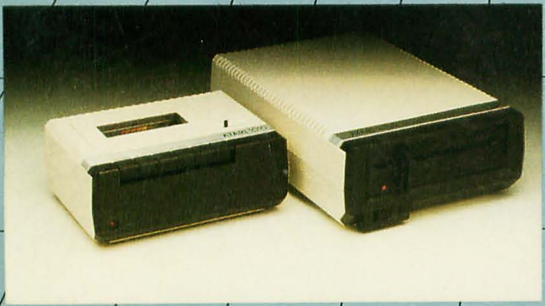
ATARI® BASIC

XL™ EDITION



"Runs on XE MACHINES"

BOB ALBRECHT
LEROY FINKEL
JERALD R. BROWN

ATARI® BASIC

XL™ Edition

ATARI® BASIC

XL™ Edition

Bob Albrecht
LeRoy Finkel
Jerald R. Brown

A Wiley Press Book
John Wiley & Sons, Inc.
New York • Chichester • Brisbane • Toronto • Singapore

Publisher: Judy V. Wilson
Editor: Theron Shreve
Managing Editor: Katherine Schowalter
Composition & Make-Up: The Publisher's Network

Library of Congress Cataloging in Publication Data

Albrecht, Bob, 1930–
Atari BASIC, XL edition.

Includes index.
1. Atari computer – Programming. 2. Basic (Computer program language) I. Finkel, LeRoy. II. Brown, Jerald, 1940– . III. Title.
QA76.8.A82A43 1984 001.64'2 84-20803

ISBN 0-471-80726-5

Printed in the United States of America

85 86 10 9 8 7 6 5 4

Table of Contents

To the Reader

This book is for people who want to learn how to use, program and enjoy ATARI computers. Using this book, you can teach yourself how to read and understand ATARI BASIC. This is a beginner's book; no previous computer experience is required.

ATARI computers are friendly, fun, easy to use by kids and adults, at home, school or elsewhere. They are superb low-cost computers for educational and recreational use.

That's really what this book is all about – to help you learn to use an ATARI computer for your own recreation and education. So play and learn your way through this book – have a good time!

You will learn ATARI BASIC, a language you use to tell the computer what *you* want it to do. Since BASIC is a language, learn it as you might learn any language, such as English, Spanish, or Swahili.

- Learn a little bit and use it.
- Learn a little more and use it.
- And so on. Be patient. There's no hurry. Becoming fluent in BASIC takes some time, but you can enjoy every moment.
- Be confident. BASIC is a simple language, much more easily learned than English, Spanish, or Swahili.

This book is not a reference manual. It is not a textbook. It doesn't even try to cover all of ATARI BASIC. *That* would take three or four books this size, if we explained everything as slowly and carefully as we do in this book.

If you can read a newspaper or a comic book, you can use this book to help you teach yourself to read, understand, and use BASIC. Before you begin, browse through Appendix I, "Look Here First," for sources of additional information to help you learn.

Wait! Before plunging into Chapter 1, read "How to Use This Book."

How to Use This book

This is a Self-Teaching Guide you can use to teach yourself to read and understand ATARI BASIC. Each chapter consists of several sections. At the end of each section (except Chapter 1) are questions for you to answer or exercises for you to follow. If we ask you to do something on the computer, please do it, and your ATARI computer will be your best teacher. EXPERIMENT – try it and find out what happens.

If we ask questions, please answer them. We've left some space for your answers throughout this book, but if you need more space, use a separate sheet of paper. Don't peek at our answers until you have written yours, then compare. You will find our answers at the end of each quiz. Sometimes our answer is the correct answer; sometimes our answer is one of many possible answers and yours may be just as right or even better. You decide.

We encourage you to use this book while seated comfortably in front of an ATARI computer. Try our examples and exercises and you and the computer will soon know what does and doesn't work.

The first page of each chapter briefly lists what the chapter covers. Scan that list and if you feel you already know it, skip to the back of the chapter and take the Self-Test. If you miss a Self-Test question, review the section of the chapter in which the topic is discussed. You may also wish to browse through the chapter to find the little treasures (variations, challenges, puns, and so on) that we have buried there for you to find.

Things get more challenging as you progress through the book. If you are a beginner, start with the first chapter and work (play!) your way through the book. If however, you already know some BASIC, feel free to browse, skip around and meander through the book. Use the beginning-of-chapter objectives list and the end-of-chapter Self-Test as your guides.

Of course, look at the Table of Contents. You will see the titles of 12 chapters and 8 appendices. Curious? Go ahead – explore.

This book is full of challenges ranging from easy to hard to awful. You will find challenges with and without solutions and many of our challenges and solutions are unusual. If you are a puzzler – a person who likes to solve problems (especially in unusual ways!) – we think you will like our challenges.

One more thing. As you use this book and your ATARI computer, know this:

YOU CAN DO NOTHING WRONG!

You can't harm the computer by a typing mistake. You may make some but this is a natural part of exploring and learning. Risk it! Try it and find out what happens. You can learn more from your own patient exploration than from this or any book.

So, explore, enjoy, and tell us about your discoveries as you teach yourself how to use, program, and enjoy your computer.

Bob Albrecht
LeRoy Finkel
Jerald R. Brown
Dymax
P. O. Box 310
Menlo Park, CA 94026

Chapter One

The ATARI Computers

Relax. Make yourself comfortable. The first chapter is especially easy! In this chapter you will learn how to use, program, and enjoy your ATARI computer. So get ready for an adventure in sound, color and convivial computing!

ATARI computers are friendly, fun, and easy to learn by kids and adults, at home or at school. You can use them for education or recreation, or for more "serious" stuff such as math, science, home and personal management or small business.

You will also begin to learn "computerese", the jargon or terminology of computers. You will be able to better understand the literature of the computer age and enjoy discussing the exciting things you are doing with your ATARI computer.

When you finish this chapter, you will know a few things about the ATARI computers and be able to use the following words and phrases:

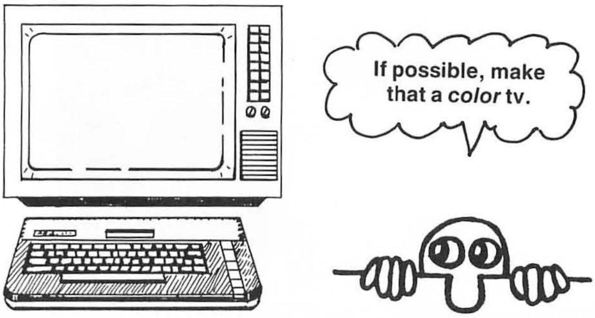
- ATARI computers: 400,800,600XL and 800XL
- programs
- cartridges
- cassettes and ATARI Program Recorder
- diskettes and disk drive
- paddles and joysticks
- chips: microprocessor, memory
- ROM (Read Only Memory)
- RAM (Random Access Memory)
- memory locations
- bytes and the mysterious "K"

At the end of this chapter, you will be ready to continue on to Chapter 2, in which you begin "talking" to your ATARI computer!

MEET THE ATARI COMPUTERS

This book is a beginner's guide to the ATARI home computers. You can use it to help you learn how to operate an ATARI 400, 800, 600XL or 800XL computer.

You begin with an ATARI computer and a television set.



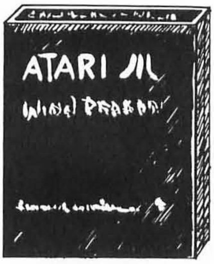
Of course, these two parts must be connected. How to do this is explained in the *ATARI Owner's Guide* that accompanies your ATARI computer, which we assume you already have. For information on ATARI products, write to ATARI, Home Computer Division, P.O. Box 50047, San Jose, CA 95150 or call toll-free 800-538-8543, (in California, call 800-672-1404).

PROGRAMS

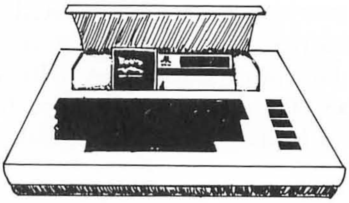
Thousands of *programs* are available for ATARI computers. A program is simply a procedure, a set of instructions, a plan for doing something. You may have already used, or created, or been frustrated by programs written in English (or another language). For example:

- A recipe for baking a cake.
- Instructions for opening a combination lock.
- Directions on how to get to your house from the airport.
- And, of course, those maddening instructions for assembling toys, tricycles, playpens, furniture and so on—probably at the last minute on Christmas Eve.

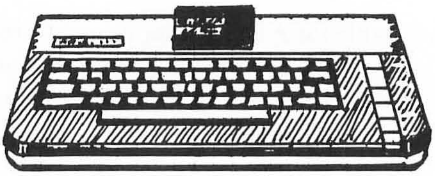
The easiest programs to use on your ATARI computer are available as *cartridges.*



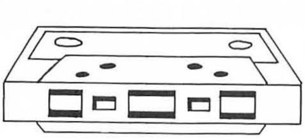
To install a cartridge in an ATARI 400 or 800, open the cartridge door, plug in the cartridge and close the door.

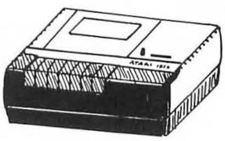


To install a cartridge in an ATARI 600XL or 800XL just plug in the cartridge. There is no door.



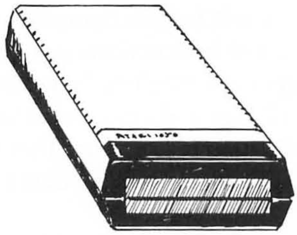
Many programs are available on prerecorded tape cassettes. A single cassette, which might cost from $3 to $30, might contain one large program or several small programs. These programs can be entered into the computer by means of the ATARI Model 410 or Model 1010 Program Recorder.

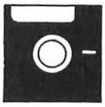




By using the program recorder, you can quickly load a program into your computer, then enjoy its use as written by an expert. As you learn to program in ATARI BASIC yourself, you can record your programs on tape cassettes then use the program recorder to read them back into the computer. Reduce finger fatigue by using the cassette recorder. Appendix A tells you how.

Programs are also available on diskettes and are entered by means of a *disk drive.*





Programs load *much* faster from a diskette than from a tape cassette. You can also save your programs on a diskette, then quickly reload them another time. For information on how to do this, consult the *Disk Operating System Reference Guide* that comes with the ATARI 810 or ATARI 1050 disk drive.

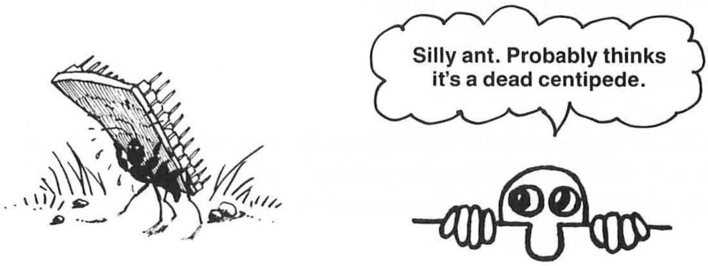
Did your ATARI computer include paddles or joysticks? These useful control devices can be used for games and other activities in which things are happening on the screen.

INSIDE THE COMPUTER

Inside the ATARI computer case is—the computer! It consists of a number of *chips* or tiny wafers containing thousands of microscopic circuits.

The tiny wafer is about this big: □

Well, that's too small for clumsy human fingers to handle. So the tiny wafer is put into a larger package with lots of legs that can be used to connect it to the other parts of the computer.



One of the chips is the *microprocessor* which does the actual work of computing. Other chips provide the *memory* of the computer. Still more chips control colors and graphics on the screen and give the ATARI computer a wide range of musical notes and other sounds.

ATARI BASIC

As you work (or play) through this book, you will learn a computer language called BASIC. A computer language is simply a set of instructions used to communicate with a computer. Compared to natural languages (English, Spanish, Swahili, etc.) a computer language is very simple. BASIC has a simple vocabulary (the list of words it knows) and a very formal syntax (rules of grammar that it follows). This book will help you enjoy teaching yourself how to "talk" to computers, using the language BASIC.

There are many variations, or dialects, of BASIC. In this book, you will learn ATARI BASIC which is considerably different from Microsoft™ BASIC, used on several home computers. An excellent version of Microsoft BASIC is available from ATARI in two storage versions: cartridge or diskette.

As you learn BASIC, you will read and understand computer programs written in BASIC as easily as a child learns to read and understand a natural language such as English.

Simple things first.

Then a little more.

Then a little more.

And so on.

As with a child, when you begin to understand, you may wish to express yourself in the language you are learning. You may write your own, original, never-before-seen-on-Earth-or-anywhere-else programs—your programs!

A BASIC program is a set of instructions that tells the computer what to do and how to do it in the language the computer understands—BASIC. A set of instructions to make the computer do what you want it to do, following the rules of BASIC, is called a *program—your* program.

MEMORY

ATARI BASIC is "built-in" to the ATARI 600XL and 800XL computers. The vocabulary (words of ATARI BASIC) and syntax (rules of grammar) are stored in part of the memory of the computer. This memory consists of a group of chips called

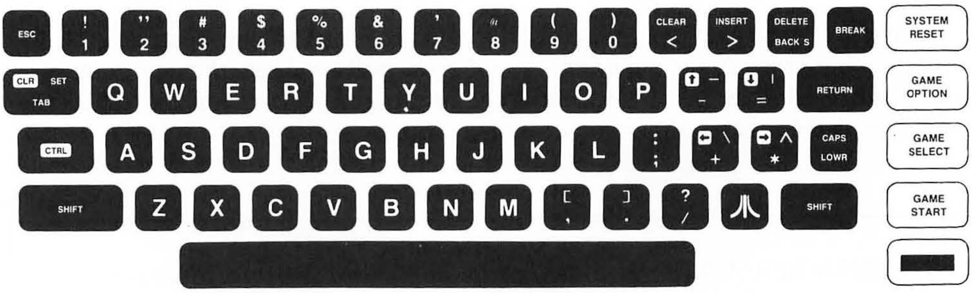
ROM, or *Read Only Memory.* Information on ROM is permanently stored, much like the information in a book or on a phonograph record. The computer can read information from ROM but cannot erase it or change it in any way.

> REMEMBER: ROM is Read Only Memory.

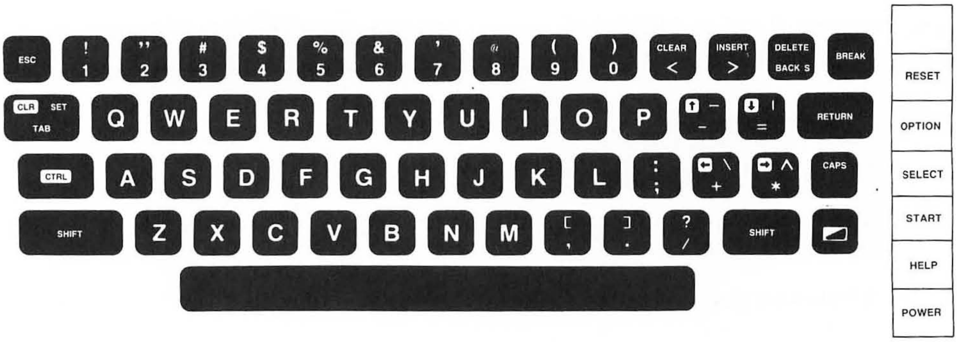
ATARI BASIC is not built-in to the older ATARI 400 and 800 computers. Instead, it is available as a plug-in cartridge. Inside the cartridge you will find ROM chips containing ATARI BASIC stored permanently for your use. When you want to use ATARI BASIC on an ATARI 400 or 800 just plug in the ATARI BASIC cartridge.

THE KEYBOARD

The most obvious feature of the computer is its keyboard. The ATARI 400 or 800 keyboard is shown below:



The ATARI 600XL and 800XL keyboards are somewhat different.



The keyboard is your control center. You will use it to communicate with the computer. Use the keyboard to type information into the computer. As you type, the information that you type is stored in the computer's memory. This type of memory is called RAM, which means *Random Access Memory*, and is different from ROM. *You* can put information into RAM, but not into ROM. You can also erase or change information in RAM. RAM is like a blackboard or scratch pad. Yes, RAM is just more chips. Good grief! Will those chips never stop?

Actually, RAM should have been called Read/Write Memory, or RWM, because information can be read from it, or written into it. Unfortunately, RWM is hard to pronounce.

REMEMBER: ROM is Read Only Memory. Information in ROM is permanently stored and can't be changed. RAM is Random Access Memory. Information in RAM can be erased, changed, or replaced.

What you type on the keyboard is stored in RAM. It also appears on the TV screen so that you can see what you type. As you will soon see, the computer also prints information on the TV screen. Together, the keyboard and the TV screen provide two-way communication with the computer.

REMEMBER: When you type on the keyboard, the information you type is stored in the RAM memory.

The memory (RAM and ROM) of your ATARI computer consists of thousands of *memory locations*. Each memory location can hold (store) one *byte*.

- A byte can be a letter, A to Z.
- A byte can be a decimal digit, 0 to 9.
- A byte can be a punctuation symbol or a special character (+, *, #, %, etc.).
- A byte can be a code for a shape or a color or a sound.

How much memory does your ATARI have? It depends on the computer.

COMPUTER	BYTES OF RAM
400	16,384
800	16,384 to 49,152
600XL	16,384 to 65,536
800XL	65,536

As a shorthand notation, many people might say a computer has 4K bytes or 16K bytes or 32K bytes or 64K bytes. One (1)K bytes equals 1024 bytes.

Here is a handy table to help you translate from computerese to English or vice versa.

COMPUTERESE	ENGLISH
1K	1,024
2K	2,048
4K	4,096
8K	8,192
16K	16,384
32K	32,768
48K	49,152
64K	65,536

> REMEMBER: A byte is a small unit of information such as a letter or a single digit. The memory of an ATARI computer consists of thousands of locations. Each location can store one byte. One K (1K) bytes equals 1024 bytes.

OK, you have now had a brief introduction to the ATARI computer and to the jargon of computers, *computerese*. If you have not already done so, hook up your computer and plunge into the next chapter. If you don't know how to hook up your computer, then:

- read the *ATARI OWNERS GUIDE* that accompanies your computer; or
- yell for help! There are so many ATARI computers in use, someone might hear you.

Before you move on to Chapter 2, you may wish to take the following Self-Test so that you can amaze and delight yourself by how much you have already learned.

SELF-TEST

Answers to the Self-Test questions follow the last question.

1. Inside an ATARI computer, you will find a bunch of "chips." A chip (check the one that applies best):

   (a) can be carried by a strong ant.
   (b) contains thousands of very tiny electronic circuits in a small space.
   (c) can be seen only by mountain sheep (RAMs?) with telescopic eyesight.
   (d) is manufactured by leprechauns using miniature tools.
   (e) requires only a little avocado dip.

2. Thousands of programs exist for ATARI computers. What is a program?

   ____

3. You can buy programs at department stores, discount stores, toy stores, computer stores, software stores, by mail, and probably in other ways. Programs for ATARI computers are available in at least three forms. What are these forms?

   (a) ______________________________
   (b) ______________________________
   (c) ______________________________

4. Suppose your ATARI computer system consists of only an ATARI computer and a TV set. Which kind of programs (cartridge, cassette, diskette) can you use?______________________________

5. Complete each sentence.

   (a) To use programs recorded on tape cassettes, you must have

   ______________.

   (b) To use programs recorded on diskettes, you must have______________.

6. Your ATARI computer has two kinds of memory. What are they called?

   (a) ______________
   (b) ______________

7. Answer the following by writing RAM and/or ROM.

   (a) Which can be erased or changed?____________________
   (b) Which is permanent and can't be changed?____________________
   (c) From which can information be read?____________________
   (d) Which is used to store information from the keyboard?____________________

8. ATARI BASIC is a computer language built-in to the newer ATARI 600XL and 800XL computers. In what form is ATARI BASIC available for the older ATARI 400 and 800?

   ____

9. What is a byte?____

   ____

10. What is a memory location?________________________________

    ____

Answers to Self-Test

1. The most appropriate answer is probably (b) contains thousands of very tiny electronic circuits in a small space. However, if you are a computer pirate using trained ants to pilfer parts, answer (a) might be more appropriate.

2. A program is a set of instructions that tells the computer what to do and how to do it. (It is the program that makes the computer appear smart or dumb, friendly or arrogant, helpful or stand-offish, passive or interactive.)

3. Most commercially available programs are available on (a) cartridges, (b) tape cassettes, or (c) diskettes. Cartridges are the easiest to use, especially if you are a beginner in using computers.

4. Cartridge. All ATARI computers have a slot into which you can plug in program cartridges. You don't have to buy additional equipment to use cartridges. Cartridges are very "fumble-proof" compared to cassettes or diskettes. For the ATARI 400 and 800 computers, ATARI BASIC is available as a cartridge.

5. (a) To use programs recorded on tape cassettes, you must have an ATARI Model 410 or Model 1010 Program Recorder. You use it to get information into the computer's memory (RAM) or to save information already in the memory, on tape. Your aching fingers will testify to the usefulness of devices such as the cassette recorder. Your impatient mind will sometimes grumble at its slowness.
   (b) To use programs recorded on diskettes, you must have an ATARI Model 810 or Model 1050 disk drive. Use it to *quickly* load information into the computer's memory (RAM) or save information already in the memory, on a diskette.
6. (a) RAM (Random Access Memory).
   (b) ROM (Read Only Memory).
7. (a) RAM; (b) ROM: (c) both RAM and ROM; (d) RAM
8. For the ATARI 400 and 800 computers, ATARI BASIC is available as a plug-in cartridge. Guess how ATARI BASIC is stored inside the cartridge. Yup, in ROM!
9. A byte is a small unit of information such as a letter A to Z, a single digit 0 to 9, a punctuation symbol (, . : ; ! ? -), a special symbol (# % etc.), a code for a color or sound or shape, or even a space.
10. A memory location is a place in the memory big enough to store one byte. An ATARI computer has thousands of memory locations, sometimes expressed as a number of K-bytes, where 1K = 1024. For example, the ATARI 400 has 16K locations and the ATARI 800XL has 64K locations.

    Since each memory location can store one byte, we also say that the ATARI 400 has "16K bytes" of memory and the 800XL has 64K bytes of memory.

Chapter Two

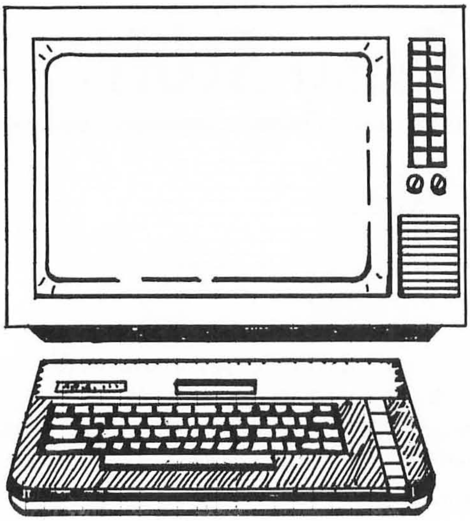
Easy Stuff

Now your fun begins. In this chapter, you will start to learn how to talk to your ATARI by typing instructions called *direct statements* or *commands*. A direct statement (command) tells the computer to do something. The computer does it immediately, then waits for your next instruction.

When you finish this chapter, you will be able to:

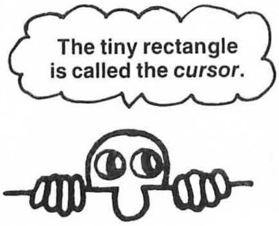
- Clear the TV screen
- Use direct PRINT statements to print information on the screen
- Use SETCOLOR to change the colors on the screen
- Use SOUND to tell the computer to make musical (and non-musical) sounds
- Recognize error messages from the computer (in case you make an error or tell the computer something it doesn't understand)
- Correct typing errors
- Look forward with confidence to the next exciting chapter

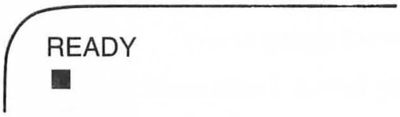
You can begin with an ATARI computer, ATARI BASIC, and a television set.



