATIONS TO THE
COMPUTER
230  T:WHOSE SERIAL NUMBER IS #R.
240  E:

READY
■
```



```
WELCOME TO THE PILOT
LUCKY NUMBER SWEEPSTAKES

NOW GET READY . . . HERE WE GO.

ROUND AND ROUND THE CPU GOES . . .
WHERE SHE STOPS, NOBODY KNOWS.

AND THE PILOT LUCKY NUMBER IS . . .

26177

CONGRATULATIONS TO THE COMPUTER
WHOSE SERIAL NUMBER IS 26177.

READY
■
```

Figure 5-6

The units in the PA:N command may either be a number or a numeric variable. Each unit is equal to 1/60th of a second. Thus **PA:60** will pause for 1 second; **PA:30** will pause for ½ second; **PA:300** will pause for 5 seconds, and so on.

The PA: command is often used in combination with the SOUND command to tell the computer how long to sound one or more voices. For example, the program in Figure 5-7 uses SO: and PA: to add a random three-chord polka to the Polka Dots program of Figure 4-25.

```
10  R:DOTS POLKA
20  *BEGIN
30  C:#N=0
40  GR:CLEAR
50  T:
60  T:               DOTS POLKA
70  *POLKA
80  C:#X=?\150-75
90  C:#Y=?\62-31
100  C:#C=?\3+1
110  GR(#C=1):PEN YELLOW
120  SO(#C=1):13,17,20
130  GR(#C=2):PEN RED
140  SO(#C=2):13,18,22
150  GR(#C=3):PEN BLUE
160  SO(#C=3):15,20,24
170  GR:GOTO #X,#Y
180  U:*DOT
190  U:*UMPAH
200  C:#N=#N+1
210  J(#N<50):*POLKA
220  U:*FINALE
230  J:*BEGIN
240  E:
300  *DOT
310  GR:TURNTO 90
320  GR:10(DRAW 1;TURN 18)
330  GR:10(FILL 1;TURN 18)
340  E:
400  *UMPAH
410  C:#R=#N\3
420  PA(#R=0):20
430  SO:0,0,0
440  PA(#R=0):5
450  E:
500  *FINALE
510  SO:13,17,20
520  PA:60
530  SO:13,18,22
540  PA;60
550  SO:15,20,24
560  PA:60
570  SO:13,17,20,25
580  PA:120
590  SO:0,0,0
600  PA:480
610  E:

READY
■
```

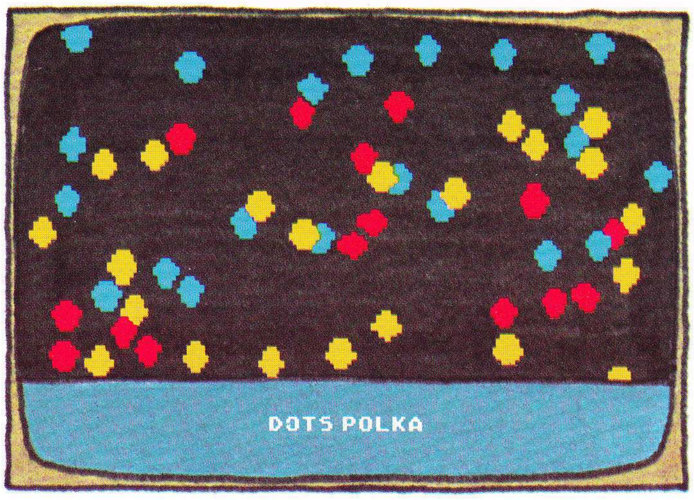


Figure 5-7

The SO: and PA: commands can be used together to make music. The table in Figure 5-8 shows the PAUSE values for various musical notes.

PAUSE VALUES FOR MUSICAL NOTES

PA:16	Sixteenth Note
PA:32	Eighth Note
PA:64	Quarter Note
PA:128	Half Note
PA:256	Whole Note

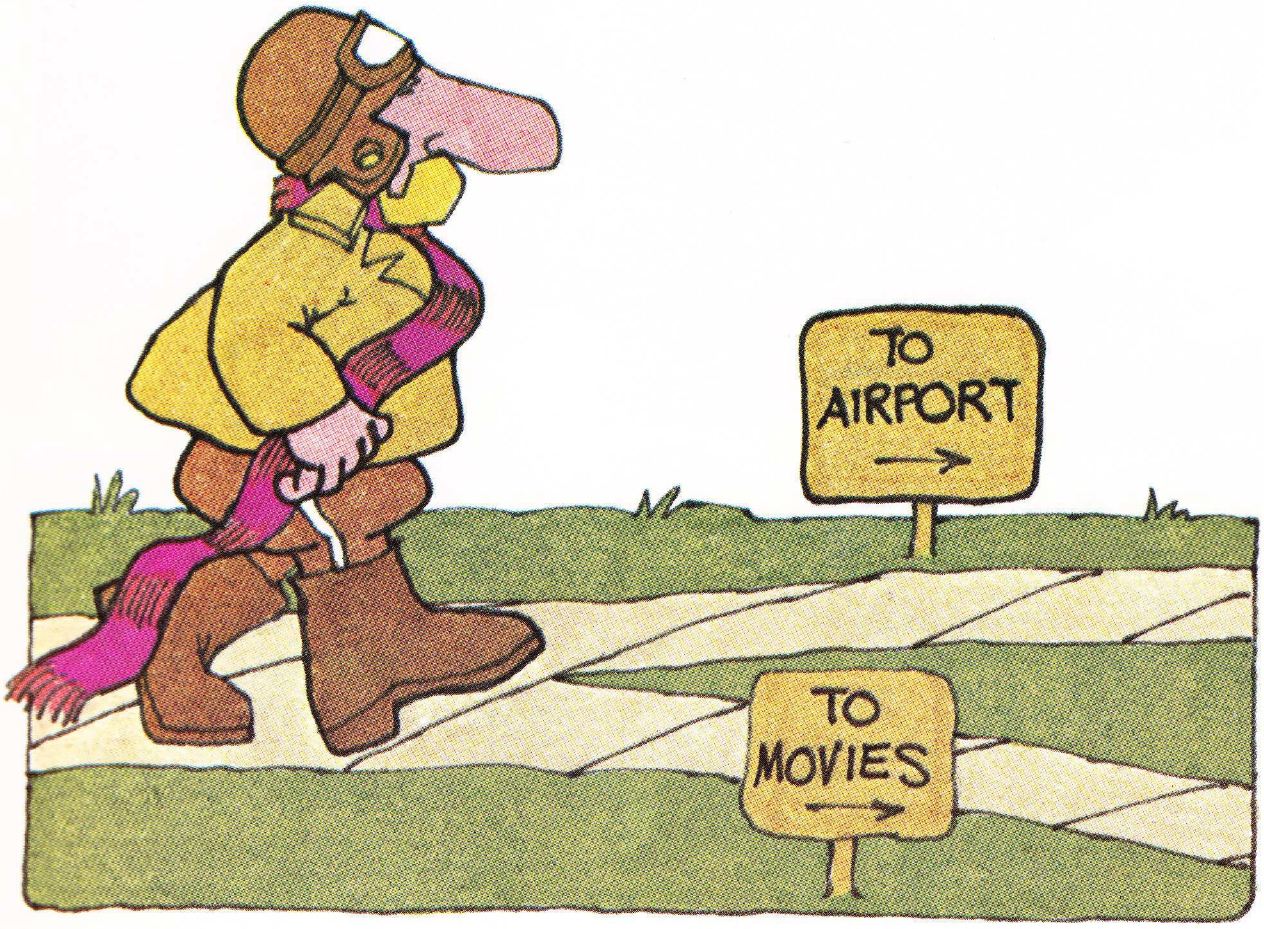
Figure 5-8

The program shown in Figure 5-9 uses SO: and PA: to play the the song "Mary Had a Little Lamb."

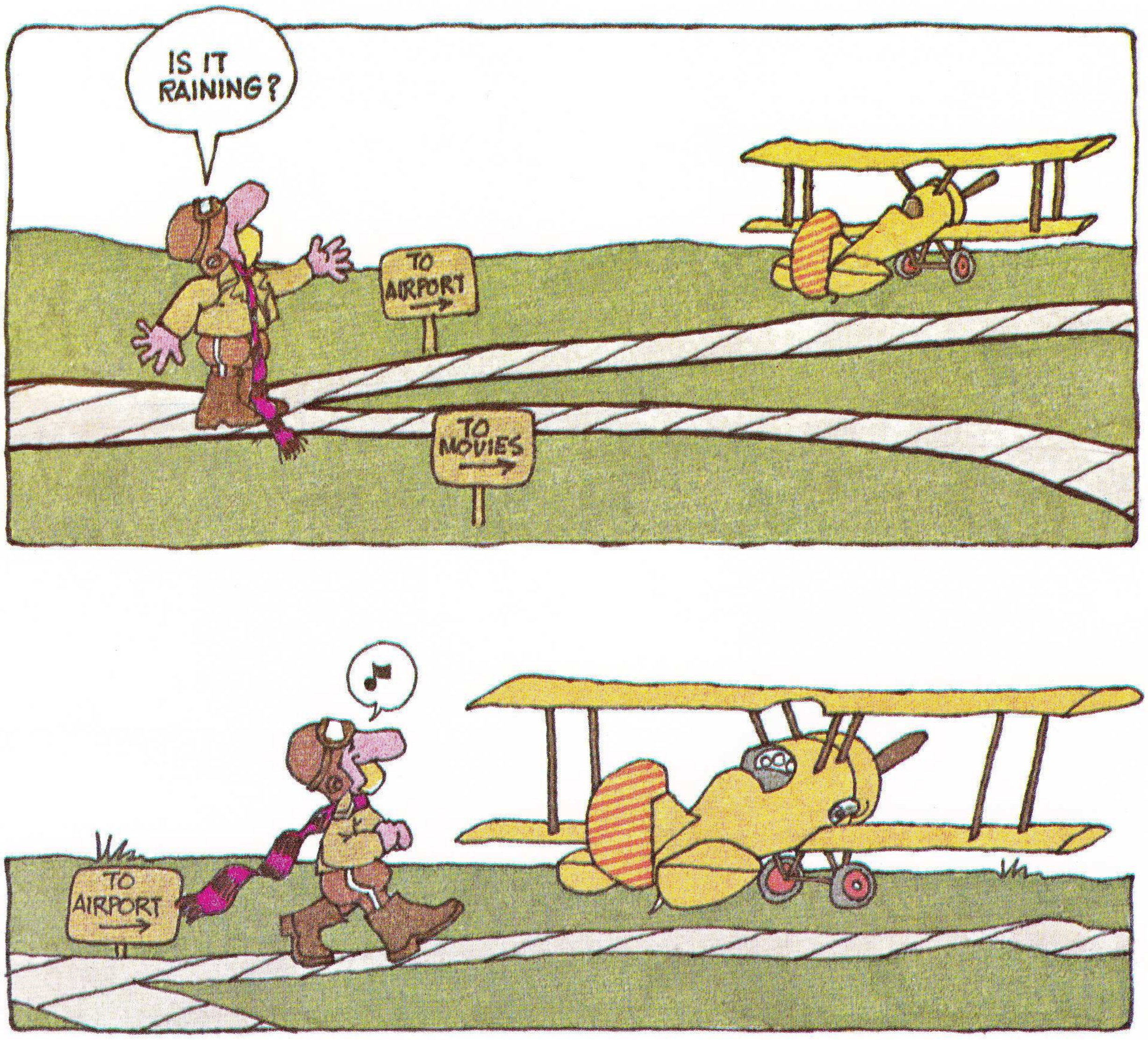
```
10 R:MARY HAD A LITTLE LAMB
20 R:ARRANGED IN 4 VOICES
30 R:FOR COMPUTER
40 C:#T=0
50 *CODA
60 C:#T=#T+1
70 SO:17,13,8
80 PA:24
90 SO:15,13,8
100 PA:8
110 SO:13,8
120 PA:32
130 SO:15,13,8
140 PA:32
150 SO:17,13,8
160 PA:32
170 SO:17,13,8
180 PA:32
190 SO:17,13,8
200 PA:64
210 SO:15,12,8
220 PA:32
230 SO:15,12,8
240 PA:32
250 SO:15,12,8
260 PA:64
270 SO:17,13,8
280 PA:32
290 SO:20,17,13,8
300 PA:32
310 SO:20,17,13,8
320 PA:64
330 SP:17,13,8
340 PA:24
350 SO:15,13,8
360 PA:8
370 SO:13,8
380 PA:32
390 SO:15,13,8
400 PA:32
410 SO:17,13,8
420 PA:32
430 SO:17,13,8
440 PA:32
450 SO:17,13,8
460 PA:32
470 SO:15,12,8
480 PA:32
490 SO:15,12,8
500 PA:32
510 SO:17,13,8
520 PA:32
530 SO:15,12,8
540 PA:32
550 SO:13,8,5,1
560 PA:192
570 SO:0
580 J(#T<2):*CODA
590 E:

READY
■
```

Figure 5-9

6 CONDITIONALS



CONDITIONAL COMMANDS



A *condition* is something about which a YES or NO decision can be made. For example, the weather could be a condition for deciding what to do today: Is it raining outside? If YES, go to the movies; if NO, go flying.

In PILOT, there are two types of conditions:

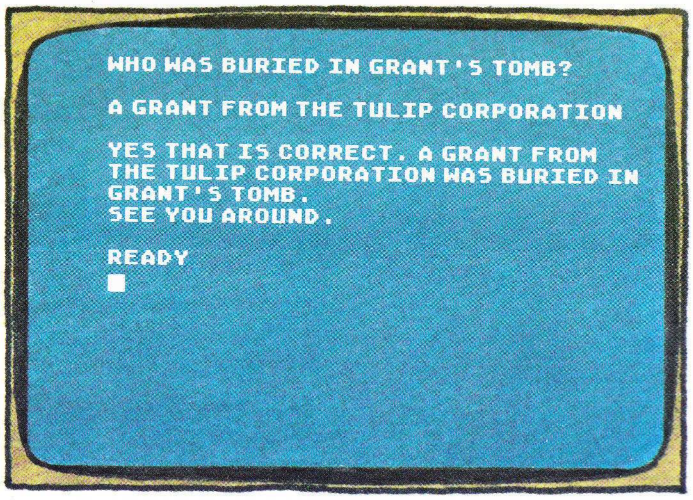
- String match conditions
- Numeric conditions

Y OR N

String Match Conditions

A MATCH command always sets up a YES or NO condition: YES (Y) there was a match, or NO (N), there was not a match. For example, in the program shown in Figure 6-1, any answer containing the name "GRANT" produces a YES condition; any answer not containing "GRANT" produces a NO condition.

```
10  R:GRANTS TOMB
20  T:WHO WAS BURIED IN GRANT'S
TOMB?
30  T:
40  A:$ANYONE
50  M:  GRANT
60  T:
70  TY:YES, THAT IS CORRECT.
$ANYONE WAS BURIED IN GRANT'S
TOMB.
80  TN:NO, $ANYONE WAS NOT BURIED
IN GRANT'S TOMB; GRANT WAS.
90  T:SEE YOU AROUND.
100 E:

READY
■
```



```
WHO WAS BURIED IN GRANT'S TOMB?

A GRANT FROM THE TULIP CORPORATION

YES THAT IS CORRECT. A GRANT FROM
THE TULIP CORPORATION WAS BURIED IN
GRANT'S TOMB.
SEE YOU AROUND.

READY
■
```

Figure 6-1

You can make any PILOT command (T:, A:, M:, J:, C:, U:, E:, GR:, SO:, and PA:) conditional on a YES-match or NO-match by putting a Y or an N between the command and the colon (:). Here are some examples:

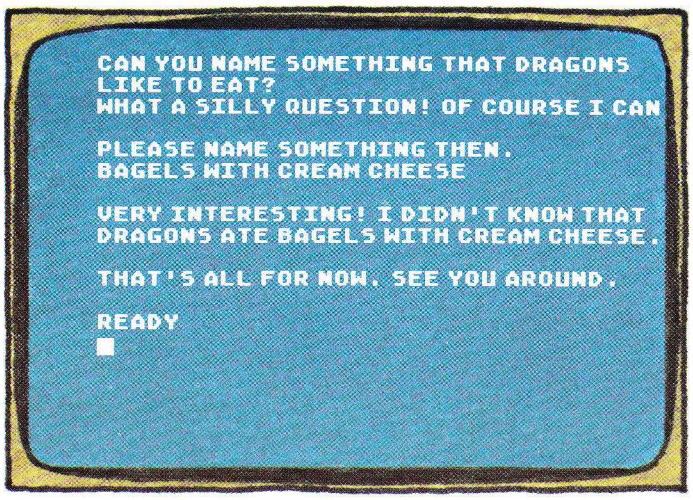
TY:HELLO	If there is a match, type "HELLO"; otherwise, skip this command.
TN:WRONG	If no match, type "wrong"; otherwise, skip this command.
EY:	If there is a match, end the program; otherwise, keep going.
JN: ✻ NEXT	If no match, jump to ✻ NEXT; otherwise, skip this command.
UY: ✻ BOX	If there is a match, use the module ✻ BOX; otherwise skip this command.
SOY:13,17	If there is a match, sound notes 13 and 17; otherwise skip this command.

Figure 6-2 shows a program that uses String Match Conditions.

```
10 R:DRAGON FEED
20 T:CAN YOU NAME SOMETHING THAT
30 T:DRAGONS LIKE TO EAT?
40 A:$FOOD
50 M:  YES, YUP, OK, FINE,
SURE, CAN, THINK I CAN, THINK SO
60 T:
70 TY:PLEASE NAME SOMETHING THEN.
80 AY:$FOOD
90 M:  NO, NOT, CAN'T, DON'T,
UNSURE
100 TY:SORRY TO HEAR THAT.
110 JY:*YONDER
120 M:PEOPLE, PRIN, KNI, CAKE,
PRETZ, FLOWER
130 T:
140 TY:YES,DRAGONS SURE DO EAT
$FOOD.
150 TN:VERY INTERESTING! I DIDN'T
KNOW THAT DRAGONS ATE $FOOD.
160 *YONDER
170 T:
180 T:THAT'S ALL FOR NOW. SEE YOU
AROUND.
190 E:

READY
■
```

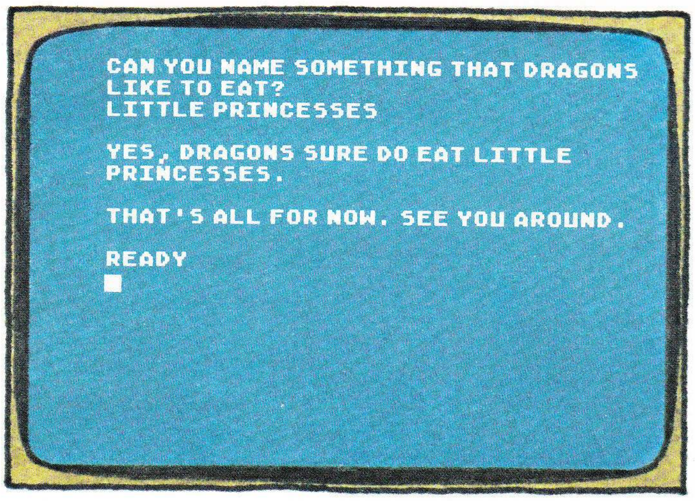
Figure 6-2

You can see how this program works in Figures 6-3 and 6-4.



```
CAN YOU NAME SOMETHING THAT DRAGONS
LIKE TO EAT?
WHAT A SILLY QUESTION! OF COURSE I CAN

PLEASE NAME SOMETHING THEN.
BAGELS WITH CREAM CHEESE

VERY INTERESTING! I DIDN'T KNOW THAT
DRAGONS ATE BAGELS WITH CREAM CHEESE.

THAT'S ALL FOR NOW. SEE YOU AROUND.