ATARI BASIC is built-in to the ATARI 600XL and 800XL computers. For the older ATARI 400 and 800 computers, ATARI BASIC is stored in a separate ROM cartridge.

We assume your ATARI is set up and ready to go, with ATARI BASIC built-in or plugged in. Ready? Begin. Turn on the computer and the TV. This is what you should see.





```
READY
■
```

If you don't see this, turn everything off and make sure the computer and the TV are properly connected.

Whenever you see the cursor you know it is your turn to do something, otherwise the computer will simply wait patiently until you are ready to use it. After a while, it will change the screen colors every few seconds to protect your TV from having images burned permanently into the screen.

Questions

1. ATARI BASIC is built-in to the ATARI 600XL and 800XL. How is ATARI BASIC installed in an older model 400 or 800?____________________

____

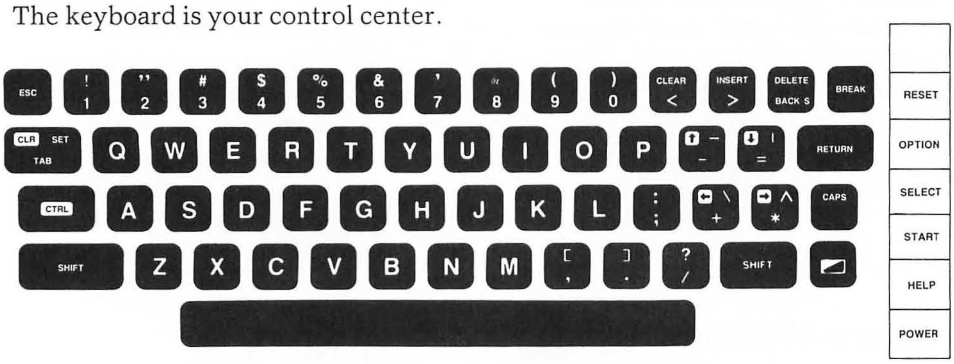
2. When you first turn on the computer, with ATARI BASIC ready-to-go, you see the word READY and a rectangular blob. What is the rectangular blob called?________________________________

Answers

1. ATARI BASIC is available as a ROM cartridge. Plug it into the ROM slot on the ATARI 400 or into the left ROM slot on the ATARI 800.
2. The cursor. When you see the cursor, you know the computer is ready and waiting for instructions from you.

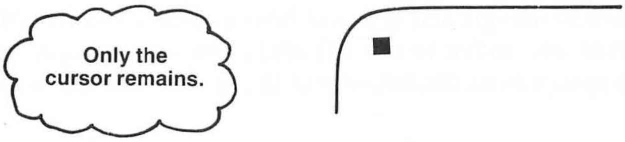
THE KEYBOARD

The keyboard is your control center.



Find the SHIFT keys and the CLEAR < key on your keyboard.

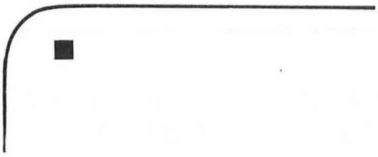
Hold down either SHIFT key and press the CLEAR key.



The screen is blank except for the cursor. Pressing SHIFT and CLEAR simultaneously erases the screen, leaving only the cursor in its *home position*. The home position of the cursor is near the top left corner of the blue part of the TV screen.

Another way to clear the screen and home the cursor is to press and hold down the CTRL key (ATARI 400 or 800) or CONTROL key (ATARI 600XL or 800 XL) and then press the CLEAR key. CTRL is an abbreviation for CONTROL. The CTRL key is located on the left of the keyboard, directly above the SHIFT key. The CTRL key is similar to the SHIFT key in that you use it only in connection with other keys. Nothing happens if you press it without pressing another key at the same time.

When you clear the screen you should see:



Find the SPACE key, the long bar on the bottom of the keyboard:

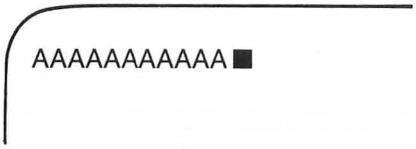
Watch the cursor and press the SPACE key. The cursor will move one space to the right and you will hear a click. Press the SPACE key several times. Each time, you will see the cursor move one space to the right and you will hear a click.

Press the SPACE key and hold it down. The cursor will move and click across the screen. At the right end it will continue on the next line down. When the cursor is about half-way across the screen, release the SPACE key and the cursor will stop.

Find the DELETE BACK S key:

Watch the cursor and press the DELETE key. The cursor will move one space to the left and you will hear a click. Press the DELETE key several times. Each time, the cursor will move to the left and you will hear a click. Hold the DELETE key down and the cursor will move to the left and keep moving until it reaches a position about two spaces from the left edge of the screen. There it will stop.

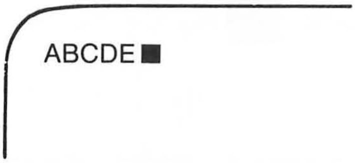
Clear the screen. The cursor is now in its home position. Press the [A] key and hold it down until you see this on the screen:



```
AAAAAAAAAAA■
```

Press the DELETE key and the cursor will move one place left and erase an A. Press the DELETE key again. The cursor will move left and erase another A. Hold down the DELETE key and the cursor will move to the left, erasing all the A's.

Clear the screen, then type [A] [B] [C] [D] [E].

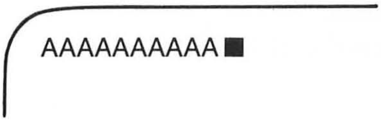


```
ABCDE■
```

DO THIS:	AND SEE THIS:
Press DELETE	ABCD■
Press DELETE	ABC■
Press DELETE	AB■
Press DELETE	A■
Press DELETE	■

REMEMBER: Press the DELETE key to move the cursor one place to the left and erase whatever was there.

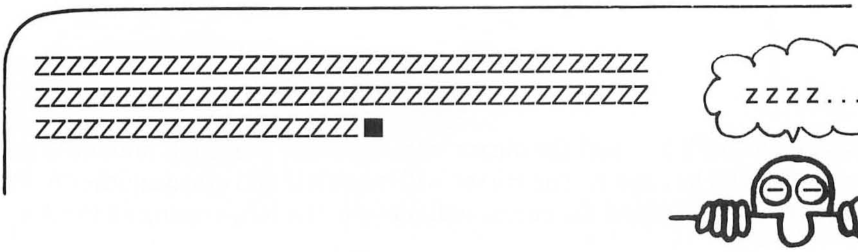
Clear the screen and then hold down the [A] key until you see this on the screen:



```
AAAAAAAAAA■
```

Hold down the SHIFT key and press the DELETE key. Zip! Faster than a blink, the cursor zips to the left and erases the entire line of A's.

Try this. Hold down any letter key until about two and a half lines of that letter are on the screen. For example:



```
ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ
ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ
ZZZZZZZZZZZZZZZZZZZZ■
```

Hold down the SHIFT key and press the DELETE BACK S key. Quick as a wink, all the Z's disappear.

> REMEMBER: Hold down the SHIFT key and press the DELETE key to erase everything to the left of the cursor. In some cases, the erasing will hop up a line and continue from the right to left end. Read on to find out more!

Clear the screen so you see only the cursor in its home position.

Find the RETURN key and press it. The cursor moves down one line.

The cursor moved from here
to here

Hold down the RETURN key. The cursor moves down the screen until it reaches the bottom line where it stops. If you continue to hold the RETURN key down it seems to "bounce" on the bottom line.

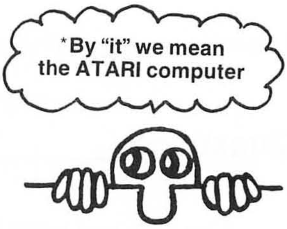
Now type your name and then press the RETURN key. Here is what happened when Karl typed his name.

```
KARL■
```

As Karl typed his name, the cursor moved to the right.

Then Karl pressed the RETURN key:

```
Karl typed this→KARL
It printed this→ERROR-KARL■
                ■
```



*By "it" we mean the ATARI computer

The ERROR message means that the computer did not understand the word KARL. Karl (who is 16 and proud of his name) was quite surprised. How could *his* name be an error?

We explained to Karl that the computer didn't understand him. The word Karl is not one of those special BASIC words that the computer understands. "Aha!" exclaimed Karl, and he, as you will also, began to learn about those special words that the ATARI does understand.

We assume the bottom of the screen still looks as follows— well, perhaps your name is there instead of Karl's name.

```
KARL
ERROR- KARL■
■
```

Press the RETURN key a few times. Each time you do, the information on the screen moves up one line. This is called *scrolling*. Pressing RETURN, if the cursor is on the bottom line, causes all the information on the screen to scroll up one line. Keep pressing RETURN or hold it down and soon everything except the cursor will disappear off the top of the screen.

Find the CAPS LOWR and press it*. Then type [A] [B] [C]

You will see:

```
abc■
```

* On the 600 XL or 800 XL, this key looks like this: CAPS

Aha! Pressing the CAPS/LOWR key lets the computer print both lower case letters and upper case letters (CAPS). To type a lower case letter, just press the letter key. To type an upper case (capital) letter, hold down the SHIFT key and press the letter key. To get back to all caps, hold down the SHIFT key and press the CAPS/LOWR key.

To get caps *only*, press SHIFT and CAPS LOWER together.

To get lower case, press only CAPS LOWER

Questions

1. Describe two ways to clear the screen.

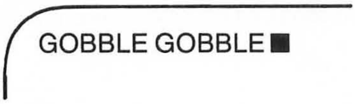
   (a) ______________________________

   (b) ______________________________

2. Where is the home position of the cursor? ______________________________

   ____

3. Some turkey left the screen looking like this:



```
GOBBLE GOBBLE ■
```

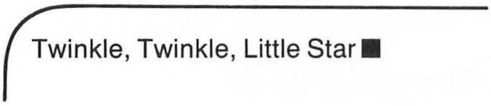
   Describe three ways (without CLEARing the screen) to erase the above turkey message.

   (a) ______________________________

   (b) ______________________________

   (c) ______________________________

4. Describe how to type the following message on the screen.



```
Twinkle, Twinkle, Little Star ■
```

   ____

5. Suppose the above is on the screen and someone presses the RETURN key. What will happen? ______________________________

   ____

____

Answers

1. (a) Hold down the SHIFT key and press the CLEAR < key.
   (b) Hold down the CTRL key (or CONTROL key) and press the CLEAR < key.
2. Near the top left corner of the screen. Actually, it is on the top line of the blue part of the screen, two spaces to the right of the left edge of the blue part of the screen.
3. (a) Press the DELETE key 13 times.
   (b) Hold down the DELETE key until the cursor gobbles up GOBBLE GOBBLE.
   (c) Hold down the SHIFT key and press the DELETE key. Gulp! GOBBLE GOBBLE is gone.
4. First press the CAPS/LOWR key. Then type the message. Use the SHIFT key to type capital letters.
5. The computer will type an ERROR MESSAGE and the cursor will reappear below the E in ERROR. Everything is OK. No damage has been done. The computer is very patient and forgiving. After you make a mistake, it will display the error and then turn on the cursor to let you know it is ready for you to try again. Many other errors are possible. You will meet some of them as you learn ATARI BASIC.

YOUR FIRST BASIC WORD

To avoid misunderstandings with a computer, you must learn its language. In ATARI BASIC, there are special words to tell the computer to do things.

A special BASIC word **PRINT**

PRINT tells the computer to print something on the TV screen. Lucy will demonstrate how it is done.

But first, find the SYSTEM RESET key on the upper right corner of the keyboard.

Press the SYSTEM RESET key and you will see:

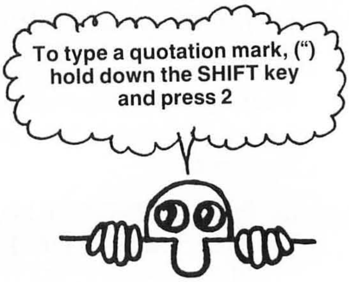
```
READY
■
```

This returns the computer to a state much like it was in when you first turned it on. OK, *now* Lucy will demonstrate. First she cleared the screen.

Lucy typed this:

```
PRINT "LUCY"■
```

Note that LUCY is enclosed in quotation marks.


To type a quotation mark, (") hold down the SHIFT key and press 2

Then she pressed the RETURN key.

Lucy typed this
The ATARI did the rest.

```
PRINT "LUCY"
LUCY

READY
■
```

Try it yourself. Since we don't know your name, try it with Lucy's name.

First, clear the screen.

You type: PRINT "LUCY" and press RETURN

Remember, to type quotation marks ("), hold down the SHIFT key and press the 2 key. Try another one.

You type: PRINT "TAKE A DRAGON TO LUNCH"
It prints: TAKE A DRAGON TO LUNCH

But don't let a dragon take *you* to lunch!

The statement: PRINT "TAKE A DRAGON TO LUNCH" is a direct PRINT statement. It tells the computer to print something on the screen. The computer prints whatever is enclosed in quotation marks following the word PRINT.

In a PRINT statement, a string is enclosed in quotation marks. A string is any bunch of keyboard characters, typed one after another.

A string can be a name: KARL

A string can be a telephone number: 415-323-6117

A string can be a message: TAKE A DRAGON TO LUNCH

A string can be gibberish: AB # J%FD + Z

A string can be almost anything you can type on the keyboard. However, since quotation marks (") mark the beginning and end of a string, they can't be part of the string itself.

> REMEMBER: A direct statement is obeyed by the computer directly after you press RETURN .

Have you made a typing error yet? If you do, ATARI BASIC has a simple way to fix it. Watch while we make a typing error.

We type: PTINT "E.T."
It prints: ERROR- PTINT "E.T."

Since we mispelled PRINT the computer doesn't know what we want. If we had noticed that we hit T when we meant to press R, we could have corrected our mistake by using the DELETE key.

DO THIS	SEE THIS	COMMENTS
Clear screen	■	
Type PT	PT■	Oops!
Press DELETE	P■	Erases T.
Finish typing	PRINT "E.T."■	Looks OK.
Press RETURN	E.T.	AOK!
	READY	
	■	

The DELETE key is great for deleting an error you just made. But suppose you are typing a long line and are almost to the end when, alas, out of the left corner of your left eye, you spot a mistake way back at the beginning.

We type:

```
PRIMT "GARFIELD LOVES LASAGNE"■
```

oops!

You haven't pressed RETURN

If you now press RETURN, you will get an ERROR message. Instead, you can delete the entire line and start over, or you can fix the mistake.

To delete the entire line, hold down SHIFT and press DELETE BACK S. Poof! The line disappears.

Many ATARI BASIC words have abbreviations. There are two ways to abbreviate the PRINT command.

Instead of PRINT, type PR.

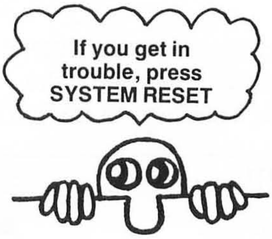
Include this period.

```
PR. "KARL"
PR. "LUCY"
PR. "TAKE A DRAGON TO LUNCH"
PR. "Garfield loves lasagne."
```

Press CAPS LOWER and try this one.

Instead of PRINT, type a question mark.

```
? "KARL"
? "LUCY"
? "TAKE A DRAGON TO LUNCH"
? "Garfield loves lasagne."
```



REMEMBER: Abbreviations for PRINT P R . or SHIFT ?/ together

EXPERIMENT! To learn more about ways of PRINTing on the screen, try these. Avoid clutter — clear the screen before you type each PRINT statement.

(a) PRINT 1,2
(b) PRINT 1;2
(c) PR. 1,2,3
(d) PR. 1;2;3
(e) PR. "GREEN","SLEEVES"
(f) PR. "GREEN";"SLEEVES"
(g) PR. "SLEEVES"," OF ","GREEN"
(h) PR. "SLEEVES";"OF";"GREEN"
(i) PR. "SLEEVES";" OF ";"GREEN"

Questions

1. Complete each of the following by showing what the computer prints.

(a) You type: PRINT "HAPPY BIRTHDAY, MOTHER!"
It prints: ____________________

(b) You type: PR. "C3PO LOVES AN OIL BATH."
It prints: ____________________

(c) You type: ? "Twinkle, Twinkle, Little Star"
It prints: ____________________

2. Complete each PRINT statement so the computer prints as shown.

(a) You type: PRINT____________________
It prints: MY HUMAN UNDERSTANDS ME

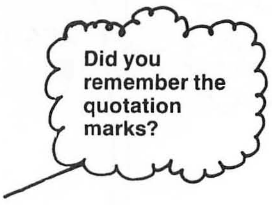
(b) You Type: PRINT____________________
It prints: BURGLARS ARE IN THE HOUSE!

Answers

1. (a) It prints: HAPPY BIRTHDAY, MOTHER!

(b) It prints: C3PO LOVES AN OIL BATH.

(c) It prints: Twinkle, Twinkle, Little Star

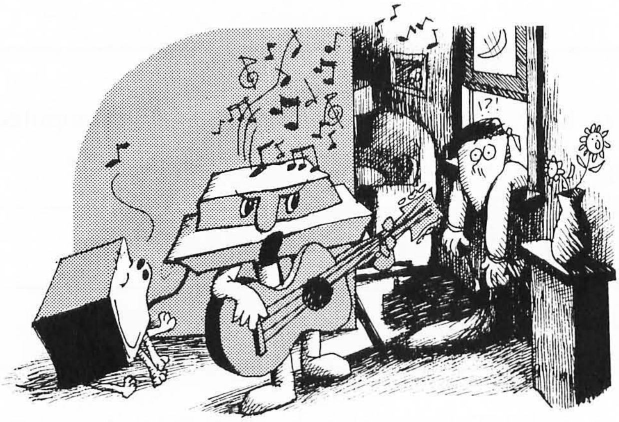


2. (a) Here are three ways to tell the computer.
PRINT "MY HUMAN UNDERSTANDS ME"
PR. "MY HUMAN UNDERSTANDS ME"
? "MY HUMAN UNDERSTANDS ME"

(b)Again, three ways.

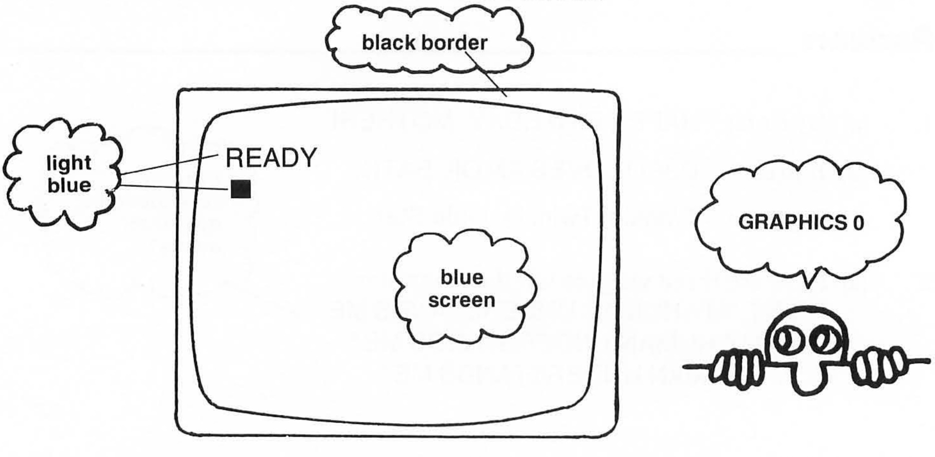
```
PRINT "BURGLARS ARE IN THE HOUSE!"
PR. "BURGLARS ARE IN THE HOUSE!"
? "BURGLARS ARE IN THE HOUSE!"
```

Along with the burglar message, we could also arrange (as you will see later) to have the computer sound an audible alarm. For example, it might play *The Jailhouse Blues* in four part harmony! Hopefully, this might remind the burglar of the possible consequences of crime and thus prevent the attempted theft.



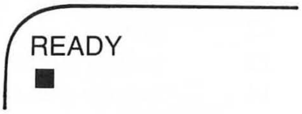
THERE'S A RAINBOW IN YOUR ATARI

When you first turn on your ATARI, it comes up in GRAPHICS 0 mode. Most of the screen is blue. The cursor is light blue, as well as the letters and other characters. There is a black border around the screen.



If you don't press a key for several minutes, the screen colors begin changing every few seconds. This is done to protect your TV from getting certain colors or images burned permanently into the screen. Press almost any key and the blue screen returns.

You can also get GRAPHICS 0 mode by pressing the SYSTEM RESET key. It is at the top right corner of the keyboard. After you press SYSTEM RESET, on the screen you should see this:



```
READY
■
```

Tired of a blue screen? Try another color.

You type: SETCOLOR 2,2,8 and press RETURN

When we did this, the screen color turned to orange. The actual color depends on the settings of the color controls on your TV set (TINT, COLOR, or whatever they are called on your set).

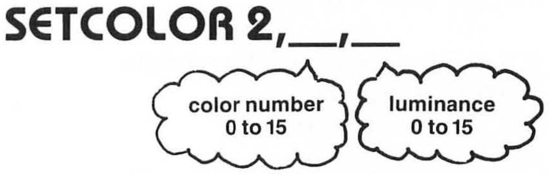
To get back to good old GRAPHICS 0 (blue screen), you can do any of the following.

Press the SYSTEM RESET key.
Type: GRAPHICS 0 and press the RETURN key.
Type: GR. 0 and press the RETURN key. GR. is the abbreviation for GRAPHICS.

Try more colors. To save time, use the abbreviation (SE.) for SETCOLOR.

SE. 2,0,8	grey screen
SE. 2,0,0	black screen ("dark grey")
SE. 2,0,14	light grey screen
SE. 2,12,8	green screen
SE. 2,12,0	dark green screen
SE. 2,12,14	light green screen

The SETCOLOR 2 command lets you set the color and the *luminance* (brightness) of the screen.



Here are the color numbers:

NUMBER	COLOR	NUMBER	COLOR
0	grey	8	light blue
1	gold	9	blue-green
2	orange	10	aqua
3	red-orange	11	green-blue
4	pink	12	green
5	violet	13	yellow-green
6	blue-purple	14	orange-green
7	blue	15	light orange

The luminance number sets the brightness from 0 (darkest) to 14 (lightest). Use even numbers from 0 to 14. Odd numbers are OK, but are recognized as the next lower even number.

REMEMBER: The second number following SETCOLOR selects the color from the above table. The third number determines the luminance from darkest (0 or 1) to medium bright (8 or 9) to lightest (14 or 15). SE. is the abbreviation for SETCOLOR.

Put your ATARI back in GRAPHICS 0 mode. The quickest way to do this is to press the SYSTEM RESET key.

Type: SE. 1,0,0 and press RETURN.

The printing on the screen becomes very dark because you have changed the luminance, or brightness, of whatever is printed on the screen, including the cursor. Try two more.

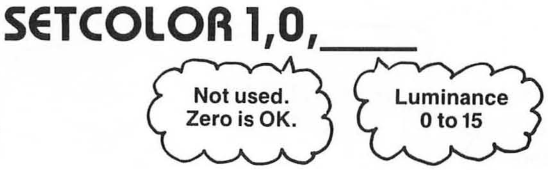
SE. 1,0,14 makes the printing very bright.
SE. 1,0,8 makes the printing medium bright.

This works with any color. Try violet.*

SE. 2,5,8 violet screen, medium bright
SE. 1,0,0 dark printing
SE. 1,0,14 very light printing

*The actual color also depends on the color settings on your TV.

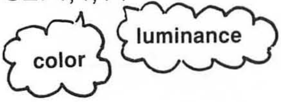
The SETCOLOR 1 command lets you control the luminance of the information printed on the screen.



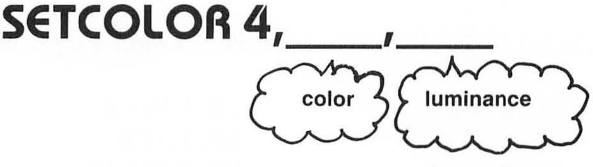
Try various luminances from 0 to 15 with colors of your choice, using SETCOLOR 1 to set your color.

Perhaps you have guessed what comes next. Instead of a black border around the screen, try other colors. Start with a normal blue screen (GRAPHICS 0) and try these.

SE. 4,4,8	pink border
SE. 4,4,0	dark pink border
SE. 4,4,14	light pink border



Use SETCOLOR 4 to control the color and luminance of the border around the screen.

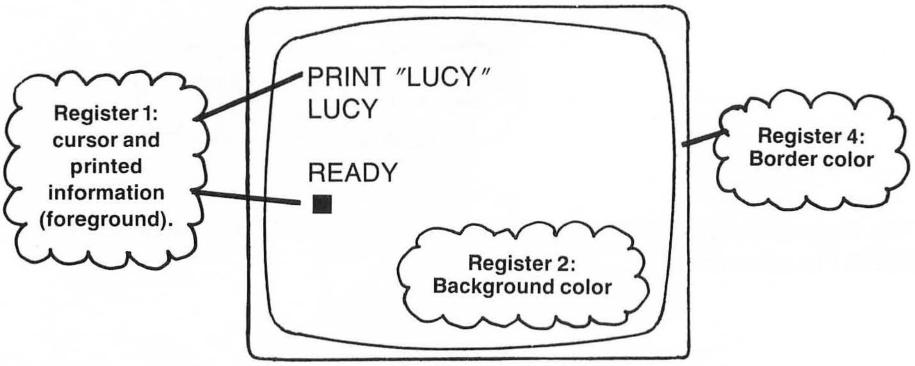


Your ATARI is your best and most patient teacher. What if? Try it and find out. Write down what happens and soon you will have your own book!

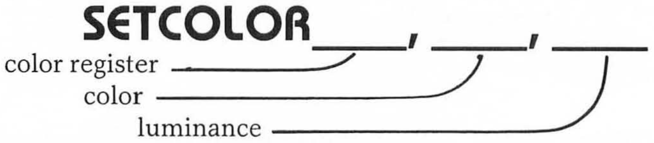
The computer uses a group of *color registers* to control the colors on the screen. In GRAPHICS 0, these registers are numbered 1, 2, and 4.

- Color register 1 controls the brightness, or luminance, of information printed on the screen. This is called the *foreground* of the screen.
- Color register 2 determines the screen's *background* color and luminance.

- Color register 4 controls the color and luminance of the border around the screen.


Register 1: cursor and printed information (foreground).
PRINT "LUCY"
LUCY
READY
■
Register 2: Background color
Register 4: Border color

A SETCOLOR command has three numbers. The first number specifies the color register; the second number specifies the color (except in SE. 1); the third number determines the luminance.



Hmmm...what if? Try it and find out. For example, try some of these.

```
SE. 2,-1,8          SE. 2,256,8
SE. 2,16,8          SE. 2,17,8
SE. 2,255,8         SE. 2,255,16
```

For color or luminance, try any number from 0 to 255.

Questions

1. In GRAPHICS O, what are the colors of the following:

   (a) the border?____________

   (b) the background?____________

   (c) the foreground?____________

2. Write a SETCOLOR command to make the foreground

   (a) very light____________

   (b) very dark____________

   (c) medium bright____________

3. Write a SETCOLOR command to make the background

   (a) aqua, medium bright____________

   (b) red, very dark____________

   (c) blue, very light____________

4. Write a SETCOLOR command to make the border

   (a) grey, very bright____________

   (b) orange, medium bright____________

   (c) black____________

Answers

1. (a) black (b) medium blue (c) light blue
2. (a) SE. 1,0,14 (b) SE. 1,0,0 (c) SE. 1,0,8
3. (a) SE. 2,10,8 (b) SE. 2,3,0 (c) SE. 2,7,14
4. (a) SE. 4,0,14 (b) SE. 4,2,8 (c) SE. 4,0,0 (dark grey)

MUSIC, PLEASE!

Your ATARI adventure has been colorful but quiet. Add some musical accompaniment. Press SYSTEM RESET, then

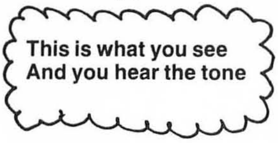
You Type: SOUND 0,121,10,10 and press RETURN.

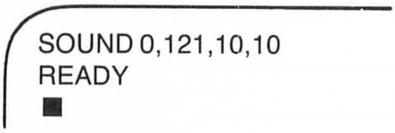
Do you hear it? If not, turn up the volume on your TV. You should hear a tone. It goes on, and on, and on.

Press SYSTEM RESET. The tone stops.

Do it again, exactly this way:

Clear the screen.
Type: SOUND 0,121,10,10 and press RETURN.

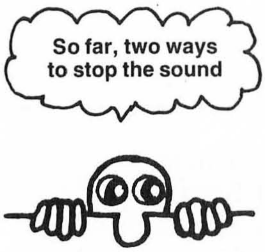


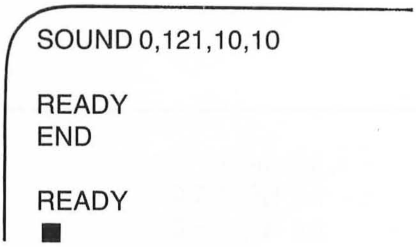


```
SOUND 0,121,10,10
READY
■
```

Type: END and press RETURN

The tone stops and the screen looks like this:





```
SOUND 0,121,10,10

READY
END

READY
■
```

You may turn off the SOUND by pressing the SYSTEM RESET key or by typing END and pressing the RETURN key. Here is still another way to turn off the sound.

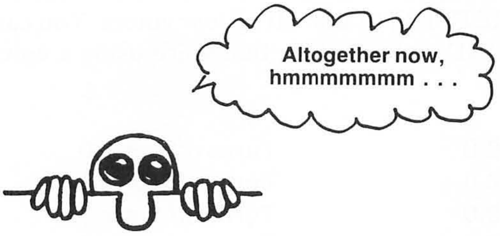
Type: SOUND 0,0,0,0 and press RETURN.

Yes, you can abbreviate SOUND. The abbreviation for SOUND is SO. Use SO. to make some sounds.

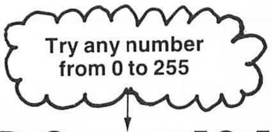
```
SO. 0,108,10,10        SO. 0,96,10,10
SO. 0,121,12,10        SO. 0,121,8,10
SO. 0,121,10,2         SO. 0,121,10,14
```

The ATARI has four *voices*. You have heard voice 0. The others are voices 1, 2, and 3. You can hear all of them at once. The first number following SOUND tells what voice to use. Clear the screen and type the following SOUND statements.

```
SOUND 0,121,10,10     Voice 0
SOUND 1,108,10,10     Voice 1
SOUND 2,91,10,10      Voice 2
SOUND 3,72,10,10      Voice 3
```



Type END to hear the restful sound of silence.
EXPERIMENT:



SOUND 0,____,10,10

The second number following SOUND controls the *pitch*, or *frequency*, of the tone. High numbers give low tones; low numbers give high tones.

Hmmm...wonder what note on the musical scale tone number 121 is? Here is a hint.

See Appendix C for a table of frequency numbers corresponding to musical notes.

The third number controls the *distortion* of the sound. The numbers 10 and 14 give a pure tone. For distortion, use only even numbers: 0,2,4,6,8,10,12,14. Of course, if you wish, try an odd number to see what happens.

The fourth number controls the loudness between 0 (silent) and 15 (loudest).

EXPERIMENT: Try *your* numbers.

SOUND ______,______,______,______

↑	↑	↑	↑
Voice	**Frequency**	**Distortion**	**Volume**
0 to 3	**0 to 255**	**0 to 14**	**0 to 15**

We have told you three ways to turn off the SOUND. Press SYSTEM RESET or type END (and press RETURN) to shut off *all four* voices. You can shut off just one voice by typing a SOUND statement for that voice using a volume of 0. We will usually do it this way:

SO. 0,0,0,0	Turns off voice 0.
SO. 1,0,0,0	Turns off voice 1.
SO. 2,0,0,0	Turns off voice 2.
SO. 3,0,0,0	Turns off voice 3.

Questions

1. Write a SOUND command using voice 2, frequency 91, distortion 10, and loudness 8.______________________________

2. The frequency can be any number from ____ to ____.

3. The distortion can be any of these numbers:________

4. The loudness can be any number from ____ to ____.

5. Describe two ways to turn off all four voices.

   (a) ______________________________

   (b) ______________________________

6. How do you turn off only voice 3?______________________________

Answers

1. SOUND 2,91,10,8 or SO. 2,91,10,8
2. 0 to 255
3. 0,2,4,6,8,10,12,14
4. 0 to 15
5. (a) Type END and press RETURN.

   (b) Press the SYSTEM RESET key.
6. Type SO. 3,0,0,0 and press RETURN.

SELF-TEST

Before zooming on to Chapter 3, dally for awhile in this self-help Self-Test.

1. What is the cursor?____________________

   ____

2. How do you clear the screen?____________________

   ____

3. What happens when you press the DELETE BACK S key?

   ____

4. Suppose you type DO THE HOMEWORK ON PAGE 157 and press the RETURN key? What does the ATARI do?

   ____

5. Suppose you type PRINT "DO THE HOMEWORK ON PAGE 157" and press RETURN. What does the ATARI do?

   ____

6. When you first turn on your ATARI, with ATARI BASIC ready to use, the computer comes up in GRAPHICS 0 mode. What does the screen look like?

   ____

7. How can you put the computer in GRAPHICS 0 mode?____________________

8. In GRAPHICS 0, three color registers control the colors on the screen. Complete the table showing what each color register controls.

COLOR REGISTER	CONTROLS
1	
2	
4	

9. In the command: **SETCOLOR 1,____,____**

↑ 2nd number ↑ 3rd number

(a) What does the 2nd number tell the computer?

(b) What does the 3rd number tell the computer?

10. In the command: **SETCOLOR 2,____,____**

↑ 2nd number ↑ 3rd number

(a) What does the 2nd number tell the computer?

(b) What does the 3rd number tell the computer?

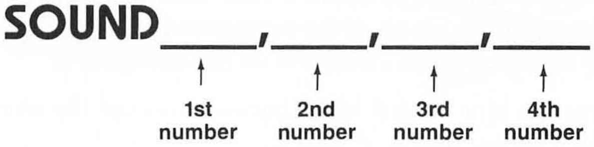
11. In the command: **SETCOLOR 4,____,____**

↑ 2nd number ↑ 3rd number

(a) What does the 2nd number tell the computer?

(b) What does the 3rd number tell the computer?

12. A SOUND command has four numbers following the word SOUND.



What does each number control and what is the range of legal values for each number?

(a) 1st number: ____________________

(b) 2nd number: ____________________

(c) 3rd number: ____________________

(d) 4th number: ____________________

13. Describe two ways to turn off all voices.

(a) ____________________

(b) ____________________

14. How do you turn off only voice 1?

____

Answers to Self-Test

1. The cursor is a small rectangle. It marks the place where information will appear on the screen as you type it on the keyboard. When you see the cursor, you know it is your turn to do something.

2. Here are two ways to clear the screen.
   (a) Hold down the SHIFT key and press the CLEAR key.
   (b) Hold down the CTRL key and press the CLEAR key.

   Either method erases everything from the screen except the cursor.

3. The cursor moves back (to the left) one place and erases any character in that position.

4. The ATARI prints an ERROR message on the screen. What's this? You are trying to get the computer to do your homework. Computers know that HOMEWORK (such as vacuuming, mopping floors, cleaning one's room, washing dishes, and so on) is for humans, not computers.

5. The ATARI prints: DO THE HOMEWORK ON PAGE 157. Ha! Of course. The computer tells *you* to do the homework...because you told it to tell you to do the homework...oh-oh, if the computer doesn't do the homework and you don't do the homework...who will do the homework?

6. The screen is blue with a black border. You see the word READY and the cursor in a lighter blue.

7. The easiest way is to press the SYSTEM RESET key. If you see the cursor alone on a line, you can also type GRAPHICS 0 or GR. 0 and press the RETURN key.

8.

COLOR REGISTER	CONTROLS
1	Foreground (printed information) color and luminance
2	Background color and luminance
4	Border color and luminance

9. (a) Nothing. The number is not used with color register 1. However, even though the number is not used, it must have a value in the range 0 to 255. We usually set it to zero (0).

   (b) The luminance of the cursor and printed characters. The computer recognizes *even* numbers from 0 to 14. Use 0 to get the darkest characters and 14 for the brightest. Odd numbers are OK, the computer simply uses the next smaller even number (0 for 1, 2 for 3, and so on).

10. (a) Background color. There are 16 colors, usually numbered 0 to 15. It is OK to use any number from 0 to 255, since the ATARI will reduce a number larger than 15 to a number in the range 0 to 15.

    (b) Luminance of the background. Use even numbers from 0 to 14. Zero (0) is darkest, 14 is brightest.

11. (a) Border color, numbered 0 to 15. It is OK to use any number from 0 to 255 since the ATARI will reduce a number larger than 15 to a number in the range 0 to 15.

    (b) Luminance of the border. Use even numbers from 0 to 14. Zero (0) is darkest, 14 is brightest.

12. (a) The first controls the *voice* to be heard. There are four voices, numbered 0, 1, 2, or 3.

    (b) The second number controls the *pitch*, or *frequency*, of the sound. It can be any number from 0 to 255. Low numbers make high sounds and high numbers make low sounds.

    (c) The third number controls the *distortion*, or *quality*, of the sound. Use even numbers from 0 to 14.

    (d) The fourth number controls the *volume*, or *loudness*, of the sound. Use numbers from 0 to 15. Zero (0) is inaudible – use 1 to get the quietest audible sound. Fifteen (15) produces the loudest sound. Of course, it also depends on how you set the volume on your TV set.

13. (a) Press SYSTEM RESET.

    (b) Type END and press RETURN.

14. Type SOUND 1,0,0,0 and press RETURN, OR
    type SO. 1,0,0,0 and press RETURN.

Now take a break. Do something relaxing, then go on to Chapter 3.

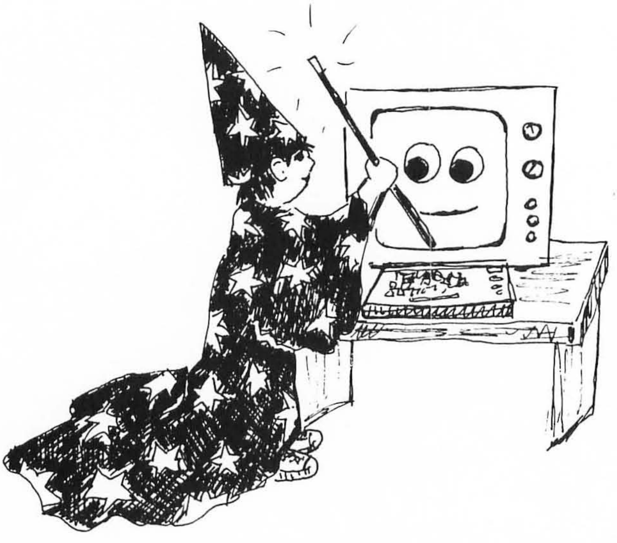
Chapter Three

Basic Programs

In this chapter, you will learn to read, understand, and use simple ATARI BASIC programs. These programs will include statements you already know how to use (PRINT, SETCOLOR, SOUND) and new statements (GOTO, REMARK, FOR, NEXT).

When you finish this chapter, you will be able to:

- Read and understand short programs that include PRINT, SETCOLOR, SOUND, GOTO, REMARK, FOR, and NEXT statements
- Use NEW to erase any old, unwanted programs from the computer's memory (RAM)
- Enter a new program into the computer's memory (RAM)

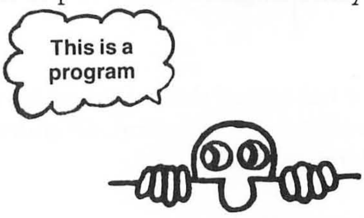


- Tell the computer to LIST a program on the TV screen
- Tell the computer to RUN (obey, execute, carry out) a program in its memory (RAM)
- Write simple programs using PRINT, SETCOLOR, SOUND, GOTO, REMARK, FOR, and NEXT statements
- Use graphics characters to put patterns on the screen
- Edit, correct, and delete statements in a program.

VERY SMALL PROGRAMS

Now you will learn how to enter a *program* into the ATARI's memory. The following program causes the computer to put Karl's name on every line of the screen.

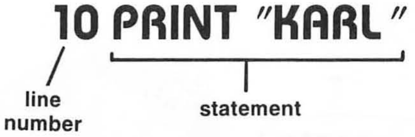
```
10 PRINT "KARL"
20 GOTO 10
```



The program consists of two *lines*.

This is a line: `10 PRINT "KARL"`
This is a line: `20 GOTO 10`

Each line consists of a *line number* followed by a *statement*. For instance:



When you type a statement that begins with a line number, the statement is *not* executed immediately after you press the RETURN key (as in Chapter 2). Instead, the statement is stored in the computer's memory for later execution.

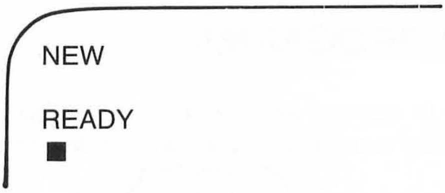
Line numbers tell the computer the order in which to follow statements in the program. Line numbers don't have to be consecutive integers such as 1, 2, 3, 4, 5, and so on. Instead it is better to number by tens as we do in the above program. Then, if you wish, you can easily insert or add more lines between ones you already have. For example, in the above program the line numbers are 10 and 20 allowing you to add nine more lines between them (lines 11, 12, 13, 14, 15, 16, 17, 18, and 19).

Of course, you don't *have* to number by tens. If you prefer numbering by thirteens or fives or jumping around, do it!

Before you enter a program, you must first remove or erase any programs that may already be stored in the memory (in RAM, that is). If you don't do this, the new program may become intertwined with an old program, resulting in confusion!

Here is how to erase old programs and get the computer ready to accept a new one. First, CLEAR the screen. Then:

You type: NEW and press the RETURN key.
You should see this on the screen:

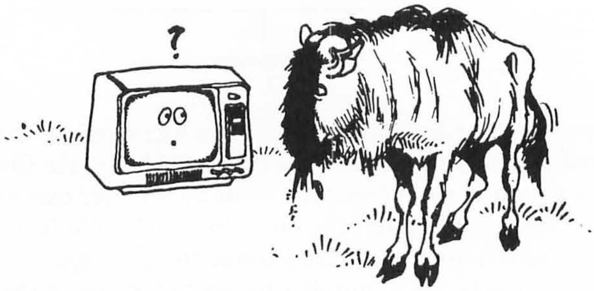


```
NEW

READY
■
```

The computer has erased the portion of its memory that stores BASIC programs. It is ready to accept a new program. Of course, if you misspell NEW, the computer may not understand you.

We type: GNU
It prints: ERROR- GNU '

Apparently the computer doesn't appreciate puns.



Now you are ready to enter the two-line program to put Karl's name on the screen. Here it is again:

```
10 PRINT"KARL"
20 GOTO10
```

Yes, you may use PR. or $ instead of PRINT.

You are ready to enter the program. Do this:

(1) Clear the screen.

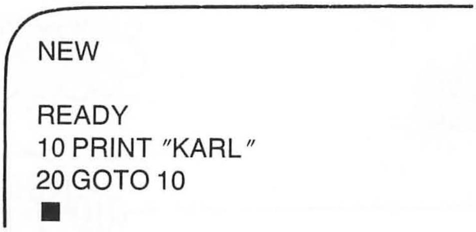
(2) Type: NEW and press RETURN .

(1) Type: 10 PRINT "KARL" and press RETURN .

(2) Type: 20 GOTO 10 and press RETURN .

If you make a typing error, correct it by using the DELETE key. If you have already pressed RETURN, simply retype the line correctly, including the line number. The program is now stored in the computer. The computer patiently awaits your next instruction.

If you did all the above without any typing errors, you will see this on the screen:

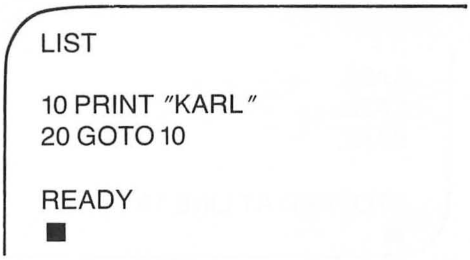


```
NEW

READY
10 PRINT "KARL"
20 GOTO 10
■
```

If it is not exactly the same, try again from the beginning. Soon, we will show some neat ways to make corrections. Is the program really stored in the computer's memory? Find out. First clear the screen. Then:

Type: LIST and press RETURN.

The screen should read:



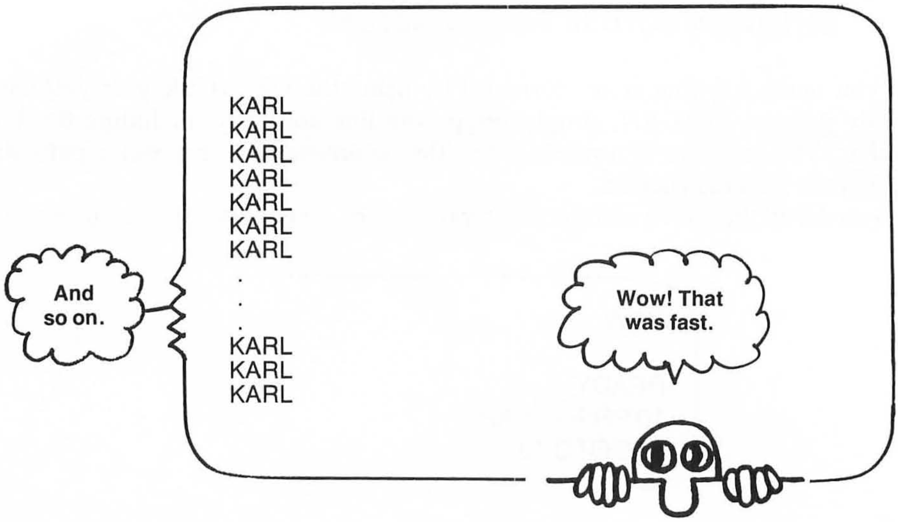
```
LIST

10 PRINT "KARL"
20 GOTO 10

READY
■
```

LIST tells the computer to print all of the statements in the program stored in the computer's memory on the screen. After listing the program, the computer prints READY and turns on the cursor to let you know it is your turn again. Yes, LIST has an abbreviation. Type L. instead of LIST.

Finally! The program is stored in memory (RAM, of course) and ready to go. It's time to RUN the program.

You type: RUN and press RETURN.

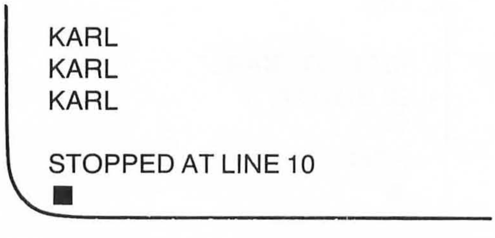
Quick as a wink, Karl's name appears on every line and you will see this on the screen:



```
KARL
KARL
KARL
KARL
KARL
KARL
KARL
.
.
.
KARL
KARL
KARL
```

Nothing more seems to happen. Actually, the computer keeps printing KARL on the bottom line. Each time it does, all the other KARLs are pushed up one place; the top KARL is "pushed off" the top of the screen. This happens so quickly, however, that only superheroes with ultrafast eyes can see it happen.

How do you stop the computer? Easy. Find the BREAK key. Press it. Congratulations! You have stopped the computer. The bottom part of the screen probably looks like this:



```
KARL
KARL
KARL

STOPPED AT LINE 10
■
```

To stop the computer, press the BREAK key.