READY
■
```

Figure 6-3



```
CAN YOU NAME SOMETHING THAT DRAGONS
LIKE TO EAT?
LITTLE PRINCESSES

YES, DRAGONS SURE DO EAT LITTLE
PRINCESSES.

THAT'S ALL FOR NOW. SEE YOU AROUND.

READY
█
```

Figure 6-4

COMPARISON

Numeric Conditions

A YES/NO decision based on comparing two numbers is called a **numeric condition**. For example, if the amount of money in my pocket is equal to, or more than, the price of a hot fudge sundae, I will order the sundae (YES condition); if it is less, I will order something less expensive (NO condition).

There are six possible comparisons you can make between two numbers. In PILOT, you express these comparisons using the following symbols:

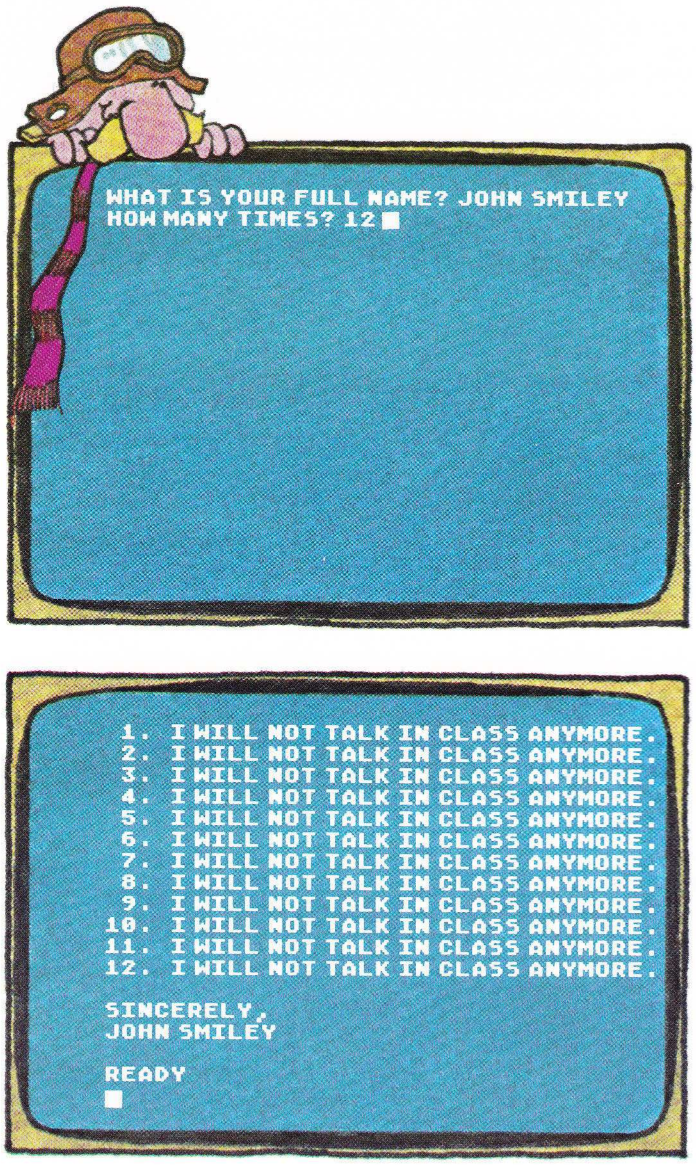
#A = #B	#A equals #B
#C < 20	#C is less then 20
#G > #H	#G is greater than #H
#G <> 3	#G is not equal to 3
#L <= 5	#L is less than or equal to 5
#B >= #C	#B is greater than or equal to #C

You can make numeric comparisons between two numeric variables, or between a numeric variable and a constant, and you can make any PILOT command conditional on a numeric comparison. Enclose the comparison in parentheses and place it between the command and the colon (:), like the Y or N in a string match condition. Here are some examples:

J (#G<3): ✱ AGAIN	JUMP if the value is #G is less than 3 to ✱ AGAIN; otherwise keep going.
T (#A=#B): RIGHT!	Type "Right!" if the value of #A equals the value of #B; otherwise, skip this command.
A(#P<=5):$ANS	Accept input from the keyboard if #P is less than or equal to 5; otherwise skip this command and keep going.
SO(#N<>#R): 3,7	Sound the tones 3 and 7 if the value of #N is not equal to the value of #R; otherwise, skip this command.

A common use of numeric conditions is setting up counting loops in programs. Figure 6-5 shows a simple example of how a numeric condition is used to print something repeatedly for a desired number of times.

```
10  R:GOOD BEHAVIOR
20  T:WHAT IS YOUR FULL NAME? \
30  A:$NAME
40  T:HOW MANY TIMES? \
50  A:#T
60  T:↰  [CLEARS THE SCREEN]
70  C:#C=1
80  *REPEAT
90  T(#C<10): \
100  T:#C. I WILL NOT TALK IN CLASS
ANYMORE.
110  C:#C=#C+1
120  J(#C<=#T):*REPEAT
130  T:
140  T:SINCERELY,
150  T:$NAME
160  E:

READY
■
```



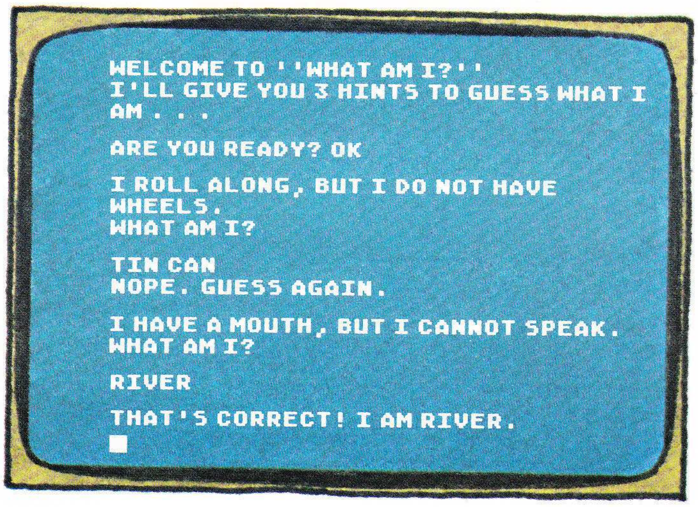
```
WHAT IS YOUR FULL NAME? JOHN SMILEY
HOW MANY TIMES? 12■
```

```
  1.  I WILL NOT TALK IN CLASS ANYMORE.
  2.  I WILL NOT TALK IN CLASS ANYMORE.
  3.  I WILL NOT TALK IN CLASS ANYMORE.
  4.  I WILL NOT TALK IN CLASS ANYMORE.
  5.  I WILL NOT TALK IN CLASS ANYMORE.
  6.  I WILL NOT TALK IN CLASS ANYMORE.
  7.  I WILL NOT TALK IN CLASS ANYMORE.
  8.  I WILL NOT TALK IN CLASS ANYMORE.
  9.  I WILL NOT TALK IN CLASS ANYMORE.
 10.  I WILL NOT TALK IN CLASS ANYMORE.
 11.  I WILL NOT TALK IN CLASS ANYMORE.
 12.  I WILL NOT TALK IN CLASS ANYMORE.