How does the program work?

```
This line: 10 PRINT "KARL"
```

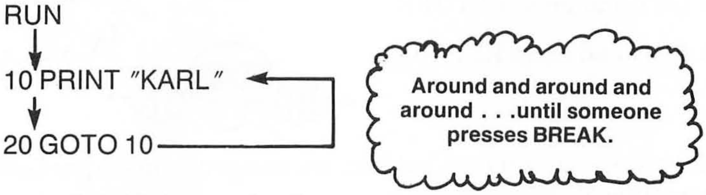
tells the computer to print the string KARL on the screen.

```
This line: 20 GOTO 10
```

tells the computer to go to line 10 and continue.

When you type RUN and press RETURN, the computer starts *executing the program,* beginning at the *smallest line number.* The computer does line 10, then line 20, then line 10, then line 20, then line 10, and so on and so on and so on...until someone presses the BREAK key.

Follow the arrows:


RUN
10 PRINT "KARL"
20 GOTO 10
Around and around and around . . .until someone presses BREAK.

The BASIC word GOTO can also be spelled GO TO or GO. (including the period). Try this: Enter each of the following programs and LIST it.

```
10 PR. "KARL"          10 ? "KARL"
20 GO. 10              20 GO TO 10
```

If you use abbreviations in a program, then LIST the program, the computer prints the full unabbreviated word, PRINT instead of PR. or ?, and GOTO instead of GO.

Questions

1. What happens when you type a direct statement or command, without a line number?______________________________

2. What happens when you type a line that begins with a line number?________

3. Before entering a BASIC program, you usually type NEW and press the RETURN key. Why?______________________________

4. How do you tell the computer to print on the screen a listing of a program stored in its memory?______________________________

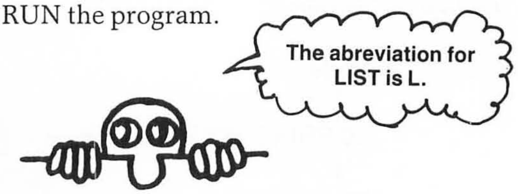
____

5. How do you tell the computer to execute (do or obey) a program stored in its memory?

6. Write a short program to put *your* name on every line of the screen.

Answers

1. The computer obeys your command immediately.
2. The computer stores the line for later use.
3. This erases, removes, or deletes any old program that might be in the computer's memory. If you don't do this, lines from an old program might get mixed up with lines from your new program, thus causing mysterious and unpredictable results when you RUN the program.
4. Type LIST and press RETURN.
5. Type RUN and press RETURN.



6. We don't know your name, but here are two programs to put Garfield's name on every line of the screen.

```
10 PR. "GARFIELD"              10 ? "Garfield"
20 GO. 10                      20 GO TO 10
```

And Jim Rudd did it this way:

```
10 PR. "Jim is GREAT!"
20 GO. 10
```

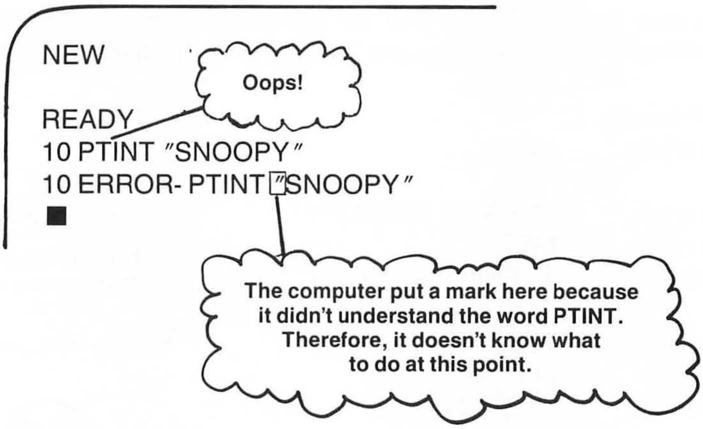
OOPS!

Here is a tiny program we think you will understand.

```
10 PRINT "SNOOPY"
20 GOTO 10
```

Suppose, while typing line 10, you misspell PRINT, but don't notice the mistake and press RETURN.

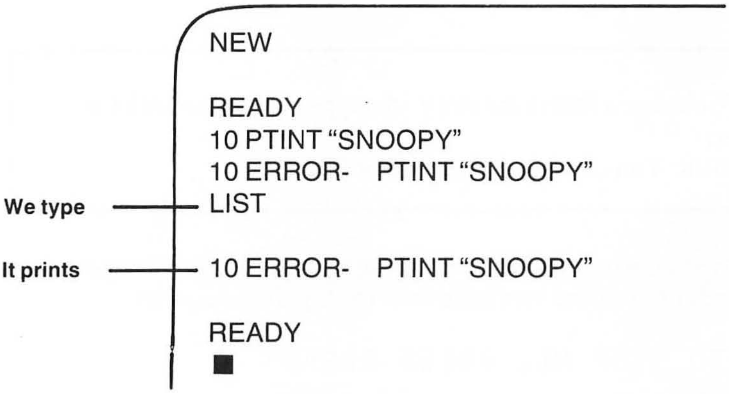
As an example, we cleared the screen, then typed NEW to erase any old programs. Then we mistyped line 10 and pressed RETURN.



```
NEW

READY
10 PTINT "SNOOPY"
10 ERROR- PTINT"SNOOPY"
■
```

Oops!
The computer put a mark here because it didn't understand the word PTINT. Therefore, it doesn't know what to do at this point.

Since we pressed RETURN, the line is stored incorrectly in the memory. To show this, we LIST the line:



```
NEW

READY
10 PTINT "SNOOPY"
10 ERROR-   PTINT "SNOOPY"
LIST

10 ERROR-   PTINT "SNOOPY"

READY
■
```

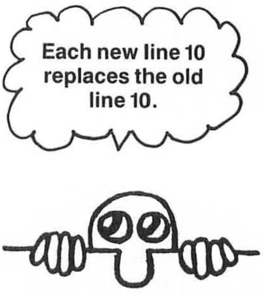
Yes, line 10 is incorrectly stored in memory. Here is one way to fix it: Simply retype the line, including the line number. The new line 10 will replace the old line 10.

Type: 10 PRINT "SNOOPY" and press RETURN.

If you now type LIST, you will see the most recent line 10. The new line 10 replaces the old line 10. To convince yourself that this happens, do the following.

Clear the screen.

You type:	NEW
You type:	10 PRINT "SNOOPY"
You type:	LIST
It prints:	10 PRINT "SNOOPY"
You type:	10 PRINT "LUCY"
You type:	LIST
It prints:	10 PRINT "LUCY"
You type:	10 PRINT "CHARLIE"
You type:	L.
It prints:	10 PRINT "CHARLIE"

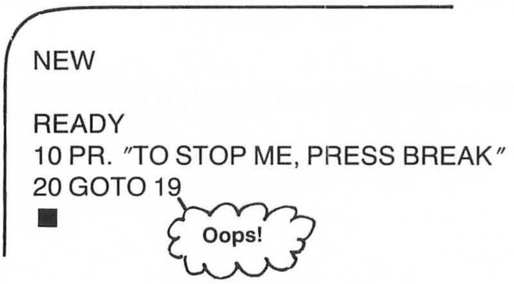


REMEMBER: To change a line in memory, just type a new line with the *same line number*.
ALSO REMEMBER: You can type L. instead of LIST.

There are many ways to make mistakes in entering a program. Here is a program we want to enter in order to remind someone how to stop the computer.

```
10 PR. "TO STOP ME, PRESS BREAK"
20 GOTO 10
```

Unfortunately, we entered it as shown on the screen:



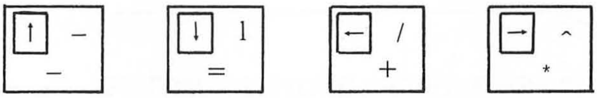
```
NEW

READY
10 PR. "TO STOP ME, PRESS BREAK"
20 GOTO 19
■
```

Well, you might expect some problems in running this program. The GOTO has no place to go – there is no line 19. Here is what happened when we ran the program.

```
RUN
TO STOP ME, PRESS BREAK

ERROR-   12 AT LINE 20
■
```

ERROR #12 is a "line not found" error. The computer could not find line 19 to go to. You will probably encounter this and many other errors as you work and play through this book. Nevermind – they are easy to fix. For a complete list of errors, visit Appendix G.

We could fix line 20 by retyping it correctly. Instead, let's do it a different way. Find these keys:



You can use these keys, along with the CTRL * key, to move the cursor around the screen.

* On the 600XL & 800XL, use the CONTROL key.

Hold down CTRL and press ↑ to move the cursor up the screen.

Hold down CTRL and press ↓ to move the cursor down the screen.

Hold down CTRL and press ← to move the cursor to the left.

Hold down CTRL and press → to move the cursor to the right.

Go ahead and try the keys. Move the cursor around. Perhaps you will notice that you can move the cursor past letters on the screen without changing them.

Clear the screen and LIST the program:

```
LIST
10 PRINT "TO STOP ME, PRESS BREAK"
20 GOTO 19

READY
■
```

Move the cursor up and over until it covers the 9 in 19:

```
20 GOTO 19
```

cursor

Type a zero (0) to correct the error.

This is what you see:

```
20 GOTO 10■
```

cursor

One more thing. Press the RETURN key. The computer stores the corrected line 20 in its memory. To see that this is true, clear the screen and LIST the program.

Now suppose we have just entered the following program:

```
10 PRINT "HAPPY BIRTHDAY, MOTER"
20 GOTO 10
■
```

Oops! We meant MOTHER.

Easy to fix. First, move the cursor up and over so it covers the E in MOTER

```
10 PRINT "HAPPY BIRTHDAY, MOTER"
```

cursor

Hold down the CTRL key and press INSERT >

```
10 PRINT "HAPPY BIRTHDAY, MOT■ER"
```

The ER moved to the right, leaving a space under the cursor. Type the letter H and press RETURN.

OK!

```
10 PRINT "HAPPY BIRTHDAY, MOTHER"
20 GOTO 10
```

Press the RETURN key again to move the cursor down to the first available open space, then LIST or RUN the program to see if it is OK. Enjoy your birthday, Mother!

Suppose the error was:

```
10 PRINT "MERRY CHRISTMAS, FAATHER"
```

OOPS!

Position the cursor under the second A in FAATHER.

10 PRINT "MERRY CHRISTMAS, FA$\boxed{A}$THER"

Hold down CTRL and press the DELETE key. The A under the cursor is erased and THER moves left one place. Everything is AOK – press RETURN.

Now you know several ways to make corrections. EXPERIMENT! Try some of these little programs. When you type one of these programs, be sure to include the comma (,) or the semicolon (;) at the right end of the PRINT statement.

```
(1) 10 PRINT "LUCY",
    20 GOTO 10

(2) 10 PRINT "LUCY";
    20 GOTO 10

(3) 10 PRINT "LUCY ";
    20 GOTO 10
```

Edit program (1) to get program (2) and edit program (2) to get program (3).

Put one space here.

The comma (,) at the end causes a "tab" to the right of several spaces. The semicolon (;) causes things to be printed close together. Experiment: PRINT *your* names or messages. Try both commas and semicolons at the end of the PRINT statement.

Questions

1. Which program would produce the RUN shown below? ________

PROGRAM A

```
20 PRINT "HA HA ";
30 GOTO 20
```

space

PROGRAM B

```
20 PRINT "HA HA "
30 GOTO 20
```

space

```
HA HA HA HA HA HA HA HA HA HA HA HA
HA HA HA HA HA HA HA HA HA HA HA HA
HA HA HA HA
```

and so on.

2. Teach the computer to cry. Write a program to fill the screen with BOO HOO.

   ____

   ____

3. The following program is stored in memory.

```
10 PRINT "JACK LOVES JILK"
20 GOTO 10
```

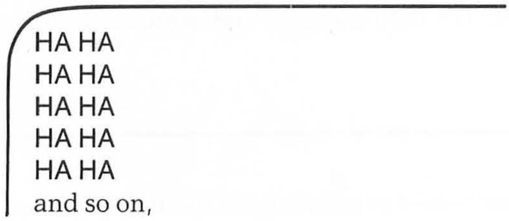
   Oops! That should say JACK LOVES JILL. Describe two ways to correct the program.

   (a) ____

   (b) ____

Answers

1. Program A produces the computer hilarity shown. The difference in the programs is the semicolon at the end of the PRINT statement in line 20 of program A. Program B produces a more subdued laughing computer, as shown below.



```
HA HA
HA HA
HA HA
HA HA
HA HA
```

and so on,

2. We did it this way:

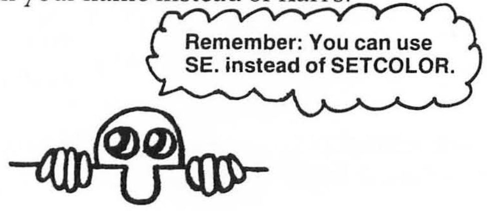
```
10 PRINT "BOO HOO ";
20 GOTO 10
```

3. The error is in line 10. Line 20 is OK.
   (a) Retype line 10: 10 PRINT "JACK LOVES JILL" and press RETURN.
   (b) Position the cursor over the K in JILK, type the correct letter (L), and press RETURN.

BRING OUT THE RAINBOW

You can put your name or message on colors of your choice, instead of good old GRAPHICS 0 blue. Try any of these with your name instead of Karl's.

```
10 SETCOLOR 1,0,0
20 SETCOLOR 2,4,8
30 SETCOLOR 4,12,8
40 PRINT "KARL"
50 GOTO 40
```



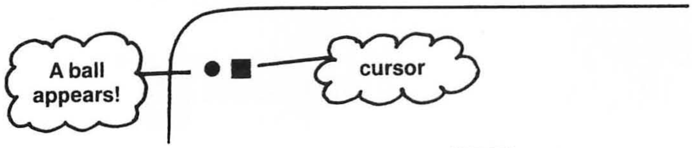
This results in dark letters on a pink screen with a green border. Change the colors: LIST the program, move the cursor to the line you want to change, and change it.

Next, try a flickering screen, flickering between two colors.

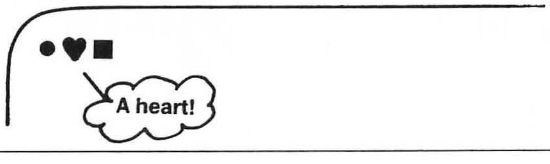
```
10 SE. 1,0,0              Dark letters
20 SE. 4,0,0              Black border

30 SE. 2,5,8              Medium pink
40 PR. "LUCY   ";

50 SE. 2,12,8             Medium green
60 PR. "LUCY   ";

70 GOTO 30
```

Experiment with different colors and luminances to get the kind of flicker you want.

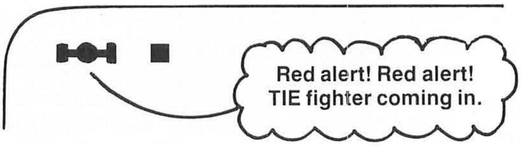
Instead of names and other messages, let's make some patterns on the screen. Your ATARI computer has a set of *graphics characters*. First, clear the screen. Hold down CTRL and press T .



Next, hold down the CTRL key and press [,



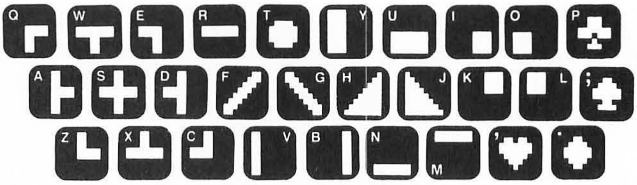
Clear the screen, then hold down the CTRL key and press A T D and a couple of spaces.



EXPERIMENT! To see the graphics characters, hold down the CTRL key and press any of these keys:

Q W E R T Y U I O P
A S D F G H J K L ;
Z X C V B N M , .

How do you remember where all those graphics characters are located on the keyboard? Relax, help is here. Consult the following keyboard chart.



Let's make some patterns on the screen. First, a screen full of hearts:

```
10 PRINT "♥";
20 GOTO 10
```

To get a heart, hold down CTRL and press ♥.

We think you will enjoy this one:

```
10 PRINT "╱╲";
20 GOTO 10
```

Hold down CTRL, press ╱ and ╲.

Try these. Change only line 10.

```
10 PRINT "▲";         CTRL H and J
10 PRINT "+";         CTRL S
10 PRINT "╱‾╲_";      CTRL F M G N
```

Of course, experiment. Make up your own patterns. Try some other colors. Use SETCOLOR 1 to make the graphics characters lighter or darker. Use SETCOLOR 2 to change the background color. Use SETCOLOR 4 to change the border color. Or, include the colors of your choice as part of the program.

```
10 SE. 1,0,10
20 SE. 2,4,4
30 SE. 4,7,8
40 PR. "♥";
50 GOTO 40
```

Try different colors and luminances in lines 10, 20, and 30.

> REMEMBER: The quickest way to change the program is to LIST it, move the cursor to the line you want to change, make the change, then press RETURN.

Questions

1. Below are two of the programs in this section. Draw arrows showing the order in which the computer obeys lines.

```
RUN                              RUN

10 SETCOLOR 1,0,0                10 SE. 1,0,0
20 SETCOLOR 2,4,8                20 SE. 4,0,0
30 SETCOLOR 4,12,8               30 SE. 2,5,8
40 PRINT "KARL   ";              40 PR. "LUCY   ";
50 GOTO 40                       50 SE. 2,12,8

                                 60 PR. "LUCY   ";

                                 70 GOTO 30
```

2. Write a program to put a "forest" on the screen. Use the "tree" on the same key as the semicolon (some people think of it as the "spade" symbol in card playing games). Also put two points (.) after the tree. When we ran our program, we saw the following on the screen:

   ♠..♠..♠.. and so on

   Of course, make the scene green!

Answers

1.

```
RUN
 ↓
10 SETCOLOR 1,0,0
 ↓
20 SETCOLOR 2,4,8
 ↓
30 SETCOLOR 4,12,8
 ↓
40 PRINT "KARL ";◄┐
 ↓                │
50 GOTO 40 ───────┘
```

```
RUN
 ↓
10 SE. 1,0,0
 ↓
20 SE, 4,0,0
 ↓
30 SE. 2,5,8 ◄──┐
 ↓              │
40 PR."LUCY ";  │
 ↓              │
50 SE. 2,12,8   │
 ↓              │
60 PR. "LUCY "; │
 ↓              │
70 GOTO 30 ─────┘
```

2.

```
10 SE. 1,0,8
20 SE. 2,12,12
30 PR. "♠.."; 
40 GOTO 30
```

Medium green trees
Light green screen.

To get the tree (♠), hold down CTRL and press :;

SOUND EFFECTS

Let's put some musical tones, noise, or sound effects into our programs. First, try this one:

```
10 SOUND 0,91,10,10       First tone
20 SOUND 0,121,10,10      Second tone
30 GOTO 10
```

You can use SO. instead of SOUND.

RUN it. The computer plays the first tone, then the second tone, then the first tone, then the second tone, then the first tone, and so on until you press BREAK. However, unless you have supersharp superhero ears, you can't really hear the two tones distinctly – they run together. When you do press BREAK, one tone may continue. Press SYSTEM RESET to shut it off.

We need a way to slow down the computer. We need a *time delay* after each sound. Here is another program, featuring two time delays:

```
10 SOUND 0,91,10,10

20 FOR Z=1 TO 100  ] Time delay
30 NEXT Z

40 SOUND 0,121,10,10

50 FOR Z=1 TO 100  ] Time delay
60 NEXT Z

70 GOTO 10
```

In lines 20 and 30, you see a FOR-NEXT loop.

```
20 FOR Z=1 TO 100
30 NEXT Z
```

This is a FOR-NEXT loop.

This FOR-NEXT loop causes the computer to count from 1 to 100, *very* fast! However, it still takes a little time. While the computer is counting from 1 to 100, you hear tone #1. Then the computer goes on to line 40, turns on tone #2, and continues on to lines 50 & 60 – another FOR-NEXT loop.

```
50 FOR Z=1 TO 100
60 NEXT Z
```

Another FOR-NEXT loop.

The FOR statement tells the computer how far to count.

FOR Z = 1 TO 100

Count from here **to here**

Enter and RUN the program. You will hear the two tones repeated again and again and again, and so on until you press BREAK. Hmmm...is it a fire truck, ambulance, or police car rushing to the scene?

Change the numbers in the SOUND statements to get other tones, distortions, or volumes. Remember:

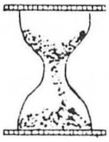
SOUND ____, ____, ____, ____

voice	pitch	distortion	volume
(0 to 3)	(0 to 255)	(0 to 15)	(0 to 15)

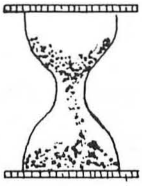
Save typing time: Use SO. instead of SOUND.

Also change the FOR statement to get longer or shorter time delays. For example:

Shorter time delay: FOR Z=1 TO 50



Longer time delay: FOR Z=1 TO 200



Try shorter and shorter delays. For example, try FOR Z=1 TO 10 or even FOR Z=1 TO 1. Also try different delays following the two tones. For example, FOR Z=1 TO 10 for one tone and FOR Z=1 TO 40 for the other tone. EXPERIMENT!

For now, use Z here FOR Z=1 TO 100
and here NEXT Z

In the next chapter, you will learn more about Z and more things to do with FOR-NEXT loops.

Questions

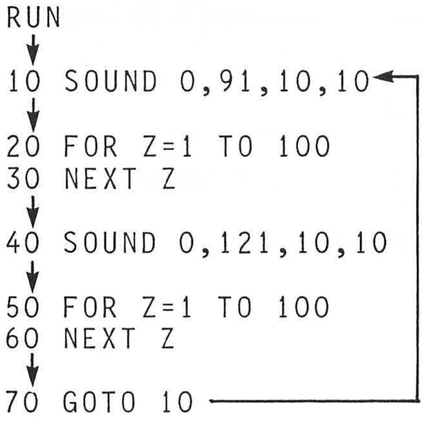
1. Draw arrows to show how the computer runs the program of this section. We include each time delay as a single item.

```
RUN

10 SOUND 0,91,10,10

20 FOR Z=1 TO 100
30 NEXT Z

40 SOUND 0,121,10,10

50 FOR Z=1 TO 100
60 NEXT Z

70 GOTO 10
```

2. Add a third sound. Use frequency number 108. Include a time delay for your sound and a new GOTO to complete the loop.

Answers

1.



```
RUN

10 SOUND 0,91,10,10

20 FOR Z=1 TO 100
30 NEXT Z

40 SOUND 0,121,10,10

50 FOR Z=1 TO 100
60 NEXT Z

70 GOTO 10
```

2. We did it this way: Our line 70 replaces the GOTO of the old program, so we put a new GOTO in line 100.

```
10 SOUND 0,91,10,10
20 FOR Z=1 TO 100              1st tone and delay
30 NEXT Z

40 SOUND 0,121,10,10
50 FOR Z=1 TO 100              2nd tone and delay
60 NEXT Z

70 SOUND 0,108,10,10
80 FOR Z=1 TO 100              3rd tone and delay
90 NEXT Z

100 GOTO 10                    Go do it again!
```

SOUND, COLOR, & A BIT OF STYLE

Our programs are getting longer. In order to make them easier for people to read and understand, we will frequently include REMARK statements. REMARK statements are for people. They explain what is happening in a program, but the computer ignores them. The REMARK statement is abbreviated as REM. For example:

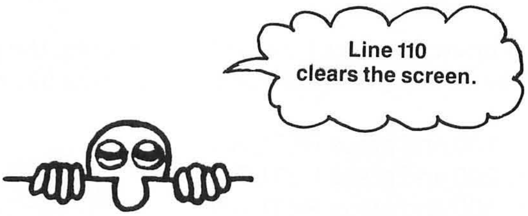
```
100 REM SIREN PROGRAM #1
```

To call attention to REM statements, we usually write them this way:

```
200 REM**PLAY 1ST TONE & DELAY
```

Here again is our "siren" program with REMARK statements to explain what is happening.

```
100 REM**SIREN PROGRAM #1
110 PR. CHR$(125)

200 REM**PLAY FIRST TONE & DELAY
210 SO. 0,91,10,10
220 FOR Z=1 TO 100
230 NEXT Z

300 REM**PLAY SECOND TONE & DELAY
310 SO. 0,121,10,10
320 FOR Z=1 TO 100
330 NEXT Z

400 REM**GO AROUND AGAIN
410 GOTO 210
```



Thanks to line 110, when the siren sounds you will see a completely blank screen, except for the cursor in its home position.

This clears the screen: PR. CHR$(125)

You will see many other uses of the CHR$ *function* in Chapter 10. For now, simply use it in a program to clear the screen.

Add some color to our program by adding these two lines:

```
215 SE. 2,3,4                 red-orange
315 SE. 2,7,4                 azure blue
```

If you have already entered the program, you can add the two new lines simply by typing them. If you then LIST the program, you should see:

```
LIST

100 REM**COLOR SIREN
110 PRINT CHR$(125)
200 REM**PLAY FIRST TONE & DELAY
210 SOUND 0,91,10,10
215 SETCOLOR 2,3,4
220 FOR Z=1 TO 100
230 NEXT Z
300 REM**PLAY 2ND TONE & DELAY
310 SOUND 0,121,10,10
315 SETCOLOR 2,7,4
320 FOR Z=1 TO 100
330 NEXT Z
400 REM**GO AROUND AGAIN
410 GOTO 210
```

We also changed the name of the program.

New line

New line

Since the computer ignores REMARK statements, the program will run just as well without them. Delete the REMARK statements like this:

Type: 100 and press RETURN.
Type: 200 and press RETURN.
Type: 300 and press RETURN.
Type: 400 and press RETURN.

Now LIST the program and you will see that the REM statements are gone. RUN the program and you will see and hear the same as before.

> REMEMBER. To delete a line (only that line), type the line number and press RETURN.

Since REM statements are ignored by the computer, you may omit them when you enter a program. We will continue to use them, however, to help you read and understand our programs.

Questions

1. Add a third color tone to the COLOR SIREN program. Begin at line 400 with a REM statement. You choose the tone number and the color. Also, remember to include a new GOTO to go back around.
2. How do you delete a single line from memory, without changing any other line?____________________
3. Write a line to clear the screen:____________________

Answers

1. We selected tone number 108 and the color green.

```
400 REM**PLAY 3RD TONE & DELAY
410 SO. 0,108,10,10
415 SE. 2,12,4
420 FOR Z=1 TO 100
430 NEXT Z

500 REM**GO AROUND AGAIN
510 GOTO 210
```

2. Type the line number and press RETURN.
3. 100 PR. CHR$(125)
   You will learn more about the CHR$ *function* in Chapter 10. Here is another way to clear the screen:

```
110 PR. "↰ "
```

To get the "broken arrow" character, press ESC, then hold down either SHIFT or CTRL and press the CLEAR key.

Still another way to clear the screen (and do lots of other things) is to begin the program with a GRAPHICS 0 line. For example:

```
110 GR. 0 or 110 GRAPHICS 0
```

This puts the computer in GRAPHICS 0 mode, thus clearing the screen. It also sets the color registers so that the screen colors are the same as if you had pressed SYSTEM RESET. Use PR. CHR$(125).when you want to clear the screen *without* changing anything else.

NAME BLINKER

The following program causes Chewbacca's name to blink on, off, and so on until someone presses BREAK:

```
100 REM**NAME BLINKER
110 GR. 0 ——————————— Start on GR. 0 with clear screen

200 REM**NAME IS ON-SCREEN
210 PR. "CHEWBACCA"
220 FOR Z=1 TO 500
230 NEXT Z

300 REM**BLANK SCREEN
310 PR. CHR$(125) ——————— Or use PR. " ↰ " here
320 FOR Z=1 TO 500
330 NEXT Z

400 REM**GO AROUND AGAIN
410 GOTO 210
```

Enter and RUN the program. You will see Chewbacca's name blinking in the upper left corner of the screen. Then, make some changes:

120 SE. 1,0,0	Dark letters
215 SE. 2,12,8	Green screen
315 SE. 2,4,8	Pink screen

With the above additions, Chewey's name will blink in dark letters on a green screen. When his name is not on-screen, you will see a blank pink screen, except for the cursor.

If you make the previous changes and LIST the program, you should see this:

```
LIST

100 REM**NAME BLINKER
110 GRAPHICS 0
120 SETCOLOR 1,0,0
200 REM**NAME IS ON-SCREEN
210 PRINT "CHEWBACCA"
215 SETCOLOR 2,12,8
220 FOR Z=1 TO 500
230 NEXT Z
300 REM**BLANK SCREEN
310 PRINT CHR$(125)
315 SETCOLOR 2,4,8
320 FOR Z=1 TO 500
330 NEXT Z
400 REM**GO AROUND AGAIN
410 GOTO 210
```

Here is another name blinker program. We have added SOUND and we use a different method to blank out the screen. We make everything on-screen invisible by setting the foreground (printed information) and background (screen color) to the same luminance.

```
100 REM**NAME BLINKER WITH SOUND
110 GR. 0

200 REM**NAME ON WITH SOUND
210 PR. CHR$(125)
220 SE. 1,0,0
230 SE. 2,12,8
240 PR."CHEWBACCA"
250 SO. 0,121,10,10
260 FOR Z = 1 TO 500
270 NEXT Z
280 SO. 0,0,0,0 ————————— This turns the sound off.
```

(continued)

```
300 REM**BLANK SCREEN
310 SE. 1,0,8
320 SE. 2,4,8
330 FOR Z=1 TO 500
340 NEXT Z

400 REM**GO AROUND AGAIN
410 GOTO 210
```

Lines 310 and 320 make everything on the screen invisible

Here is another way to write *blocks* 100 and 200.

```
100 REM**NAME BLINKER
110 GR. 0
120 PR. "CHEWBACCA"
```

This is block 100.

```
200 REM**INFO VISIBLE WITH SOUND
210 SE. 1,0,0
220 SE. 2,12,8
230 SO. 0,121,10,10
240 FOR Z=1 TO 500
250 NEXT Z
260 SO. 0,0,0,0
```

This is block 200.

Blocks 300 and 400 remain the same as before. If you try this variation, also delete lines 270 and 280 in the previous version.

To delete line 270, type: **270** and press RETURN.
To delete line 280, type: **280** and press RETURN.

Questions

1. We used PR. CHR$(125) to clear the screen. How else could we have done it?

____

2. How do you make information on the screen invisible?

____

3. How do you tell the computer to turn off the sound?

____

4. How would you make CHEWBACCA blink more rapidly?

____

____

Answers

1. Use PR. "↰ " To type the broken arrow (↰), press ESC, then hold down either CTRL or SHIFT and press CLEAR.

2. Use the same luminance for both the foreground and background. For example:

```
310 SE. 1,0,8                  Foreground, luminance 8.
320 SE. 2,4,8                  Background, luminance 8.
```

3. Use SOUND 0,0,0,0 to turn off voice 0, SOUND 1,0,0,0 to turn off voice 1, and so on.

4. Use a smaller number in the FOR statements. For example:

```
FOR Z=1 TO 100
```

SELF-TEST

Congratulations! You have completed another chapter. Take this Self-Test to remind yourself about what you have learned in Chapter 3.

1. Describe the result of using each of the following commands.

   (a) NEW____

   (b) LIST____

   (c) RUN____

2. Draw arrows to show how the computer obeys the following program.

```
RUN
10 PR. "LUCY"
20 PR. "IS GREAT!"
30 GOTO 20
```

3. When you enter a program for later use, you begin each line with a ______________.

4. Describe what happens when you hold down the CTRL key and press each of the following keys.

(a) [↑ - _] ______________________________

(b) [↓ | =] ______________________________

(c) [← \ +] ______________________________

(d) [→ ^ *] ______________________________

(e) [DELETE BACK S] ______________________________

(f) [INSERT >] ______________________________

5. If you play a musical instrument, you know about scales. You may have spent many joyful (?) hours practicing scales on a piano, guitar, flute, or other instrument. And, if you saw "The Sound of Music," you know that it all begins with DO, RE, MI! Below are the tone numbers for DO, RE, MI in the scale of C.

DO	RE	MI	FA	SOL	LA	TI	DO
121	108	96	91	81	72	64	60

Write a program to display and play one note at a time. For DO for a little while, then RE, and so on. We have begun the program for you. At the end of the program, use a GOTO 210 to start over.

```
100 REM**DO, RE, MI, ETC.
110 GR. 0

200 REM**PRINT & PLAY DO
210 PR. CHR$(125)
220 PR. "DO"
230 SO. 0,121,10,10
240 FOR Z=1 TO 500
250 NEXT Z
```

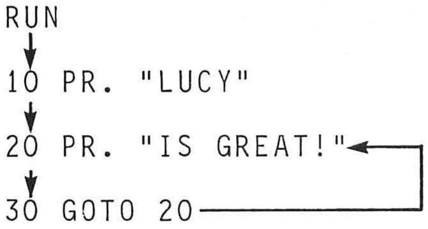
6. Modify the program of Question 5 so the screen is a different color for each note.

Answers to Self-Test

1. (a) NEW erases any programs in memory (RAM).

   (b) LIST prints on the screen the program in memory (RAM).

   (c) RUN tells the computer to carry out, do, or obey the program in memory (RAM).

2. 

```
RUN
10 PR. "LUCY"
20 PR. "IS GREAT!"
30 GOTO 20
```

3. line number.

4. (a) Moves the cursor up one line.

   (b) Moves the cursor down one line.

   (c) Moves the cursor left one space.

   (d) Moves the cursor right one space.

   (e) Erases the character under the cursor and moves everything to the right of the cursor one space left.

   (f) Moves the character beneath the cursor and everything to the right of the cursor one space right. Puts a space under the cursor.

5. Here is the rest of our program.

```
300 REM**PRINT & PLAY RE
310 PR. CHR$(125)
320 PR. "RE"
330 SO. 0,108,10,10
340 FOR Z = 1 TO 500
350 NEXT Z
```

(continued)

```
400 REM**PRINT & PLAY MI
410 PR. CHR$(125)
420 PR. "MI"
430 SO. 0,96,10,10
440 FOR Z=1 TO 500
450 NEXT Z

500 REM**PRINT & PLAY FA
510 PR. CHR$(125)
520 PR. "FA"
530 SO. 0,91,10,10
540 FOR Z=1 TO 500
550 NEXT Z

600 REM**PRINT & PLAY SOL
610 PR. CHR$(125)
620 PR. "SOL"
630 SO. 0,81,10,10
640 FOR Z=1 TO 500
650 NEXT Z

700 REM**PRINT & PLAY LA
710 PR. CHR$(125)
720 PR. "LA"
730 SO. 0,72,10,10
740 FOR Z=1 TO 500
750 NEXT Z

800 REM**PRINT & PLAY TI
810 PR. CHR$(125)
820 PR. "TI"
830 SO. 0,64,10,10
840 FOR Z=1 TO 500
850 NEXT Z
```

(continued)

```
900 REM**PRINT & PLAY DO
910 PR. CHR$(125)
920 PR. "DO"
930 SO. 0,60,10,10
940 FOR Z=1 TO 500
950 NEXT Z

1000 REM**GO AROUND AGAIN
1010 GOTO 210
```

6. We leave this one entirely to you.

Chapter Four

Number Boxes

As you learn more about ATARI BASIC, you will find it easier to get the computer to do what you want it to do. In this chapter you will learn about places in the computer's memory which we call *number boxes*. Think of number boxes as places to store numbers or as place holders where numbers can be stored while the computer uses them.

Number boxes are identified by labels called *numeric variables*. You will use numeric variables to represent numbers in BASIC programs. You will learn three ways to put numbers into number boxes, or assign *values* to numeric variables.

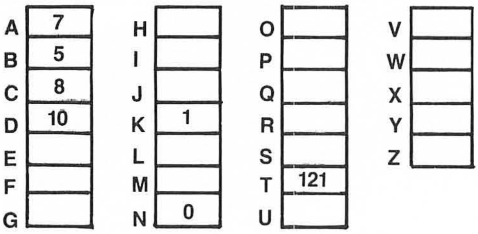
When you finish this chapter, you will be able to:

- Recognize and use numeric variables
- Assign values to numeric variables
- Use numeric values in SOUND and SETCOLOR to produce sound and color effects
- Use INPUT statements to enter values of numeric variables
- Use READ and DATA statements to supply values of numeric variables

Soon you will get the computer to do more while you do less!

26 BOXES

Imagine that, deep down inside the computer, there are 26 little *number boxes*. Each number box can hold one number at any one time.



Some of the boxes have numbers in them. For example, 7 is in box A and 5 is in box B. The letters that identify boxes (A,B,C, and so on) are called *variables*. The number in box A is called the *value of A*; the number in box B is the *value of B*; the number in C is the *value of C*; and so on.

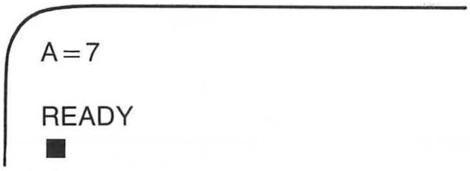
- The value of A is 7.
- The value of B is 5.
- The value of T is 121.

Number boxes can hold numbers only. The labels on number boxes (A,B,C, etc.) are sometimes called *numeric* variables.

Later you will also learn about *string boxes*, which can hold...(suspense)...strings!And you will also learn about *string variables*, whose values are strings.

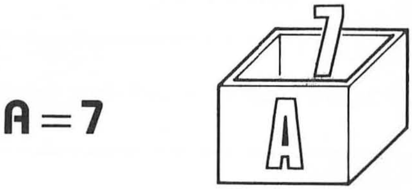
How do you put numbers into number boxes? It's easy!

- Clear the screen
- Type: A = 7 and press RETURN.

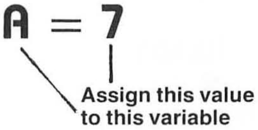


```
A=7

READY
■
```

The computer has put the number 7 into box A. Or, the computer has assigned the value 7 to the variable A. Note how you did this.



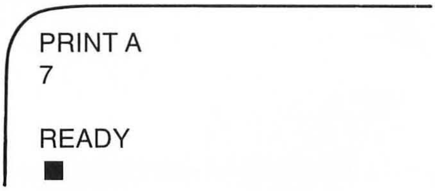
The command A = 7 tells the computer to put the number 7 into number box A. In other words, the command A = 7 tells the computer, "Assign the number 7 as the value of the variable A."



Is the number 7 in number box A? Find out. Clear the screen.

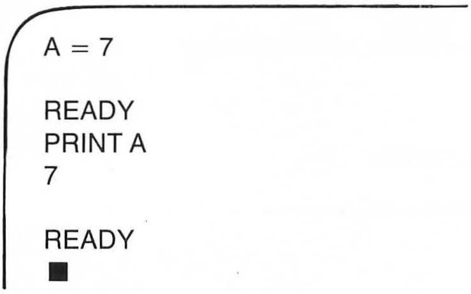
Type: PRINT A and press RETURN.

The screen should read:



```
PRINT A
7

READY
■
```

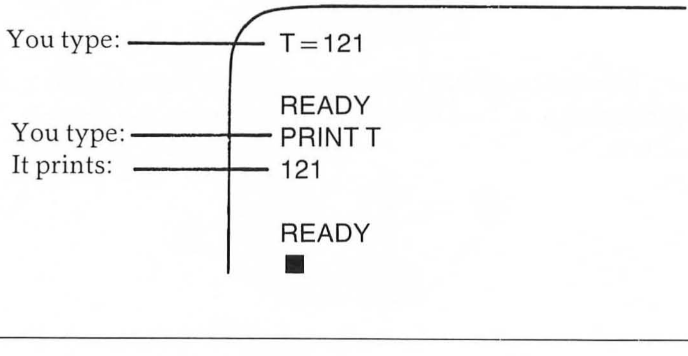
If you forgot to clear the screen before typing PRINT A, the screen might now read:



```
A = 7

READY
PRINT A
7

READY
■
```

> REMEMBER: To tell the computer to print the number that is in number box A (or to print the *value* of *variable* A), you type:
>
> **PRINT A**

Next, assign the number 121 to the variable T and then print the value of T. The screen might read:



```
T=121

READY
PRINT T
121

READY
■
```

Note the difference between the two PRINT statements below.

WITH "

PRINT "T"

Tells the computer to print the string T which is enclosed in quotation marks.

WITHOUT "

PRINT T

Tells the computer to print the value of the variable T.

You type: PRINT "T"
It prints: T

You type: PRINT T
It prints: 121

Questions

1. Some numbers are stored in number boxes:

A	7	H		O		V	
B	5	I		P		W	
C	8	J		Q		X	
D	10	K	1	R		Y	
E		L		S		Z	
F		M		T	121		
G		N	0	U			

(a) What number is in box T?________

(b) What number is in box D?________

(c) What box contains the number 1?________

(d) What box contains the number 8?________

(e) What box contains zero (0)?________

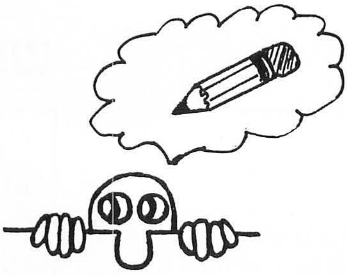
(f) You put the number 100 into box Z.

(g) What is the value of the variable A?________

(h) What number is the value of K?________

2. Boxes C and T are shown below. They are empty. Use a *pencil* to do the following.

   C □ T □



   (a) Put 3 into box C.

   (b) Put 89 into box T.

   (c) Put 125 into box T. But wait! A box can hold only *one* number at a time. Before you can put 125 into box T, you must first erase the 89 that you put there previously.

3. Pretend you are the computer and complete the following:

   (a) You type: Z=37
   You type: PRINT Z
   It prints:________

   (b) You type: P=12
   You type: P=88
   You type: PRINT P
   It prints: ________

   (c) You type: A=7
   You type: B=A
   You type: PRINT B
   It prints: ________

   (d) You type: C=46
   You type: PRINT "C"
   It prints:________

Answers

1. (a) 121; (b) 10; (c) K; (d) C; (e) N

   (f) Box Z should now look like this: Z $\boxed{100}$

   (g) 7; (h) 1

2. (a) C $\boxed{3}$; (b) T $\boxed{89}$; (c) T $\boxed{125}$
   When the computer puts a number into a box, it first erases the previous number (if any) that was in the box.

3. (a) 37; (b) 88; (c) 7; (d) C
   In (b), the 88 replaced the number 12 as the value of P. In (c), the command B=A assigned the value of A as the value of B. In (d), watch those quotation marks!

SOUND EFFECTS

In Chapter 3, you used FOR-NEXT loops as a time delay. For example, the following FOR-NEXT loop tells the computer to count from 1 to 100.

```
Start here   Stop here
220 FOR Z=1 TO 100
230 NEXT
```

As the computer counts from 1 to 100, each counting number is put, momentarily, into number box Z. Instead of Z, you can use any numeric variable. For example:

```
220 FOR T=1 TO 255
230 NEXT T
```

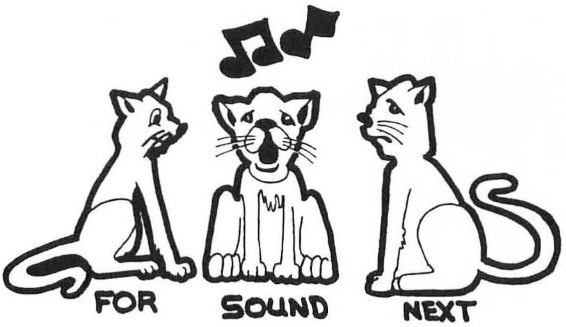
Now comes the boggler. You can put other statements *between* the FOR statement and the NEXT statement. Try this program.

```
10 GR. 0

20 FOR T=1 TO 255
30 SO. 0,T,10,10
40 NEXT T
```

From now on we will usually begin program with GR. 0.

This FOR-NEXT loop has *three* statements.

In the above FOR-NEXT loop the SOUND statement (line 30) is *between* the FOR statement and the NEXT statement.



The variable T is used in three places, in the FOR statement, the SOUND statement, and the NEXT statement. As T goes from 1 to 255, the sound will go from high to low. Try it. Enter and RUN the program. Then add this line to hear the entire sequence of sounds repeat until you press SYSTEM RESET.

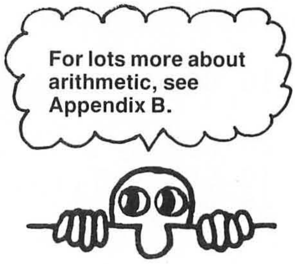
Add this line: **50 GOTO 20**

Next, try this one, which makes the sound go from low to high. Change only line 30.

```
10 GR. 0
20 FOR T=1 TO 255
30 SO. 0,255-T,10,10
40 NEXT T
50 GOTO 20
```

Line 30 has a little arithmetic. As T goes from 1 to 255, the value of 255-T goes from 254 to 0. This gives tones from a low tone (254) to a high tone (1) to no tone (0).

T	255-T
1	254
2	253
3	252
:	:
254	1
255	0



Using two voices, we can combine these effects, with one voice going from high tone to a low tone while the other voice goes from a low tone to a high tone.

```
10 GR. 0
20 FOR T=1 TO 255
30 SO. 0,T,10,10
40 SO. 1,255-T,10,10
50 NEXT T
60 GOTO 20
```

EXPERIMENT! Instead of having T go from 1 to 255, try one of these:

```
20 FOR T=1 TO 100         T goes from 1 to 100.
20 FOR T=100 TO 255       T goes from 100 to 255.
20 FOR T=50 TO 100        T goes from 50 to 100.
```

Or use numbers of your choice. Then change line 20 as follows:

```
20 FOR T=1 TO 255 STEP 2
```

This FOR statement tells the computer to count from 1 to 255 by steps of 2. T is equal to 1, then 3, then 5, then 7, and so on, up to 255. Run any of the programs with this change and listen to the SOUND. Then change line 20 again, so the computer counts in steps of 4.

```
20 FOR T=1 TO 255 STEP 4
```

Also try steps of 8 or 16 or whatever you want.

Questions

1. Describe what will happen if you run the following program.

```
10 GR. 0
20 FOR L=1 TO 15
30 SO. 0,121,10,L
40 NEXT L
50 GOTO 20
```

2. Change the program in question 1 so the loudness (L) goes from 15 to 0 in steps of − 1.________________________________

3. Describe what happens if you run the following program.

```
10 GR. 0
20 FOR T=1 TO 200
30 SO. 0,T,10,10
40 SO. 1,T+20,10,10
50 NEXT T
60 GOTO 20
```

4. Change the program in question 3 so the tone (T) goes from 200 to 1 in steps of − 1.

5. Write a program to cause the tone (T) to go from 1 to 255, then back down to 1, then up to 255, then down to 1, and so on.

Answers

1. Line 10 puts the computer in GRAPHICS 0 mode and clears the screen. The FOR-NEXT loop (lines 20,30,40) sounds tone number 121 with loudness (L) going from 1 (soft) to 15 (loud). Line 50 causes the sound to repeat. Try this: put a time delay between line 30 and line 40.

```
10 GR. 0
20 FOR L=1 TO 15
30 SO. 0,121,10,L
35 FOR Z=1 TO 200 ┐
                  ├── Time delay
37 NEXT Z        ─┘
40 NEXT L
50 GOTO 20
```

Also try shorter or longer delays.

2. Change only line 20, like this: 20 FOR L=15 TO 0 STEP −1

3. You hear two voices going from high notes to low notes. Voice 0 starts at T=1 and goes to T=200. Voice one starts at 21 (1+20) and goes to 220 (200+20).