SINCERELY,
JOHN SMILEY

READY
■
```

Figure 6-5

Figure 6-6 shows how you can use a numeric condition to limit a person to three guesses in a guessing game.

```
10  R:WHAT AM I?
20  T:WELCOME TO ''WHAT AM I?''
30  T:I'LL GIVE YOU 3 HINTS \
40  T:TO GUESS WHAT I AM . . .
50  PA:60
60  T:
70  T:ARE YOU READY? \
80  A:
90  T:
100  M:  Y,SURE,OK,FINE,ALRIGHT
110  JN:*GOODBYE
120  C:#G=1
130  *LOOP
140  T(#G=1) :I ROLL ALONG, BUT I DO
NOT HAVE WHEELS.
150  T(#G=2) :I HAVE A MOUTH, BUT I
CANNOT SPEAK.
160  T(#G=3) :I HAVE A BED, BUT I
NEVER SLEEP.
170  T:WHAT AM I?
180  T:
190  A:$ANSWER
200  M:RIVER,STREAM,CREEK
210  JY:*RIGHT
220  J(#G=3) :*NOMORE
230  T:NOPE, GUESS AGAIN.
240  T:
250  C:#G=#G+1
260  J:*LOOP
270  *NOMORE
280  T:
290  T:NOPE, THAT'S THREE GUESSES.
300  T:I AM A RIVER.
310  J:*GOODBYE
320  *RIGHT
330  T:
340  T:THAT'S CORRECT! I AM
$ANSWER.
350  *GOODBYE
360  T:
370  PA:60
380  T:SEE YOU LATER.
390  E:

READY
■
```



```
WELCOME TO ''WHAT AM I?''
I'LL GIVE YOU 3 HINTS TO GUESS WHAT I
AM . . .

ARE YOU READY? OK

I ROLL ALONG, BUT I DO NOT HAVE
WHEELS.
WHAT AM I?

TIN CAN
NOPE. GUESS AGAIN.

I HAVE A MOUTH, BUT I CANNOT SPEAK.
WHAT AM I?

RIVER

THAT'S CORRECT! I AM RIVER.
■
```

Figure 6-6

JM:

Jump on Match

The JM: command causes the computer to jump, conditional on matching one of a list of match alternatives. It is a *variable* jump rather than a constant jump. In the simple J: ✻ LABEL command, you always jump to a *single* ✻ LABEL. This variable form of the JUMP command lets you jump to one of a variety of ✻ LABELS, depending on what matches in an M: command.

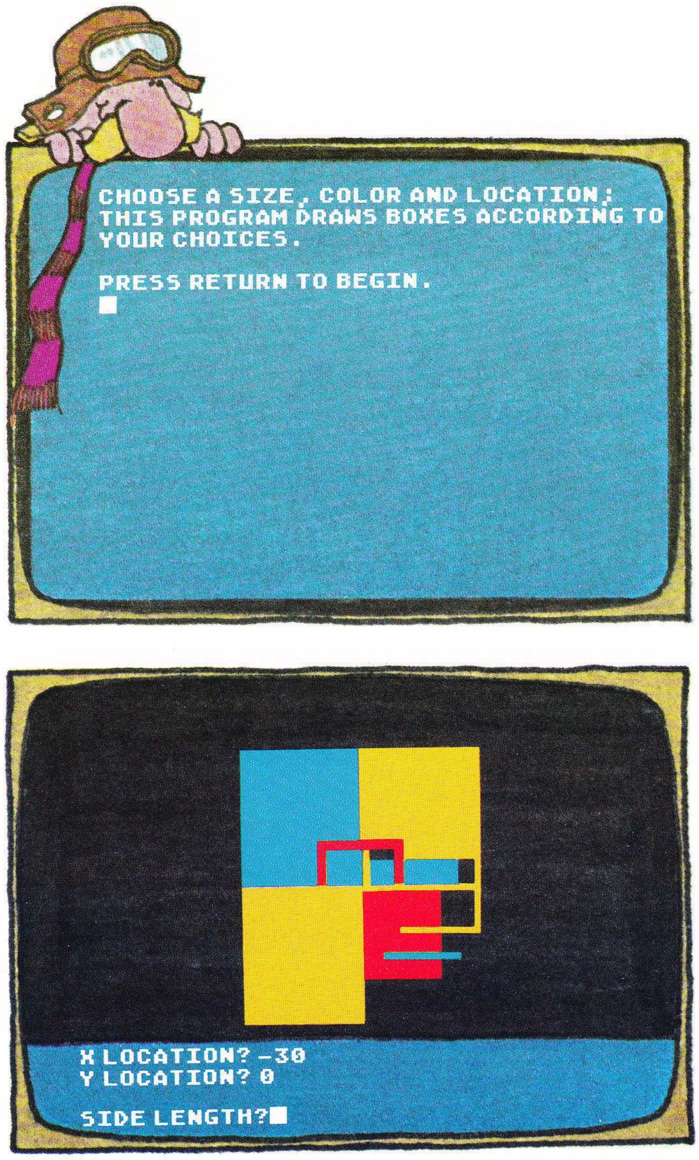
The JM: command must be paired with an M: command as follows:

```
A:
M: String1, String2, String3, String4.....
JM: ✻ LABEL1, ✻ LABEL2, ✻ LABEL3, ✻ LABEL4.....
```

The JM: command should have the same number of labels as the M: has strings. The JM: command will jump to the ✻ LABEL in the same position as the first string that matches the accepted input. In other words, if STRING1 is a match, JM: will jump to ✻ LABEL1; if STRING2 is a match, JM: will jump to ✻ LABEL2, and so on. If there is no match, or if the position of a matched string does not correspond to the position of a ✻ LABEL, then the JM: command will be skipped.

Figure 6-7 is an example of the use of the JM: command. The program draws colored boxes on the screen in the size, color, and location of your choice.