4. Change only line 20: 20 FOR T=200 TO 1 STEP −1

5.
```
10 GR.0
20 FOR T=1 TO 255
30 SO. 0,T,10,10
40 NEXT T
50 FOR T=255 TO 1 STEP -1
60 SO. 0,T,10,10
70 NEXT T
80 GOTO 20
```

**EXPERIMENT!
Try your own variations — play them on the computer.**

TURN ON THE RAINBOW

You can make the screen change color rapidly, or slowly if you prefer, by using SETCOLOR with FOR-NEXT. But first, here is a time-saving way to write a time delay.

```
FOR Z=1 TO 100: NEXT Z
```

Put a colon here.

Yes, you can put two or more statements on a line as long as you separate them with colons.

```
FOR Z=1 TO 100: NEXT Z
```

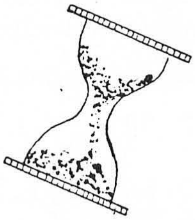
1st statement

colon

2nd statement

We will use this idea in the program below to tell the computer to flash the colors from 0 to 15.

```
10 GR. 0
20 FOR C=0 TO 15
30 SE. 2,C,8
40 FOR Z=1 TO 200:   NEXT Z
50 NEXT C
60 GOTO 20
```



Lines 30 and 40 are both *between* the FOR statement in line 20 and the NEXT statement in line 50. Therefore, lines 30 and 40 are done for *each* value of C from C=0 to C=15.

From now on, we will indent statements inside FOR-NEXT loops two spaces. We do this simply to make programs easier for people to read. The computer ignores the spaces — it won't show them when you LIST the program.

```
10 GR.0

20 FOR C=0 TO 15                          Beginning of loop

30   SE. 2,C,8                            Inside of loop
40   FOR Z=1 TO 200: NEXT Z               is indented

50 NEXT C                                 End of loop

60 GOTO 20
```

In the above program, we indented the inside of the loop and added extra line spaces to make (we hope) the program easier for *you* to read and understand. However, if you type the program as shown above, the computer ignores the extra line space and the indentation.

Next, a program to "fade in" a name in and out. This program works by slowly changing the luminance of the name on the screen. At first, in lines 210 and 320, the foreground (name) and the background have the same luminance — therefore, the name is invisible. Block 400 slowly changes the luminance of the foreground from 0 to 15; the name "fades in." Then block 500 slowly changes the foreground luminance from 15 to 0; the name fades out. This happens on a dark gray screen.

```
100 REM**NAME FADER
110 GR. 0

200 REM**SET BACKGROUND COLOR
210 SE. 2,0,0

300 REM**PRINT NAME INVISIBLY
310 SE. 1,0,0
320 PR. "TENDERHEART"

400 REM**MAKE NAME SLOWLY VISIBLE
410 FOR L=0 TO 15
420   SE. 1,0,L
430   FOR Z=1 TO 30: NEXT Z
440 NEXT L

500 REM**MAKE NAME SLOWLY INVISIBLE
510 FOR L=15 TO 0 STEP -1
520   SE. 1,0,L
530   FOR Z=1 TO 30:  NEXT Z
540 NEXT L

600 REM**GO DO IT AGAIN
610 GOTO 410
```

Black

Fade in

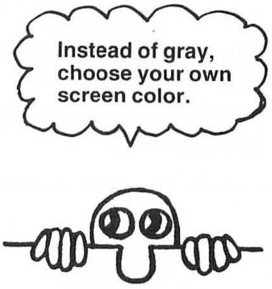
Fade out

Your turn — try some variations. Fade in and out on a white screen. Make these changes:

```
210 SE. 2,0,15
310 SE. 1,0,15
410 FOR L=15 TO 0 STEP -1
510 FOR L=0 TO 15
```

Keep the foreground luminance constant and fade the background luminance. Change the program as follows:

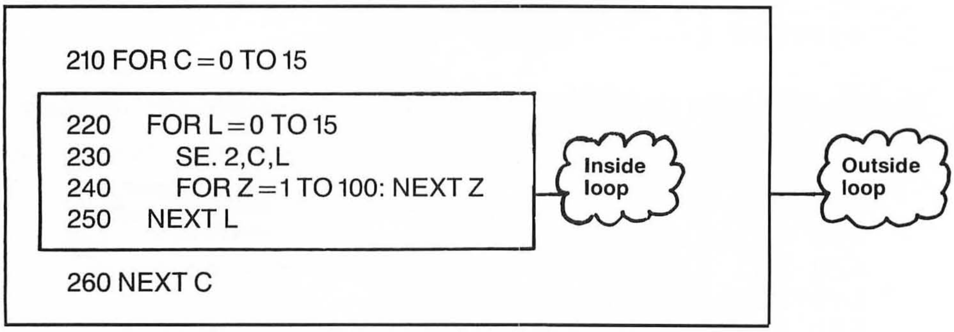
```
210 SE. 2,0,0
310 SE. 1,0,0
420 SE. 2,0,L
510 FOR L=15 TO 0 STEP -1
520 SE. 2,0,L
```



You may use 16 colors (0 to 15) and each color may have a luminance from 0 (darkest) to 15 (brightest). Well, actually there are eight distinct luminances: 0 and 1 are the same, 2 and 3 are the same, and so on. Let's ask the ATARI to flash all combinations of color and luminance.

```
100 REM**COLOR & LUMINANCE
110 GR. 0

200 REM**C=COLOR & L=LUMINANCE
210 FOR C=0 TO 15
220   FOR L=0 TO 15
230     SE. 2,C,L
240     FOR Z=1 TO 100: NEXT Z
250   NEXT L
260 NEXT C

300 REM**GO DO IT AGAIN
310 GOTO 210
```

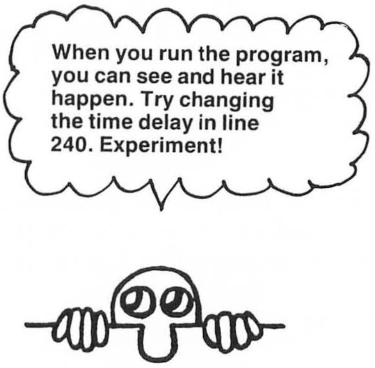
Here we have a FOR-NEXT using L *nested* inside a FOR-NEXT loop using C. There is also a FOR-NEXT loop using Z included as part of the inside loop.



```
210 FOR C=0 TO 15

220   FOR L=0 TO 15
230      SE. 2,C,L
240      FOR Z=1 TO 100: NEXT Z
250   NEXT L

260 NEXT C
```

In the inside loop, lines 230 and 240 are carried out for L going from 0 to 15 for *each* value of C in the outside loop. It's like a wheel within a wheel.

C	L
0	0
	1
	2
	:
	15
1	0
	1
	2
	:
	15
2	0
	1
	2
	:
	15
and so on.	


When you run the program, you can see and hear it happen. Try changing the time delay in line 240. Experiment!

Add some sound:

```
235     SO. 0,C*L+1,10,10
```

The expression C*L + 1 means "C times L plus 1." See Appendix B for detailed information on arithmetic.

Questions

1. If you put two statements on one line, what must you put between the statements?__________

2. Complete the following programs to flash the hues (colors) from 15 to 0.

```
10 PR.________
20 FOR______________
30   SE.2,H,8
40   FOR Z=1 TO 200:NEXTZ
50 NEXT________
60 GOTO20
```

3. To the program in question 2 above, add line 35 to tell the computer to "sound-off." In your SOUND statement, make the frequency number equal to the square of the hue (H*H).

```
35 ______________
```

4. In the COLOR & LUMINANCE program, make the frequency number equal to 255-C*L. The expression 255-C*L means "255 minus C times L."

```
235______________
```

5. Add sound to the NAME FADER program.

6. EXPERIMENT! Try other distortion numbers in the SOUND statements. Make the distortion depend on the color or the luminance.

Answers

1. A colon(:).

2.

```
10 GR. 0
20 FOR H=15 TO 0 STEP -1
50 NEXT H
```

Since we used H as the hue (color) variable in the SETCOLOR statement (line 30), you must use H in the FOR and NEXT statements.

3. 35 SO.0,H*H,10,10

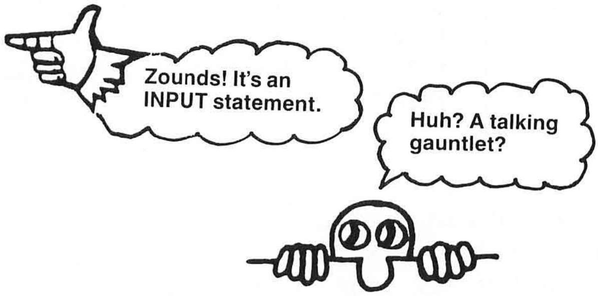
4. 235 SP.0,255-C*L,10,10

5,6. Experiment with different ways to do this. Also play with loudness. Try using two or more voices. Change the time delay or even delete it. EXPERIMENT! Your ATARI computer and your own imagination are your best and always-ready teachers.

USE INPUT TO STUFF NUMBER BOXES

The INPUT statement is very useful for box stuffing. The following program introduces the INPUT statement.

```
10 GR. 0
20 INPUT T
30 PRINT T
40 GOTO 20
```



Enter the above program, type RUN, and press the RETURN key.

```
?■
```

You get a question mark and the cursor.

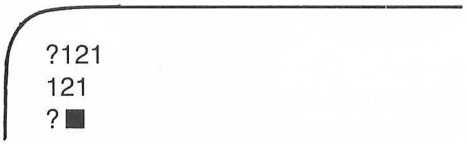
The computer is doing the INPUT statement. The statement: 20 INPUT T tells the computer:

(1) type a question mark;

(2) turn on the cursor;

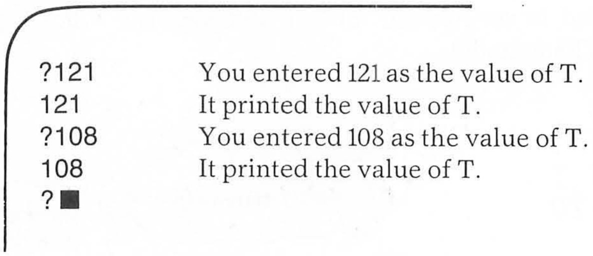
(3) wait for someone to type a number. When you type a number and press RETURN, the computer puts the number into box T and goes on to the next statement in line-number order.

Type 121 and press RETURN. The screen looks like this:



```
?121
121
?■
```

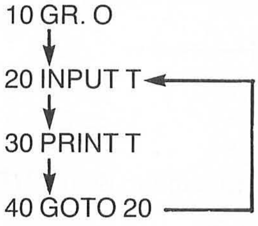
The computer wants another number, a new value for T. Cooperate with your ever-patient computer. Type 108 and press RETURN.



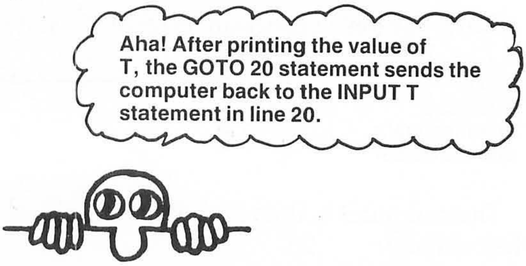
```
?121          You entered 121 as the value of T.
121           It printed the value of T.
?108          You entered 108 as the value of T.
108           It printed the value of T.
?■
```

What? It wants still another number? How do you tell the computer to stop? Press the BREAK key.

Why does the computer keep asking for a value of T? Follow the arrows.

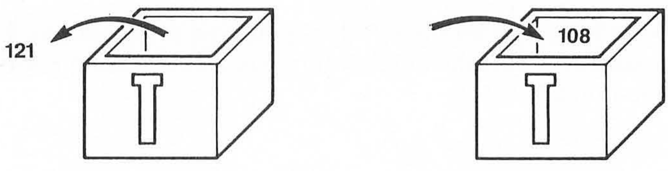


```
10 GR. O
20 INPUT T
30 PRINT T
40 GOTO 20
```



Aha! After printing the value of T, the GOTO 20 statement sends the computer back to the INPUT T statement in line 20.

The computer goes from line 20 to line 30 only after you type a value of T and press RETURN. This value *replaces* the previous value of T.



OUT WITH THE OLD   IN WITH THE NEW

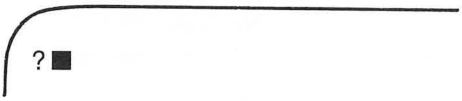
THAT'S WHAT INPUT CAN DO FOR YOU.

You can use the INPUT variable in a SETCOLOR statement. In line 40 below, we use the variable C for "Color."

```
10 GR. 0
20 INPUT C
30 SETCOLOR 1, 0, 0            We want black letters.
40 SETCOLOR 2, C, 8            Set background color.
50 GOTO 10
```

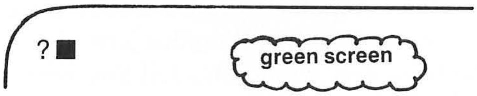
C for Color.

Try it. Enter the program and RUN it.



```
?■
```

Try a green screen. Type 12 and press RETURN.



```
?■
```

Try any color number from 0 to 15. Also try 16, 17, and so on. You will see that 16 is the same color as zero, 17 the same as one, and so on. Remember, the actual colors depend on the color settings on your TV.

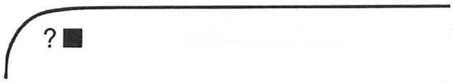
EXPERIMENT! Change line 40 to: 40 SETCOLOR 4,C,8 Now you can enter numbers to change the border color of the screen.

You can use an INPUT variable in a SOUND statement.

```
10 GR. 0
20 INPUT N
30 SOUND 0,N,10,10
40 GOTO 20
```

N for Note. Look for it in lines 20 and 30.

A RUN might go like this:



```
?■
```

Waiting for a value of N.

Enter 121 and press RETURN.

```
?121
?■
```

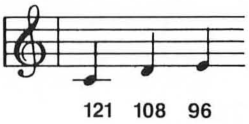
You hear note number 121.
The computer waits for another note.

Enter 108 as the next value of N.

```
?121
?108          You now hear note number 108.
?■
```

Suppose you enter 96 as the next value of N. The screen should appear as follows:

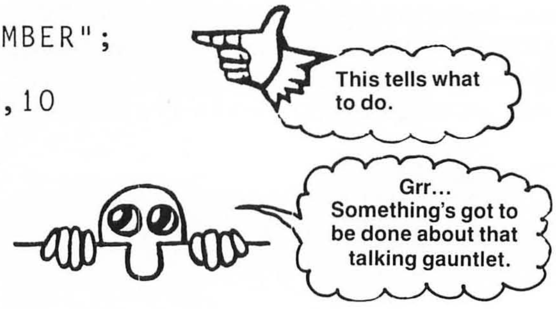
```
?121
?108
?96
?■
```



Experiment! Try low notes (N=255), higher notes (N=100), still higher notes (N=10). Type note numbers and press RETURN as fast as you can. Is it music?

The INPUT statement causes the computer to put a question mark on the screen, then wait for someone to enter something. Wouldn't it be nice if, instead of just a lonely question mark, the computer would also tell you what it wanted? Easy! Use a Print statement to tell what is wanted.

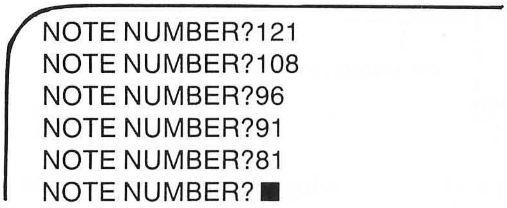
```
10 GR. 0
20 PR. "NOTE NUMBER";
30 INPUT N
40 SOUND 0,N,10,10
50 GOTO 20
```


This tells what to do.
Grr... Something's got to be done about that talking gauntlet.

As usual, to find out what a program does, enter it into the computer and RUN it.

```
NOTE NUMBER?■
```

Now you know what the computer wants! Enter several notes, one after the other. For example:



```
NOTE NUMBER?121
NOTE NUMBER?108
NOTE NUMBER?96
NOTE NUMBER?91
NOTE NUMBER?81
NOTE NUMBER?■
```

When you are tired of entering note numbers, press the BREAK key to stop the program. Oh, the sound is still on?

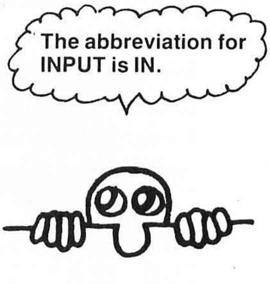
Type END and press RETURN, or
Press the SYSTEM RESET key.

We have been using single letters as variables. We confess — we didn't tell the whole truth about variables. You can use more than a single letter as a variable. For example,

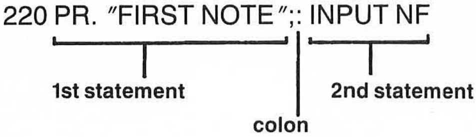
NF □ NL □ SS □ TD □

The above variables appear in the following program.

```
100 REM**SOUND EXPERIMENTER
110 GR. 0

200 REM**GET INFORMATION
210 PR. CHR$(125)
220 PR. "FIRST NOTE";: INPUT NF
230 PR. "LAST NOTE";: INPUT NL
240 PR. "STEP SIZE";: INPUT SS
250 PR. "TIME DELAY";: INPUT TD

300 REM**PLAY THE NOTES
310 FOR N=NF TO NL STEP SS
320   SO. 0,N,10,10
330   FOR Z=1 TO TD: NEXT Z
340 NEXT N

400 REM**TURN OFF SOUND & GO AROUND
410 SO. 0,0,0,0
420 GOTO 210
```



Lines 220, 230, 240 and 250 each contain two statements. For instance:



```
220 PR. "FIRST NOTE";: INPUT NF
```

We ran the program and supplied the following values for NF, NL, SS, and TD.

```
FIRST NOTE?255
LAST NOTE?1
STEP SIZE? - 1
TIME DELAY?10■
```

Before pressing RETURN

After we pressed RETURN, the computer played the notes 255, 254, 253, and so on down to 1, then politely inquired:

```
FIRST NOTE? ■
```

Go ahead and experiment with this program.

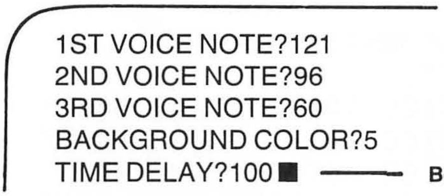
IMPORTANT NOTICE: A variable name may be up to 120 characters long! It must begin with a letter A to Z and may contain only capital letters and numbers 0 to 9. We suggest you do not use a special ATARI BASIC word such as PRINT or FOR or GOTO as a variable or as the first part of a variable. See Appendix F for a complete list of special ATARI BASIC words.

Questions

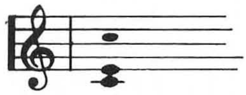
1. Write a program to ask for the background color (BKC) and background luminance (BKL), border color (BRC) and border luminance (BRL), and foreground luminance (FL). Then set the screen to these colors and go around again. Here is an outline of our program.

```
100 REM**SCREEN COLORS & LUMINANCES
200 REM**GET INFORMATION
300 REM**SET BACKGRND, BORDER, FOREGRND
400 REM**GO AROUND AGAIN
```

2. Write a program to play a "color chord" using three voices. When you run your program, it should begin this way:



```
1ST VOICE NOTE?121
2ND VOICE NOTE?96
3RD VOICE NOTE?60
BACKGROUND COLOR?5
TIME DELAY?100■
```



Before pressing RETURN

Press RETURN and hear the chord on a purple screen. Musicians call a three-note chord a *triad*.

Answers

1.
```
100 REM**SCREEN COLORS & LUMINANCES
110 GR.0

200 REM**GET INFORMATION
201 PR. CHR$(125)
220 ? "BACKGROUND COLOR";:INPUT BKC
230 ? "BACKGROUND LUMINANCE";:INPUT BKL
240 ? "BORDER COLOR";:INPUT BRC
250 ? "BORDER LUMINANCE";:INPUT BRL
260 ? "FOREGROUND LUMINANCE";:INPUT FL

300 REM**SET BACKGRND, BORDER, FOREGRND
210 SE. 2, BKC, BKL
320 SE. 4, BRC, BRL
330 SE. 1, 0, FL

400 REM**GO AROUND AGAIN
410 GOTO 210
```

In lines 220 through 260, we used a question mark (?) as an abbreviation for PRINT. There is also an abbreviation for INPUT.

IN. is the abbreviation for INPUT

2.

```
100 REM**COLOR CHORD
110 GR. 0

200 REM**GET INFORMATION
210 PR. CHR$(125)
220 PR. "1ST VOICE NOTE";: IN. N1
230 PR. "2ND VOICE NOTE";: IN. N2
240 PR. "3RD VOICE NOTE";: IN. N3
250 PR. "BACKGROUND COLOR";: IN. BKC
260 PR. "TIME DELAY";: IN. TD

300 REM** PLAY THE CHORD
310 SE. 2, BKC, 8
320 SO. 0, N1, 10, 10
330 SO. 1, N2, 10, 10
340 SO. 2, N3, 10, 10

400 REM** TIME DELAY
410 FOR Z=1 TO TD: NEXT Z

500 REM**TURN OFF THE VOICES
510 SO. 0,0,0,0
520 SO. 1,0,0,0
530 SO. 2,0,0,0

500 REM**GO PLAY ANOTHER ONE
610 GOTO 210
```

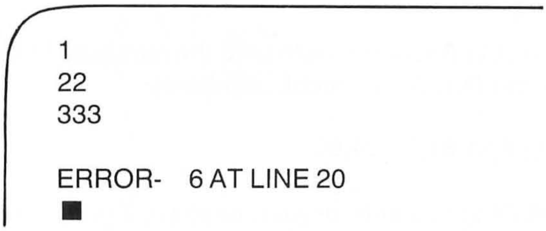
USE READ AND DATA TO STUFF NUMBER BOXES

There is yet another way to stuff numbers into number boxes, or — in *computerese* — assign values to numeric variables. Use READ and DATA statements, as shown in the following program.

```
10 GR. 0
20 READ N ——————————— This is a READ statement.
30 PRINT N
40 GOTO 20
50 DATA 1, 22, 333 ——— This is a DATA statement.
```

The statement: 20 READ N tells the computer to read one number from a DATA list and put the number into box N (assign the number as the value of N). Every time 20 is executed, the *next* number in the DATA list is read and assigned as the value of N.

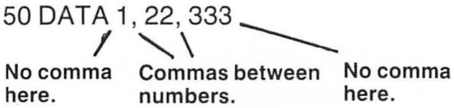
This is how the screen looks when you RUN the program:



```
1
22
333

ERROR-   6 AT LINE 20
■
```

The error message really means "Out of data in line 20." There are three numbers in the DATA statement. The computer read and printed all three numbers, then tried to read a fourth number. Since there are only three numbers in the DATA statement, it printed an error message. That's OK, since it was finished doing what we wanted it to do.