```
10  R:BOXES IN COLOR
20  T:CHOOSE A SIZE, COLOR AND
LOCATION.
30  T:THIS PROGRAM DRAWS BOXES
ACCORDING TO YOUR CHOICES.
40  T:
50  T:PRESS RETURN TO BEGIN
60  A:
70  GR:CLEAR
80  *LOOP
90  T:
100  T:SIDE LENGTH? \
110  A:#S
120  T:WHAT COLOR (RED,BLUE,
YELLOW,BLACK)? \
130  A:$COLOR
140  M: RE, BLU, YE, BLA
150  JM:*RED,*BLU,*YEL,*BLA
160  *CONTINUE
170  T:
180  T:X LOCATION? \
190  A:#X
200  T:Y LOCATION? \
210  A:#Y
220  U:*BOX
230  J:*LOOP
240  *RED
250  GR:PEN RED
260  J:*CONTINUE
270  *BLU
280  GR:PEN BLUE
290  J:*CONTINUE
300  *YEL
310  GR:PEN YELLOW
320  J:*CONTINUE
330  *BLA
340  GR:PEN ERASE
350  J:*CONTINUE
360  *BOX
370  GR:GOTO #X,#Y
380  GR:TURNTO 90;3(DRAW #S;TURN
90)
390  GR:FILL #S
400  E:

READY
■
```



```
CHOOSE A SIZE, COLOR AND LOCATION;
THIS PROGRAM DRAWS BOXES ACCORDING TO
YOUR CHOICES.

PRESS RETURN TO BEGIN.
■
```

```
X LOCATION? -30
Y LOCATION? 0

SIDE LENGTH?■
```

Figure 6-7

APPENDIX A

Special Function Keys and Screen Editing

SPECIAL FUNCTION KEYS

A number of keys on the **ATARI 400/800™ Personal Computer System** keyboards have a special function or purpose. These functions are described below.

⅄	**Reverse (Inverse) Video Key**, or ATARI Logo Key: Pressing this key causes the text to be reversed on the screen (dark text on light background). Press the key a second time to return to normal text.
CAPS LOWR	**Lowercase Key**: Pressing this key shifts the screen characters from uppercase (capitals) to lowercase. To restore the characters to uppercase, press the SHIFT key and the CAPS LOWR key simultaneously.
ESC	**Escape Key**: Pressing this key causes a command to be entered into a program for later execution. Example: To clear the screen, you would enter: 10 T: ESC CTRL CLEAR This will cause a curved arrow to appear on your screen as follows: **10 T: ↰**
CTRL	**Control Key**: This key is used in conjunction with other keys to print special graphics control characters. See back cover or PILOT Reference Card for the specific keys and their screen-character representations.
CTRL 1	**Control 1**: Holding down CTRL and pressing **1** temporarily stops whatever is happening on the screen (e.g., a program listing). Pressing CTRL **1** again restarts the screen action.
BREAK	**Break Key**: Pressing this key during program execution causes execution to stop. The line number and statement where the execution stopped are displayed on the screen followed by ***READY***.

SYSTEM RESET **System Reset Key**: Similar to BREAK in that pressing this key stops program execution. Also returns the screen display to the Text mode, clears the screen, and returns margins and other variables to their default values.

SET CLR TAB **Tab Key**: Press SHIFT and the SET CLR TAB keys simultaneously to set a tab. To clear a tab, press the CTRL and SET CLR TAB keys simultaneously. Used alone, the SET CLR TAB advances the cursor to the next tab position. In Deferred mode, set and clear tabs by preceding the above with a line number, the command T:, a quotation mark, and pressing the ESC key.

```
Examples: 100 T:ESC SHIFT SET CLR TAB
          200 T:ESC CTRL SET CLR TAB
```

Default tab settings are placed at columns 5, 14, 22, 30, and 38.

DELETE BACK S **Back Space Key**: Pressing this key replaces the character to the left of the cursor with a space and moves cursor back one space.

CLEAR **Clear Key**: Pressing this key while holding down the SHIFT or CTRL key blanks the screen and puts the cursor in the upper left corner.

RETURN **Return Key**: Terminator to indicate an end of a line of PILOT. Pressing this key causes a numbered line to be interpreted and added to a PILOT program in RAM.

An unnumbered line (in Immediate mode) is interpreted and executed immediately. Any variables are placed in a variable table.

SCREEN EDITING

The keyboard and display are logically combined for a mode of operation known as screen editing. Each time a change is completed on the screen, the RETURN key must be pressed. Otherwise, the change is not made to the program in memory.