Look at the DATA statement in line 50. Commas separate the numbers in the list.



```
50 DATA 1, 22, 333
```

READ and DATA statements make it easy to play music. Here is a program to play a DO, RE, MI scale in the key of C.

```
100 REM**READ & DATA MUSIC
110 GR.0

200 REM**READ A NOTE NUMBER
210 READ N

300 REM**PLAY THE NOTE
310 SO. 0,N,10,10
320 FOR Z=1 TO 128: NEXT Z
330 SO. 0,0,0,0
```

(continued)

```
400 REM**GO FOR ANOTHER NOTE
410 GOTO 210

900 REM**HERE ARE THE NOTES
910 DATA 121,108,96,91
920 DATA 81, 72, 64,60
```

We intentionally used two DATA statements to hold the numbers, but we could have put all the numbers in *one* DATA statement, as follows:

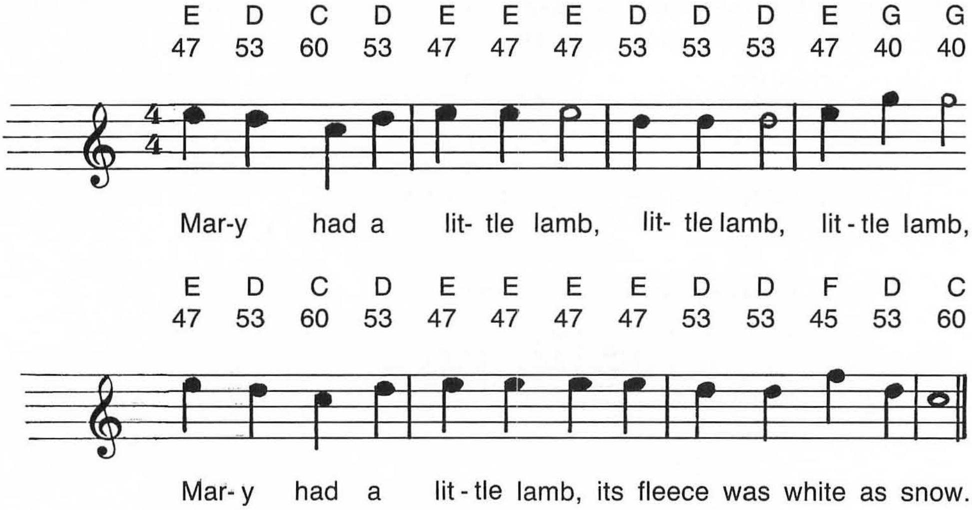
```
910 DATA 121,108,96,91,81,72,64,60
```

Or, you may use three DATA statements, or four, or more, if you wish. Want to have the computer play a tune? Put as many tone numbers as you wish for the tune in DATA statements.

VARIATION: Change block 200 of the program, as follows:

```
200 REM**READ & PRINT NOTE
210 READ N
220 PRINT N
```

And now, a little familiar music — *Mary Had a Little Lamb* in the scale of C. Here is the music (We have added the note letters and numbers):


E D C D E E E D D D E G G
47 53 60 53 47 47 47 53 53 53 47 40 40
4
4
Mar-y had a lit- tle lamb, lit- tle lamb, lit - tle lamb,
E D C D E E E E D D F D C
47 53 60 53 47 47 47 47 53 53 45 53 60
Mar- y had a lit - tle lamb, its fleece was white as snow.

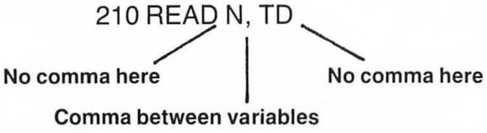
For each note, we also need a time delay. In the above melody, there are quarter-notes (♩), half-notes (𝅗𝅥), and whole-notes (𝅝). We will use time delays for these as follows:

TYPE OF NOTE	TIME DELAY
quarter (♩)	128
half (𝅗𝅥)	256
whole(𝅝)	512

A half-note is twice as long as a quarter-note and a whole-note is twice as long as a half-note and four times as long as a quarter note. In our program, we put both the note numbers and time delays in DATA statements. Our READ statement (line 210) reads both the note number and the time delay. We will explain the READ statement later, after you enter and run the program.

```
100 REM**PLAY A TUNE #1
110 PR. CHR$(125)

300 REM**READ NOTE & TIME DELAY
310 READ N, TD

500 REM**PLAY THE NOTE
510 SO. 0,N,10,10
520 FOR Z=1 TO TD: NEXT Z
530 SO. 0,0,0,0

600 REM**GO FOR ANOTHER NOTE
610 GOTO 310

900 REM**NOTE NUMBERS & TIME DELAYS
910 DATA 47,128,53,128,60,128,53,128
920 DATA 47,128,47,128,47,256
930 DATA 53,128,53,128,53,256
940 DATA 47,128,47,128,40,256
950 DATA 47,128,53,128,60,128,53,128
960 DATA 47,128,47,128,47,128,47,128
970 DATA 53,128,53,128,45,128,53,128
980 DATA 60,512
```

The READ statement in line 310 reads *two* numbers from a DATA statement, a note number (N) and a time delay number (TD).



You can make the melody play faster or slower by changing the time delays. However, there is a better way. Here is another version of *Mary Had a Little Lamb*.

```
100 REM**PLAY A TUNE #2
110 GR. 0

200 REM**READ TEMPO (HOW FAST TO PLAY)
210 READ TEMPO
```

This is new.

```
300 REM**READ NOTE & TYPE OF NOTE (TN)
310 READ N, TN

400 REM**COMPUTE TIME DELAY (TD)
410 TD = TEMPO/TN
```

This is new.

```
500 REM**PLAY THE NOTE
510 SO. 0,N,10,10
520 FOR Z=1 TO TD: NEXT Z
530 SO. 0,0,0,0

600 REM**GO FOR ANOTHER NOTE
610 GOTO 310

900 REM**TEMPO, NOTES & TYPES
910 DATA 512
920 DATA 47,4, 53,4, 60,4, 53,4
930 DATA 47,4, 47,4, 47,2
940 DATA 53,4, 53,4, 53,2
950 DATA 47,4, 47,4, 40,2
960 DATA 47,4, 53,4, 60,4, 53,4
970 DATA 47,4, 47,4, 47,4, 47,4
980 DATA 53,4, 53,4, 45,4, 53,4
990 DATA 60,1
```

Lots of changes here.

Line 210 reads the *tempo* which is the speed at which the melody is played. The tempo number is in the first DATA statement, line 910. To play faster, use a smaller tempo number; to play slower, use a larger tempo number. In lines 920 through 990, each note has a note number (N) and a type of note (TN) number.

TYPE OF NOTE	TN
whole note (𝅝)	1
half note (𝅗𝅥)	2
quarter note (♩)	4

TEMPO and TN are used in line 410 to compute the time delay:

```
410 TD = TEMPO/TN
```

Time Delay equals TEMPO divided by Type of Note

REMEMBER: To tell the computer to divide, use the "slash" key (/). For detailed information on arithmetic, see Appendix B.

Questions

1. In the program called READ & DATA music, add a line to set the background color of the screen equal to the note number.

   315____________________

2. In the program called PLAY A TUNE #1, add two lines to set the background and border colors as follows:

   Background color = Note number (N)
   Border color = Time delay divided by 16

3. In the program called PLAY A TUNE #2, the TEMPO number in line 910 is 512. Show the time delay (TD) for each type of note.

TYPE OF NOTE	TN	TD
whole note	1	______
half note	2	______
quarter note	4	______

Answers

1. 315 SE. 2,N,8

   Also try: 315 SE. 2,N,N
   315 SE. 2,N,N/16
   and so on — EXPERIMENT!

2. 320 SE. 2,N,8
   330 SE. 4,TD/16,8

3.

TYPE OF NOTE	TN	TD	
whole note	1	512	(512/1 = 512)
half note	2	256	(512/2 = 256)
quarter note	4	128	(512/4 = 128)

SELF-TEST

Take a break. Do something physical and relaxing. Jog, play tennis, stretch, ride a bike, go for a walk. Then browse through this Self-Test to assure yourself that you are learning more and more about BASIC.

1. In BASIC, a number box is identified by a label, or name. What is this label called?____

2. Suppose that the number 7 is in number box A. Then 7 is called the ____ of numeric variable A.

3. Without actually using the computer, complete each of the following. (It's OK to guess!)

(a) You type: F = 96
You type: PRINT F
It prints: _____

(b) You type: KOLOR = 12
You type: PRINT KOLOR
It prints: ____

(c) You type: TD = 50
You type: TD = 100
You type: PRINT TD
It prints: ____

(d) You type: KOLOR = 12
You type: FREQ = 16*KOLOR
You type: PRINT FREQ
It prints: ____

(e) You type: H = 7
You type: PRINT "H"
It prints: ____

4. The line: 20 PR. "WHAT NOTE";: IN. N tells the computer to____________
____

5. Describe what might happen if you run the following program.

```
10 GR. 0
20 PR. "HOW OLD ARE YOU";:IN.AGE
30 SO. 0,AGE,10,10
40 FOR Z=1 TO 200: NEXT Z
50 SO. 0,0,0,0
60 GOTO 20
```

____

6. For each of the following FOR statements, write the sequence of values that the variable A will take on. We have already done the first one.

FOR STATEMENT	SEQUENCE OF VALUES
FOR A = 1 TO 7	1,2,3,4,5,6,7
FOR A = 2 TO 6	
FOR A = 1 TO 7 STEP 2	
FOR A = 0 TO 7 STEP 2	
FOR A = 0 TO 4 STEP 5	
FOR A = 1 TO -1 STEP -1	
The next two are tricky! It's OK to guess.	
FOR A = 5 TO 3	
FOR A = 1 TO 2 STEP 0	

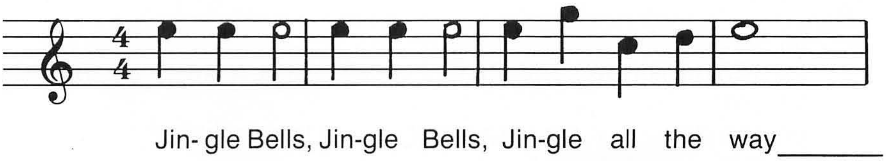
7. For each of the following programs, pretend that you are the computer. Show what will be on your screen when the program is run.

(a)
```
10 GR. 0
20 FOR B=0 TO 10
30    PR. 10-B
40    FOR Z=1 TO 500: NEXT Z
50 NEXT B
60 PR. "BLASTOFF!!!"
```

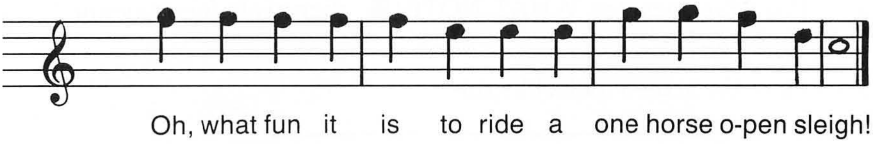
(b)
```
10 GR. 0
20 FOR COUNTDOWN=10 TO 0 STEP-1
30    PR. COUNTDOWN
40    FOR Z=1 TO 500: NEXT Z
50 NEXT COUNTDOWN
60 PR."BLASTOFF!!!"
```

8. Below is the music for *Jingle Bells*. We have added the note letters and numbers.

E	E	E	E	E	E	E	G	C	D	E
47	47	47	47	47	47	47	40	60	53	47

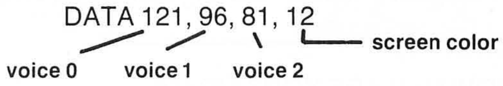


F	F	F	F	F	E	E	E	G	G	F	D	C
45	45	45	45	45	47	47	47	40	40	45	53	60



Put the necessary information into DATA statements for PLAY A TUNE #2 so the computer can play *Jingle Bells*.

9. Write a program to play "color chords." Each chord consists of three notes played by voices 0, 1, and 2. Your program should read the three note numbers and the screen color, then play the chord for a little while using the screen color for the chord. The note numbers and screen color numbers are in DATA statements. For example:



Here are some color chords to play.

```
1000 REM**COLOR CHORDS
1010 DATA 121,96,81,12
1020 DATA 91,72,60,3
1030 DATA 81,64,53,8
```

Add as many more as you want!

Answers to Self-Test

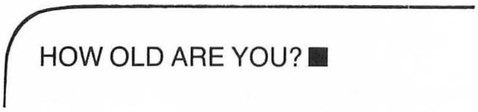
1. A numeric variable

2. value

3. (a) 96
   (b) 12 You can't use COLOR as a variable since it is a special BASIC word.
   (c) 100
   (d) 192 (Since KOLOR is 12, FREQ = 16*12 = 192)
   (e) H Because H is enclosed in quotation marks.

4. Print the message WHAT NOTE?■, then wait for someone to enter the value of N.

5. First, the screen looks like this:



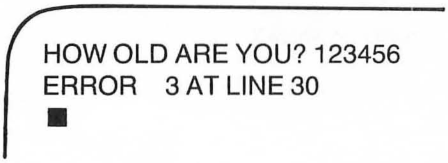
```
HOW OLD ARE YOU? ■
```

If you type your age and press RETURN, the computer uses your age as a note number and plays a note.

Young people will hear high notes.

Older people will hear lower notes.

Venerable dragons will hear nothing and see an error message.



```
HOW OLD ARE YOU? 123456
ERROR    3 AT LINE 30
■
```

You will hear a tone for any number up to 65535.

Numbers from 65536 and up give the error message.

6.

FOR STATEMENT	SEQUENCE OF VALUES
FOR A = 1 TO 7	1,2,3,4,5,6,7
FOR A = 2 TO 6	2,3,4,5,6
FOR A = 1 TO 7 STEP 2	1,3,5,7
FOR A = 0 TO 7 STEP 2	0,2,4,6
FOR A = 0 TO 4 STEP 5	0
FOR A = 1 TO -1 STEP -1	1,0,-1
FOR A = 5 TO 3	5 only (no STEP -1)
FOR A = 0 TO 2 STEP 0	0,0,0,0,etc.

7. Both programs print the same information, as follows:

```
10
9
8
7
6
5
4
3
2
1
0
BLASTOFF!!!
```

8.

```
900 REM**TEMPO, NOTES & TYPES
910 DATA 512
920 DATA 47,4, 47,4, 47,2
930 DATA 47,4, 47,4, 47,2
940 DATA 47,4, 40,4, 60,4, 53,4
950 DATA 47,1
960 DATA 45,4, 45,4, 45,4, 45,4
970 DATA 45,4, 47,4, 47,4, 47,4
980 DATA 40,4, 40,4, 45,4, 53,4
990 DATA 60,1
```

Merry Christmas!

9. We leave this one entirely to you.

Chapter Five

String Boxes

Now you know about number boxes, handy tools for doing things with numbers. Next, put some *string boxes* in your Atari BASIC toolbox.

String boxes are places in the computer's memory that can store strings. String boxes are identified by names, or labels, called *string variables*

In this chapter you will learn how to put strings into string boxes and then use them in many ways.

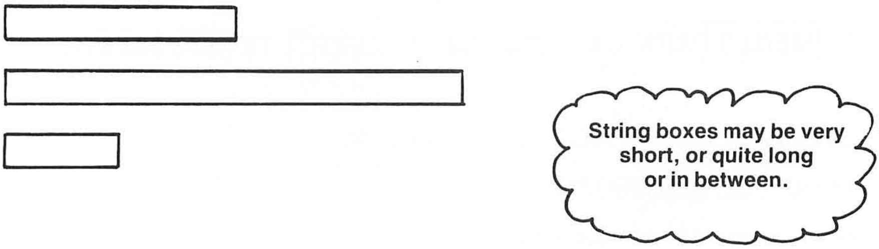
When you finish this chapter, you will be able to:

- Recognize and use string variables
- Assign values to string variables
- Use the DIM statement to reserve memory space for string variables
- Use the INPUT statement to enter values of a string variable from the keyboard
- Use READ and DATA statements to supply values of a string variable
- Write programs using both numeric and string variables
- Use subroutines and the GOSUB and RETURN statements

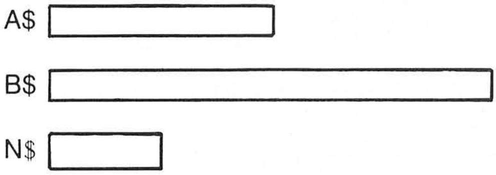
Your power increases! More tools appear in your computer toolbox. You are a step closer to getting the computer to do what *you* want it to do!

STRING BOXES

Put your imagination to work again. This time, imagine that inside the computer are a number of little *string boxes.*


String boxes may be very short, or quite long or in between.

A string box is identified by a label, or name, consisting of a letter followed by a dollar sign. For example, here are boxes called A$, B$, and N$.



Into a string box you can put. . .(suspense, trumpets, fanfare!). . .a *string.*

A string can be a name: GEORGE FIREDRAKE
A string can be a telephone number: 415-323-6117
A string can be a message: TRUST YOUR PSYCHIC TAILWIND
A string can be gibberish: 12BZ$=A

A string may include almost any keyboard character. Quotation marks are sometimes used to enclose a string, as shown below:

"REALITY EXPANDS TO FIT THE AVAILABLE FANTASIES."

REMEMBER: The quotation marks are not part of the string. They *enclose* the string.

"RIDE THE THIRD WAVE!"

This string is enclosed in quotation marks

The labels that identify string boxes (A$, B$, C$, etc.) are called *string variables.* The string in the box is called a *string value.* The string in box A$ is called the *value of A$;* the string in box B$ is called the value of B$; the string in C$ is called the value of C$; and so on.

Below are three string boxes called A$, N$, and T$:

A$ | MENLO PARK, CA 94025 | N$ | GEORGE | T$ | 415 323 6117 |

- The value of A$ is MENLO PARK, CA 94025
- The value of N$ is GEORGE
- The value of T$ is 415 323 6117

In Chapter 4, you learned about number boxes and numeric variables.

These are numeric variables: A B C N Z
These are string variables: A$ B$ C$ N$ Z$

Do you see the difference? A string variable always ends with a dollar sign ($), while a numeric variable does not end in a dollar sign.

Perhaps you remember that a numeric variable may be a letter followed by a digit or even a number of letters and digits. A string variable may also be a group of letters and/or digits, followed by a dollar sign. If so, the first character must be a letter.

These are numeric variables: T1 ABC KOLOR
These are string variables: T1$ ABC$ KOLOR$

Please recall, however, that you can't use a special BASIC word as a variable. So you can't use COLOR, GO or TO. See Appendix F for a list of reserved words.

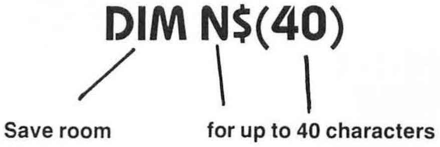
Before using a string variable, you must tell the computer how long the string might be. The length of a string is the number of characters within it, including spaces.

STRING	LENGTH
A	1
AB	2
ABC	3
A B C	5

Use the DIM statement to tell the computer the maximum number of characters you intend to use in a string variable.

You type: DIM N$(50) and press RETURN.
It prints: READY

You have told the computer to reserve room in its memory for values of N$ up to 40 characters long.



DIM N$(40)

Save room

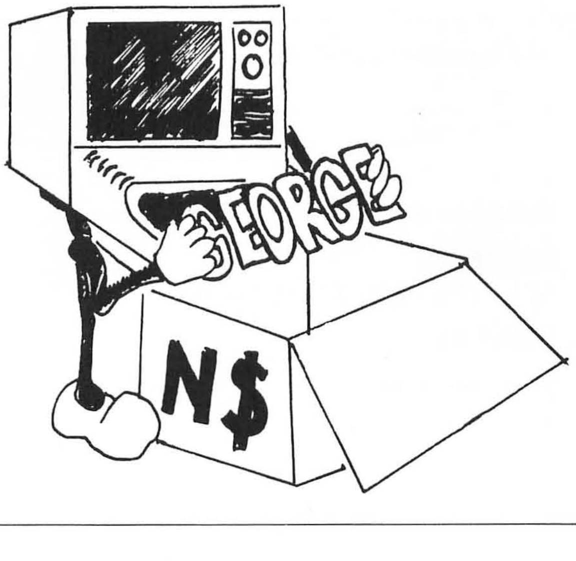
for up to 40 characters

You can reserve room for more than one variable in a single DIM statement. For example:

```
DIM N$(40),V$(30)
```

Here is how you tell the computer to stuff a string into a string box, or, more formally, how to *assign* a string to a string variable. To tell the computer to put the string GEORGE into string box N$, do this:

You type: DIM N$(40)
You type: N$ = "GEORGE"



The computer has put the string GEORGE into string box N$. Remember how you did this.

N$ = "GEORGE"

String variable **String**

Now find out what is in string box N$. Tell the computer to print the value of string variable N$.

You type: PRINT N$
It prints: GEORGE

To tell the computer to print the value of string variable N$, you type:

PRINT N$

IMPORTANT NOTICE: When you use this method to assign a value to a string variable, you must enclose the string in quotation marks.

A$ = "PLEASE ENCLOSE ME IN QUOTATION MARKS"

Questions

1. For each variable, write N if it is a numeric variable, S if it is a string variable, and OOPS! if it is neither a numeric nor string variable.

A$__________ X__________ TD.__________ $ABC__________
NMBR__________ NAME$__________ P2.3__________ 3N$__________

2. Your turn. Complete the following:

You type: DIM A$(30)
You type: A$ = "THE FORCE IS WITH YOU"

You type: PRINT A$
It prints: __________

3. How would you assign the string TIM to the string variable NAME$?

   (a) You type: ____________________

   (b) You type: ____________________

   What would you then type to tell the computer to print the value of NAME$?

   (c) You type: ____________________

   (d) Write commands to assign the string BARBARA to the string variable NAME$ and then to print the value of NAME$.
   You type: ____________________
   You type: ____________________

Answers

1.

A$	S	X	N	TD	N	$ABC	OOPS!
NMBR	N	NAME$	S	P2.3	OOPS!	3N$	OOPS!

2. (a) THE FORCE IS WITH YOU

3. (a) DIM NAME$(20)

   (b) NAME$ = "TIM"

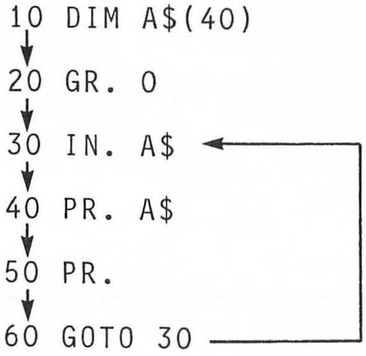
   (c) PRINT NAME$

   (d) NAME$ = "BARBARA"

   PRINT NAME$

USE INPUT TO STUFF STRING BOXES

You can also use the INPUT statement to put strings into string boxes or, as some prefer to say, assign strings as values of string variables.



```
10 DIM A$(40)
20 GR. 0
30 IN. A$
40 PR. A$
50 PR.
60 GOTO 30
```

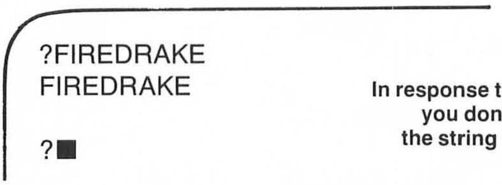
This program simply prints whatever you enter as the value of A$

In following this program, the computer always returns to the INPUT statement, prints a question mark, turns on the cursor and waits patiently for a value of A$.

Store the program and RUN it. It begins this way:

```
?■
```

Type the string FIREDRAKE, without quotation marks.



```
?FIREDRAKE
FIREDRAKE

?■
```

In response to an INPUT statement, you don't have to enclose the string in quotation marks.

EXPERIMENT! Type a string and press RETURN. Try several strings. When you are finished, press BREAK to stop the computer.

You can use the INPUT statement to put a name, a message, graphics, characters, or whatever you want everywhere on the screen.

```
10 DIM MSG$(40)
20 PR. CHR$(125)
30 PR. "MESSAGE";IN. MSG$
40 PR. MSG$;
50 GOTO 40
```

Experiment with this program. Put a name on the screen. Type spaces *before* the name — do they appear? Type spaces after the name — do they appear? Type graphics characters. Add SOUND or SETCOLOR. Experiment!

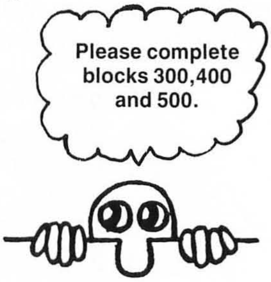
Next, try this program.

```
100 REM**SOUND OFF FOR SOMEONE
110 DIM NAME$(40)
120 GR. 0

200 REM**FOR WHOM?
210 PR. CHR$(125)
220 PR. "YOUR NAME";: IN.NAME$

300 REM**CRESCENDO FOR NAME$
310 PR. CHR$(125)
320 FOR N=255 TO 1 STEP -1
330   PR.NAME$
340   SO. 0,N,10,10
350 NEXT N

400 REM**DO IT AGAIN
410 SO. 0,0,0,0
420 GOTO 210
```

Questions

1. Complete the following program to blink a name, message or any string on and off. If you need help, review the NAME BLINKER section in Chapter 3.

```
100 REM**STRING BLINKER
110 DIM S$(40)
120 GR.0

200 REM**GET THE STRING TO BLINK
210 PR. CHR$(125)
220 PR. "STRING";: IN. S$

300 REM**BLINK S$ ON

400 REM**BLINK S$ OFF

500 REM**GO BLINK AGAIN
```



2. Write a program to "fade-in" and "fade-out" a string someone enters from the keyboard in response to an INPUT statement. If you need help, review the NAME FADER program in Chapter 4.