```
Example: 10 T:REMEMBER TO PRESS RETURN
         15 T:AFTER LINE EDIT
         20 T:THIS LINE IS GOING AWAY
         30 T:THIS WILL BE THE FIRST
         40 T:LINE ON THE SCREEN.
         50 E:
```

To delete line 20 from the program, type the line number and press the RETURN key. Merely deleting the line from the screen display does **not** delete it from the program.

In addition to the special function keys described above, there are cursor control, insert, and delete keys that allow immediate editing capabilities. These keys are used in conjunction with the CTRL or SHIFT keys as follows:

CTRL ↑	Moves cursor up one physical line without changing the program or display.
CTRL →	Moves cursor one space to the right without disturbing the program or display.
CTRL ↓	Moves cursor down one physical line without changing the program or display.
CTRL ←	Moves cursor one space to the left without disturbing the program or display.

Like the other keys on the ATARI keyboard, holding the cursor control keys for more than ½ second causes the keys to repeat.

INSERT	**Insert key**: Press the SHIFT and SYSTEM RESET keys simultaneously to insert a line. To insert a single character, press the CTRL and INSERT keys simultaneously.
SHIFT INSERT	Inserts one physical line.
CTRL INSERT	Inserts one character space.
DELETE BACK S	**Delete key**: Press the SHIFT and DELETE BACK S keys simultaneously to delete a line. To delete a single character, press CTRL DELETE BACK S simultaneously.
SHIFT DELETE	Deletes one physical line.
CTRL DELETE	Deletes one character or space.

Note: You should not use screen editing features other than Backspace/Delete in AUTO mode. Instead, return to the Immediate mode and LIST your program to use screen editing.

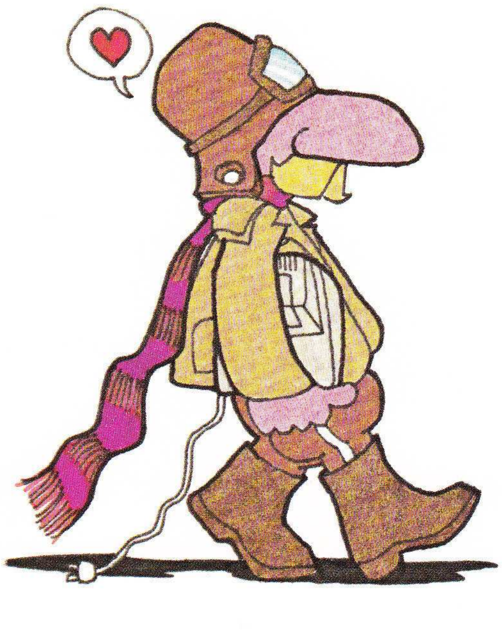
APPENDIX B
PILOT I/O Error Codes

This appendix lists the ATARI 400/800 Computer Systems' I/O error codes within the context in which they will be seen in ATARI PILOT. Not all of the system codes are presented here, because some of them cannot occur within the PILOT environment.

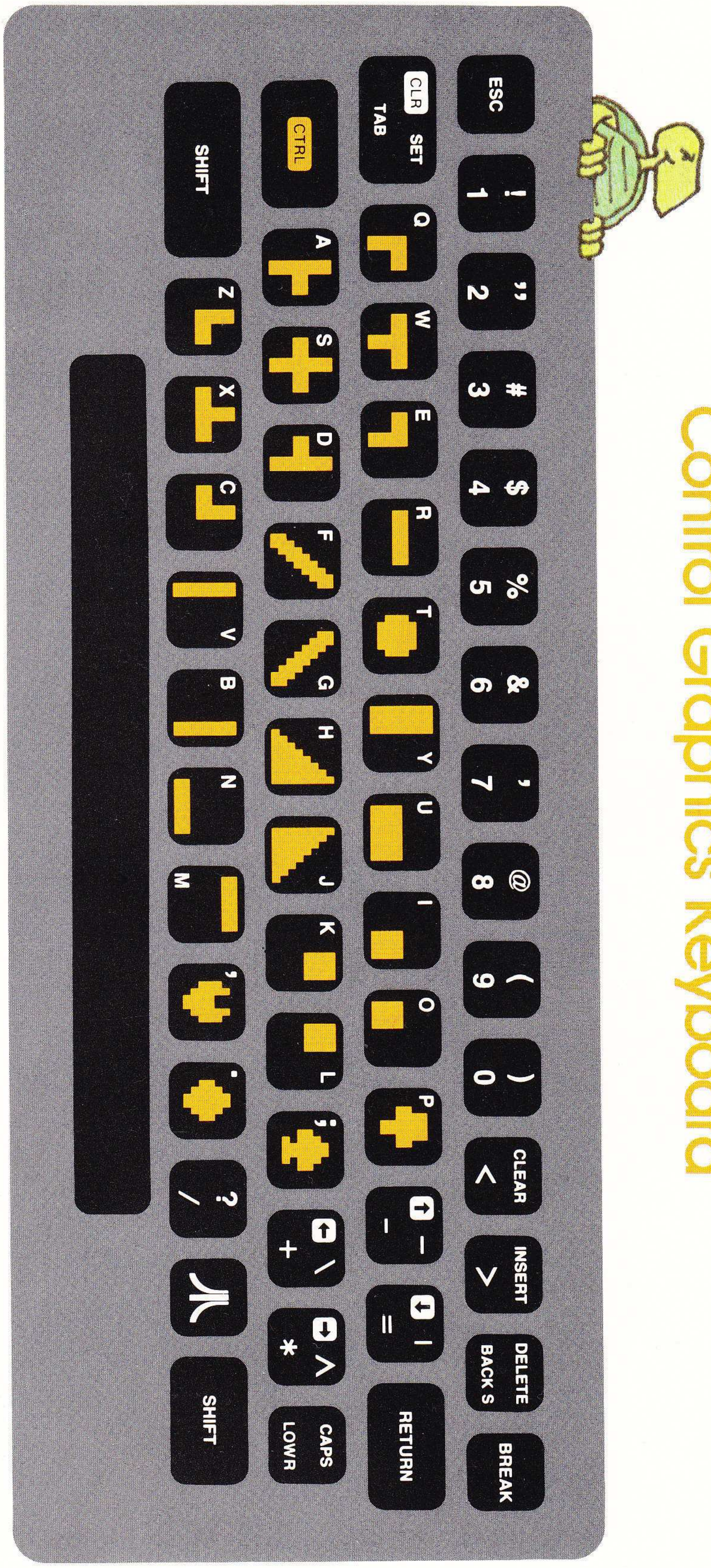
130	A nonexistent device was specified.
131	A READ command followed a WRITE command with the same device specified.
135	A WRITE command followed a READ command with the same device specified.
136	End-of-file condition.
138	Device timeout; device doesn't respond. (See Note)
139	Device NAK. (See Note)
140	Serial bus framing error. (See Note)
141	Screen cursor out of range (READ from or WRITE to 'S').
142	Serial bus data from overrun. (See Note)
143	Serial bus data from checksum error. (See Note)
144	Device DONE error. (See Note)
145	Disk read after write compare error. (See Note)
146	Function not implemented for device (e.g., OUT:K).
147	Insufficient RAM for operating the graphics screen.
160	Disk drive number error.
161	Too many concurrent disk files being accessed.
162	Disk is full (no free sectors).
163	Fatal system data I/O error.
164	File number mismatch. (See Note)

165	Disk file naming error.
167	Disk file locked.
169	Disk directory full (64 files).
170	Disk file not found in directory.

Note : These errors indicate problems over which the user has no direct control; they are due to hardware problems and should seldom be seen.



Control Graphics Keyboard



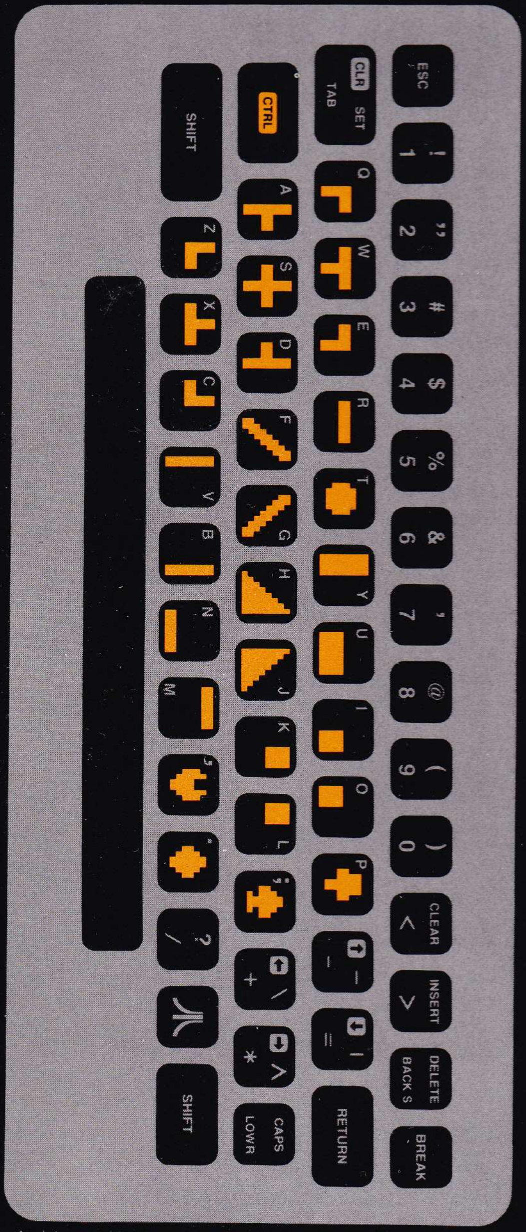
Programmed Inquiry Learning Or Teaching

with "Turtle" Graphics

Pocket Reference Card

ATARI® A Warner Communications Company

CONTROL GRAPHICS KEYBOARD



VARIABLE TYPES

Numeric variables #A to #Z	Integer variables can contain a number from -32768 to 32767.
String variables $ followed by alphanumeric name (example: $FOOD)	String variables can contain literal strings of up to 255 characters.
Machine variables % followed by one or two characters (example: %Y)	Machine variables contain the status of light pen, joystick, cursor location, and other machine parameters.
Pointer variables @ followed by a number or by B and a number (example: @B77)	Pointer variables refer to the contents of bytes (B) or words whose decimal address in memory is given by the numeric portion of the variable name.

ARITHMETIC OPERATORS

+	Addition
−	Subtraction
*	Multiplication
/	Division
\	Modulo operation (remainder after division)
?	Random number generation (generates random number between -32768 and 32767).

RELATIONAL OPERATORS

=	Equal
>	Greater than
<	Less than
>=	Greater or equal
<=	Less or equal
<>	Not equal

Atari, Inc., Computer Division
P.O. Box 427
Sunnyvale, CA 94086

COMMANDS

Command	Function	Example	Result
T:	Type to screen	T:THIS IS SOME TEXT	Prints THIS IS SOME TEXT on the screen.
		T:#A or T:$NAME	Prints contents of numeric variable (#A) or string variable ($NAME) on the screen.
A:	Accept from keyboard	A:	Places keystrokes in the accept buffer.
		A:#B or A:$FOOD	Places keystrokes in the accept buffer and in the variable #B or $FOOD.
M:	Match command	M: YES, YUP, SURE	Checks each entry in the list against contents of accept buffer and sets Y and N flags. (See CONDITIONALS.)
R:	Remark	R: LOOP BEGINS HERE	Remark for user only—no effect on program.
J:	Jump command	J: *JUMPHERE	Jumps execution to the label *JUMPHERE.
E:	Ends program	E:	Terminates program execution.
C:	Compute command	C: #A=#A+#B	Performs computation shown to right of colon (add #B to #A and put the result in #A).
U:	Use module	U: *NEXTSET	Uses the module *NEXTSET and returns at the end of that module.
SO:	Sound command	SO: #A,#B,#C,#D	Generates up to four musical tones (tones increase by ½ step per number).
		SO: 13	Generates Middle C.
PA:	Pause command	PA: 5	Stops PILOT execution for 5/60 second.
JM:	Jump on match	JM: *HERE, *START, *END	Jumps to label corresponding to match field.
MS:	Match to make strings	MS: IS, ARE, WILL	Splits contents of buffer into $LEFT, $MATCH, and $RIGHT.
POS:	Position text cursor	POS: 12,15	Positions cursor at 12th column and 15th row.
VNEW:	Clears variables	VNEW:	Clears all variables.
		VNEW: # or VNEW: $	Clears all numeric or all string variables.
CALL:	Calls machine language program	CALL: 4096	PILOT jumps to execute machine language program at decimal location 4096.

CONDITIONALS

The execution of any PILOT command can be made conditional on the presence or absence of a string match or on the truth of a numeric relationship.

String Match Conditions
The conditional variables Y (YES) and N (NO) are set by the result of a match (M:) command.

Numeric Conditions
Relational operators are used to compare two numbers or variables. If the relation is true, the accompanying PILOT command will be executed.

Examples:

TY: GOOD GUESS (prints GOOD GUESS if result of preceding match command was true).

JN: *RESTART (jumps to *RESTART if result of preceding match command was false).

E (#A < 7): (ends program execution if #A is less than 7).

Comm
CLEAR
QUIT
PEN
GOTO
DRAW
TURNT
FILLTO
GO
DRAW
TURN
FILL

LIST
RUN
SAVE
DOS
NEW
LOAD
AUTO
REN
TAPE:
TAPE:
TSYNC
READ
WRITE
CLOSE

creen.

(#A) or string

fer.

fer and in the

t contents of
s. (See

program.

PHERE.

ht of colon
#A).

urns at the

(tones in-

ond.

atch field.

, $MATCH,

d 15th row.

ables.

anguage pro-

the truth of a

OT command

GRAPHICS COMMANDS

Command	Function	Example	Result
CLEAR	**Clear screen**	GR: CLEAR	Clears text and graphics window.
QUIT	**Return to text mode**	GR: QUIT	Clears graphics window and returns to text-only mode.
PEN	**Sets pen color to red, yellow, blue, or erase (background color) or lifts pen**	GR: PEN RED	Sets pen color to red.
		GR: PEN UP	Lifts pen.
GOTO	**Cursor move (absolute)**	GR: GOTO 0,0	Moves cursor to 0,0 (center of screen).
DRAWTO	**Draw line (absolute)**	GR: DRAWTO 10,15	Draws line from previous cursor position to coordinate 10,15.
TURNTO	**Turn cursor heading (absolute)**	GR: TURNTO 0	Turns cursor to 0 degrees (straight up).
FILLTO	**Fill (absolute)**	GR: FILLTO -6,5	Draws line from previous cursor position to -6,5 and fills to the right with color.
GO	**Cursor move (relative)**	GR: GO 25	Moves cursor 25 units in present heading.
DRAW	**Draws line (relative)**	GR: DRAW 17	Draws 17 units in present heading.
TURN	**Turns cursor (relative)**	GR: TURN 48	Turns cursor heading by 48 degrees to the right.
FILL	**Fill (relative)**	GR: FILL 30	Draws line 30 units along present heading and fills to the right with color.

EXECUTIVE AND INPUT/OUTPUT COMMANDS

LIST	Lists program statements on screen.
RUN	Starts executing program at first statement.
SAVE	Saves program to cassette (SAVE C:) or to disk (SAVE D:filename). To list a program on a printer, type SAVE P:.
DOS	Enters the Disk Operating System command menu.
NEW	Erases all PILOT program statements in memory.
LOAD	Loads a program from cassette (LOAD C:) or disk (LOAD D:filename) without erasing resident program list.
AUTO	Enters auto numbering made for writing programs.
REN	Renumbers program lines.
TAPE: ON	Turns cassette motor on.
TAPE: OFF	Turns cassette motor off.
TSYNC:	Allows PILOT to synchronize specially prepared tapes.
READ	Reads data from attached device to the accept buffer (**Example:** READ: C, $DATA reads from cassette to buffer and variable $DATA).
WRITE	Writes data to an attached device (**Example:** WRITE: R1, HELLO THERE writes HELLO THERE to RS-232 port R1).
CLOSE	Closes device previously opened with READ or WRITE (**Example:** CLOSE:C).

C017812 REV.2