Answers

1. Here is the rest of the program.

```
300 REM**BLINK S$ ON
310 PR. S$
320 FOR Z=1 TO 500: NEXT Z

400 REM**BLINK S$ OFF
410 PR. CHR$(125)
420 FOR Z=1 TO 500: NEXT Z

500 REM**GO BLINK AGAIN
510 GOTO 310
```

2.

```
100 REM**STRING FADER
110 DIM S$(40)
120 GR. 0
130 PR. "STRING";: IN. S$

200 REM**SET BACKGROUND COLOR
210 SE. 2,0,0
```

Dark gray

```
300 REM**PRINT NAME INVISIBLY
310 SE. 1,0,0
320 PR. S$

400 REM**MAKE NAME SLOWLY VISIBLE
410 FOR L=0 TO 15
420   SE. 1,0,L
430   FOR Z=1 TO 30: NEXT Z
440 NEXT L
```

Fade in

```
500 REM**MAKE NAME SLOWLY INVISIBLE
510 FOR L=15 TO 0 STEP -1
520   SE. 1,0,L
530   FOR Z=1 TO 30: NEXT Z
540 NEXT L
```

Fade out

```
600 REM**GO DO IT AGAIN
610 GOTO 410
```

Line 120 clears the screen to its usual GR. 0 blue.
Also try these variations:

- Keep the foregound luminance constant and fade the background luminance. Change the program as follows.

```
420 SE. 2,0,L
520 SE. 2,0,L
```

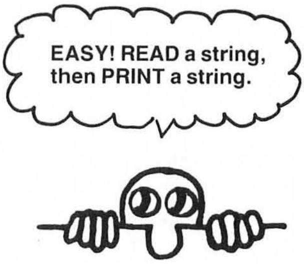
Instead of gray, choose your own screen color.

- Use an INPUT statement to obtain the screen color.

USE READ AND DATA TO STUFF STRING BOXES

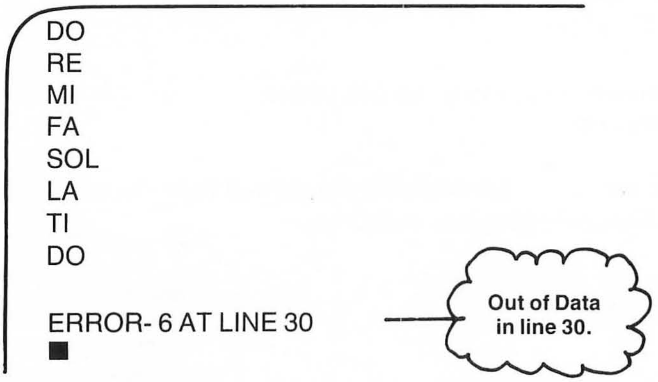
You can use READ and DATA statements to stuff strings into string boxes. First try this program.

```
10 DIM N$(20)
20 GR. 0
30 READ N$
40 PRINT N$
50 GOTO 30
60 DATA DO,RE,ME,FA
70 DATA SOL,LA,TI,DO
```



The statement: **30 READ N$** tells the computer to read one string from a DATA list and put the string into box N$ (assign the string as the value of N$). Every time line 30 is executed, the *next* string in the DATA list is read and assigned as the value of N$.

This is how the screen looks when you RUN the program:



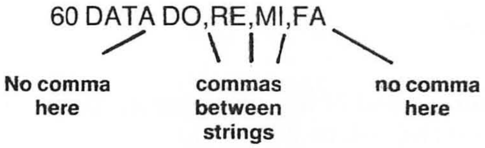
```
DO
RE
MI
FA
SOL
LA
TI
DO

ERROR- 6 AT LINE 30
■
```

The computer read and printed all the values in the DATA statement, then tried to read yet another. Alas, there were no more strings to be read, so the computer printed an error message. That's OK, since it was finished doing what we wanted it to do.

Look at the DATA statements in lines 60 and 70:

```
60 DATA DO,RE,MI,FA
70 DATA SOL,LA,TI,DO
```

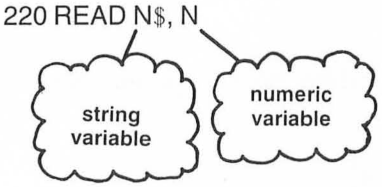
The two DATA statements contain eight strings. We use commas *between* strings:



Hmmm. . .who said, "It all begins with do, re, mi?" Try this program — see what you hear and hear what you see.

```
100 REM**DO RE MI
110 DIM N$(20)
120 GR. 0

200 REM**READ NOTE & NOTE NUMBER
210 PR CHR$(125)
220 READ N$,N

300 REM**SHOW & PLAY NOTE
310 PR. N$
320 SO. 0,N,10,10
330 FOR Z=1 TO 400: NEXT Z
340 SO.0,0,0,0

400 REM**GO FOR ANOTHER
410 GOTO 210

1000 REM**NOTES & NOTE NUMBERS
1010 DATA DO,121
1020 DATA RE,108
1030 DATA MI,96
1040 DATA FA,91
1050 DATA SOL,81
1060 DATA LA,72
1070 DATA TI,64
1080 DATA DO,60
```

The READ statement (line 220) tells the computer to read a value for a string variable and a value for a numeric variable:



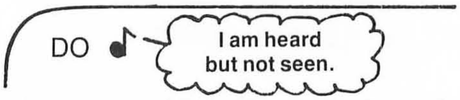
```
220 READ N$, N
```

We put the eight pairs of values for N$ and N in eight separate DATA statements. To save time and space, we could have done it this way:

```
1010 DATA DO,121,RE,108,MI,96,FA,91
1020 DATA SOL,81,LA,72,TI,64,DO,60
```

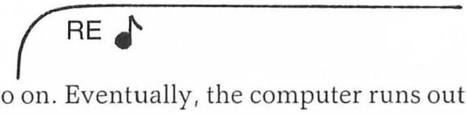
You can even put all the information in *one* DATA statement!

RUN the program. First you see DO on the screen and hear the musical tone for DO in the scale of C.



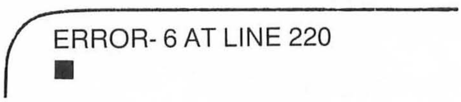
```
DO ♪
```

When you see RE and hear the musical tone for RE.



```
RE ♪
```

And so on. Eventually, the computer runs out of music and you see:



```
ERROR- 6 AT LINE 220
■
```

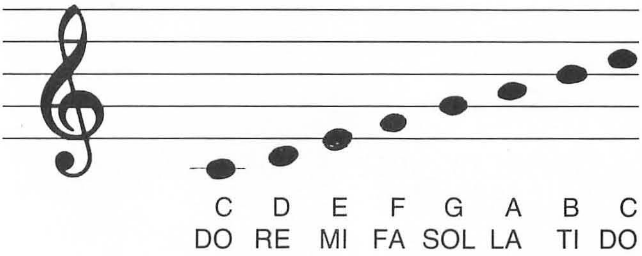
Mix up the DO, RE, MI's. Write your own DATA statements, as many as you want. Change the time delay in line 330. Make it longer. Close your eyes. When you hear the sound, guess. Is it DO, RE, MI,. . .? Open your eyes and look at the screen. Are you correct? To practice your DO, RE, MI's (in the scale of C), try this variation:

```
100 REM**DO RE MI
110 DIM N$(20)
120 GR. 0

200 REM**READ NOTE & NOTE NUMBER
210 PR. CHR$(125)
220 READ N$,N

300 REM**PLAY THE NOTE
310 SO. 0,N,10,10
320 FOR Z=1 TO 400: NEXT Z
330 SO. 0,0,0,0

400 REM**WAIT, THEN TELL WHICH NOTE
410 FOR Z=1 TO 1000: NEXT Z
420 PRINT N$

500 REM**GO FOR ANOTHER
510 GOTO 210

1000 REM**NOTES & NOTE NUMBERS
1010 DATA MI,96, TI,64, TI,64, DO,121
1020 DATA LA,72, DO,60, SOL,81, FA,91
1030 DATA MI,96, RE,108, FA,91, SOL,81
1040 DATA TI,64, DO,60, MI,96, LA,72
1050 DATA FA,91, DO,121, TI,64, TI,64
   and so on.
```

Now you can keep your eyes open and watch the screen. First you hear the note. Guess — is it DO, RE, MI. . .? Then you see the answer. Want more time to answer? Change the time delay in line 410.

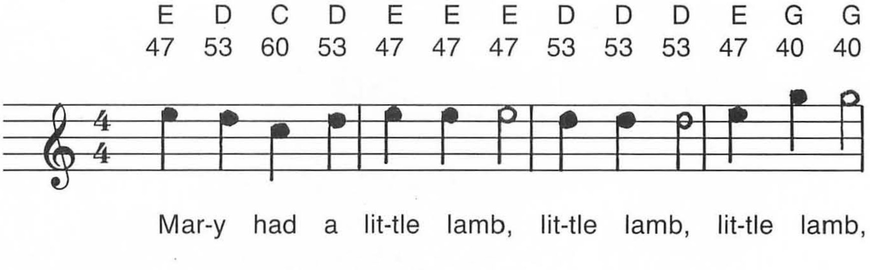
Questions

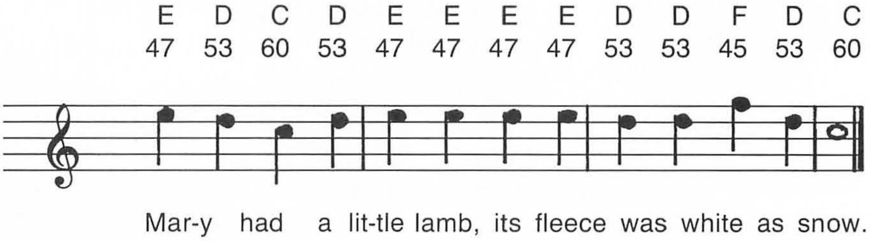
1. Instead of DO, RE, MI, put the letters for the notes on the screen. Here are both for the scale of C.



To do this, you need change only the DATA statements in the DO, RE, MI program.

2. Here again are the words and music for *Mary Had a Little Lamb:*

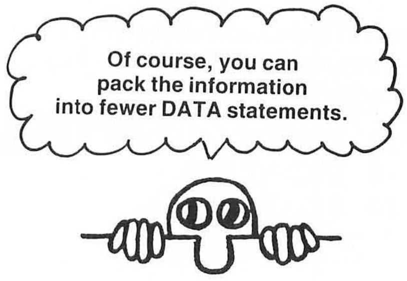

E D C D E E E D D D E G G
47 53 60 53 47 47 47 53 53 53 47 40 40
4
4
Mar-y had a lit-tle lamb, lit-tle lamb, lit-tle lamb,


E D C D E E E E D D F D C
47 53 60 53 47 47 47 47 53 53 45 53 60
Mar-y had a lit-tle lamb, its fleece was white as snow.

Modify the DO, RE, MI program to play *Mary Had a Little Lamb* so the syllable or word for each note appears on the screen while the note is played. First *Mar* then-*y* then *had* and so on.

Answers

1.
```
1010 DATA C, 121
1020 DATA D, 108
1030 DATA E, 96
1040 DATA F, 91
1050 DATA F, 81
1060 DATA A, 72
1070 DATA B, 64
1080 DATA C, 60
```


Of course, you can pack the information into fewer DATA statements.

2. Did you remember to include the *duration* of each note on the type of note (quarter, half, whole)?

```
100 REM**WORDS & MUSIC
110 DIM WORD$(20)
120 GR. 0

200 REM**READ TEMPO (HOW FAST TO PLAY)
210 READ TEMPO

300 REM**READ WORD$, NOTE, TYPE OF NOTE
310 READ WORD$,N,TN

400 REM**SHOW WORD$, PLAY NOTE
410 PR. CHR$(125)
420 PR. WORD$
430 SO. 0,N,10,10
440 FOR Z=1 TO TEMPO/TN: NEXT Z
450 SO. 0,0,0,0

500 REM**GO FOR ANOTHER
510 GOTO 310

1000 REM**TEMPO,WORD$,NOTE, TYPE NOTE
1010 DATA 512
1020 DATA MAR,47,4,Y,53,4,HAD,60,4,A,53,4
1030 DATA LIT,47,4,TLE,47,4,LAMB,47,2
1040 DATA LIT,53,4,TLE,53,4,LAMB,53,2
1050 DATA LIT,47,4,TLE,40,4,LAMB,40,2
1060 DATA MAR,47,4,Y,53,4,HAD,60,4, A,53,4
1070 DATA LIT,47,4,TLE,47,4,LAMB,47,4,ITS,47,4
1080 DATA FLEECE,53,4,WAS,53,4,WHITE,45,4,AS,53,4
1090 DATA SNOW,60,1
```

SUBROUTINES MAKE IT EASIER

Imagine your ATARI computer with a giant TV screen. It's the big game of the year and your computer is firing up the Rooter Club — building energy to help your team win the game.

```
100 REM**GO TEAM GO!!!
110 GR. 0
120 SE. 1,0,0


200 REM**'GO' ON A BLUE SCREEN
210 SE. 2,8,8
220 PR. CHR$(125): PR. "GO"
230 FOR Z=1 TO 500: NEXT Z

300 REM**'TEAM' ON AN ORANGE SCREEN
310 SE. 2,1,8
320 PR. CHR$(125): PR. "TEAM"
330 FOR Z=1 TO 500: NEXT Z

400 REM**'GO!!!' ON A GREEN SCREEN
410 SE. 2,12,8
420 PR. CHR$(125): PR. "GO!!!"
430 FOR Z=1 TO 500: NEXT Z

500 REM**KEEP IT GOING
510 GOTO 210
```

Three time delays in that last program. Aha! A splendid opportunity to introduce. . .(fanfare!). . .*subroutines.* Write the time delay once as a subroutine, then use it as often as needed.

The following program has a *time delay subroutine* beginning at line 900. This subroutine is used, or *called,* in lines 230, 330, and 430.

```
100 REM**GO TEAM GO!!!
110 GR. 0
120 SE. 1,0,0

200 REM**'GO' ON A BLUE SCREEN
210 SE. 2,8,8
220 PR. CHR$(125): PR. "GO"
230 GOSUB 910 ———— Use time delay subroutine
```

(continued)

```
300 REM**'TEAM' ON AN ORANGE SCREEN
310 SE. 2,1,8
320 PR. CHR$(125): PR. "TEAM"
330 GOSUB 910 ————————— Use time delay subroutine

400 REM**'GO!!!' ON A GREEN SCREEN
410 SE. 2,12,8
420 PR. CHR$(125): PR. "GO!!!"
430 GOSUB 910————————— Use time delay subroutine

500 REM**KEEP IT GOING
510 GOTO 210

900 REM**TIME DELAY SUBROUTINE
910 FOR Z=1 TO 500: NEXT Z
920 RETURN
```

The time delay subroutine is called (used) by the GOSUB 910 statement in lines 230, 330, and 430. Obedient as always, the computer goes to line 910, does it, then moves on to line 920. Aha! Line 920 says RETURN. So, the computer RETURNs to the statement following the GOSUB 910, and continues from there. Clever!

A subroutine is a helper. You call it when you need it. It does what it is designed to do, then RETURNs to the statement following the GOSUB that called it.

We use a slightly different time delay subroutine in a program to blink a name or message on and off:

```
100 REM**STRING BLINKER
110 GR. 0
120 DIM MSG$(40)
130 SE. 1,0,0

200 REM**WHAT TO BLINK?
210 PR. CHR$(125)
220 PR. "WHAT SHALL I BLINK?";
230 IN. MSG$

300 REM**MESSAGE ON
310 PR. CHR$(125)
320 PR. MSG$
330 GOSUB 910
```

(continued)

```
400 REM**MESSAGE OFF
410 PR. CHR$(125)
420 GOSUB 910

500 REM**GO BLINK
510 GOTO 310

900 REM**TIME DELAY SUBROUTINE
910 TD = 1000
920 FOR Z=1 TO TD: NEXT Z
930 RETURN
```

The time delay subroutine is called in lines 330 and 420. Line 910 in the subroutine sets the amount of delay. To slow down or speed up the blinking, change line 910. For example:

Faster blinking: 910 TD = 500
Slower blinking: 910 TD = 2000

Next, a program to blink a message with two sounds — one sound when the message is on and another when the message is off:

```
100 REM**STRING BLINKER WITH SOUND
110 GR. 0
120 DIM MSG$(40)
130 SE. 1,0,0

200 REM**WHAT TO BLINK?
210 PR. CHR$(125)
220 PR. "WHAT SHALL I BLINK";
230 IN. MSG$
```

Blocks 100 and 200 are just like STRING BLINKER, except for a slight change in line 100.

```
300 REM**MESSAGE ON WITH SOUND #1
310 PR. CHR$(125)
320 PR. MSG$
330 N = 60: TD = 500
340 GOSUB 810
```

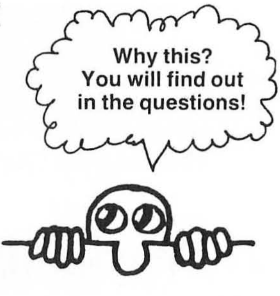
Set the note and time delay, then GOSUB and play.

```
400 REM**MESSAGE OFF WITH SOUND #2
410 PR. CHR$(125)
420 N = 121: TD = 500
430 GOSUB 810
```

Set the note and time delay, then GOSUB and play.

(continued)

```
500 REM**GO BLINK AGAIN
510 GOTO 310

800 REM**PLAY A NOTE SUBROUTINE
810 SO. 0,N,10,10
820 FOR Z=1 TO TD: NEXT Z
830 SO. 0,0,0,0
840 RETURN

900 REM**TIME DELAY SUBROUTINE
910 FOR Z=1 TO TD: NEXT Z
920 RETURN
```



EXPERIMENT! Change lines 330 and 420 to get the two notes and durations (time delays) that sound good to you. Try very short or very long time delays. Try notes close together or far apart.

Questions

Our STRING BLINKER WITH SOUND program contains an unused TIME DELAY SUBROUTINE in block 900. Let's use it now.

1. Change the program so you hear sound # 1 when the name is on and silence when the name is off. You need change only two lines.
2. Change the PLAY A NOTE SUBROUTINE so it calls (uses) the TIME DELAY SUBROUTINE. Yes, a subroutine can call a subroutine! You need change only one line in block 800.

Answers

1. Change lines 420 and 430 to instruct the computer to call the TIME DELAY SUBROUTINE instead of the PLAY A NOTE SUBROUTINE.

```
420 TD = 500
430 GOSUB 910
```

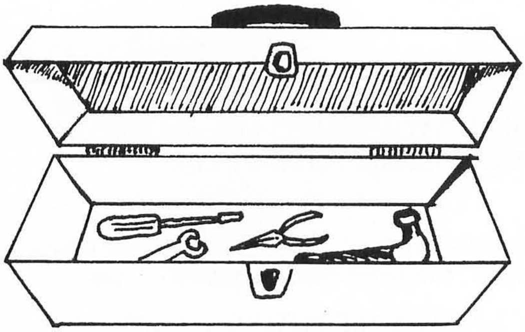
2. Replace the time delay in line 820 with a GOSUB to call the TIME DELAY SUBROUTINE.

```
820 GOSUB 910
```

The RETURN statement in line 920 sends the computer back to line 830; the computer shuts off the sound, goes on to line 840, and RETURNs to the line following the GOSUB that called the PLAY A NOTE SUBROUTINE.

PROGRAMMER'S TOOLBOX

Subroutines are handy tools. As you work and play through this book, you will acquire more tools for your *Programmer's Toolbox.*



In this chapter, we used four subroutines: three time delay subroutines and one subroutine to play a note. Here they are without line numbers. You may add line numbers, then write GOSUBs to use them.

```
REM**TIME DELAY SUBROUTINE
FOR Z=1 TO 500: NEXT Z
RETURN
```

```
REM**TIME DELAY SUBROUTINE
TD=500
FOR Z=1 TO TD: NEXT Z
RETURN
```

```
REM**TIME DELAY SUBROUTINE
FORZ=1 TO TD: NEXT Z
RETURN
```

```
REM**PLAY A NOTE SUBROUTINE
SO. 0,N,10,10
FOR Z=1 TO TD: NEXT Z
SO. 0,0,0,0
RETURN
```

SELF-TEST

Congratulations! You have completed another chapter. Undoubtedly, you now have the stringth (oops) to twine your way through this Self-Test.

1. What is a string?______________________________

____

2. What is a "string box?"__________________________

____

3. What is a string variable?________________________

____

4. What is the *value* of a string variable?_______________

____

5. Write line 110 to tell the computer to reserve 30 character positions for S$, 20 positions for V$, and 40 positions for P$.

   110____

6. Complete the following. Assume all variables have appeared in DIM statements.

   (a) You type: Z$ = "ZZZZZ"
   You type: PR. Z$
   It prints: ________

   (b) You type: A$ = "DO RE MI"
   You type: B$ = A$
   You type: PR. B$
   It prints: ________

   (c) You type: S$ = "THE DRAGON   "
   You type: V$ = "ATE "
   You type: P$ = "THE LAZY KNIGHT."
   You type: PR.S$;V$;P$
   It prints: ____

7. Describe what the INPUT statement does.

   ____

   ____

8. Describe how the READ and DATA statements work together.

   ____

   ____

9. On page 113, we showed you a program called SOUND OFF FOR SOMEONE. After you enter someone's name, you see the name on the screen and hear a crescendo of notes from the lowest note (255) to the highest note (1). Please make a few changes to the program.

   Make the letters in the name very dark.

   Ask for the screen border and background colors. Use these colors during the crescendo.

   ____

   ____

   ____

   ____

   ____

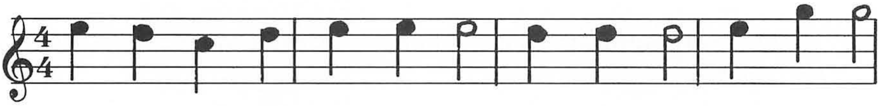
   ____

10. Write a program to play *Mary Had a Little Lamb* using the following screen colors for notes of the song.

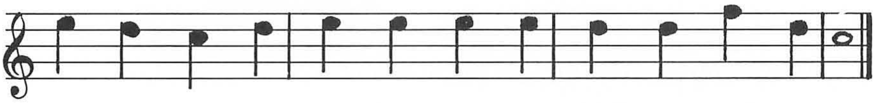
Note	40	45	47	53	60
Color	pink	red-orange	green	blue	purple
Color Number	4	3	12	7	5

Put all necessary information to play the song in DATA statements. Here, again, are the words and music:

E	D	C	D	E	E	E	D	D	D	E	G	G
47	53	60	53	47	47	47	53	53	53	47	40	40



E	D	C	D	E	E	E	E	D	D	F	D	C
47	53	60	53	47	47	47	47	53	53	45	53	60



11. What are subroutines and how do you use them?

12. Change our STRING BLINKER WITH SOUND (page 124) so that the screen is red-orange when the message is on-screen and green when the screen is blank. Do this by changing only lines 330, 420, and the PLAY A NOTE SUBROUTINE.

```
330 ______________
420 ______________
800 REM**PLAY A NOTE SUBROUTINE
810 ______________
820 ______________
830 ______________
840 ______________
850 ______________
```

13. Write a subroutine to play a chord with three notes. Use the following variables:

N0	Note number for voice 0.
N1	Note number for voice 1.
N2	Note number for voice 2.
TD	Time duration.

To use your subroutine, we might write:

```
340 N0=121: N1=96: N2=60: TD=128
350 GOSUB ________
              \
               \
```

line number of <u>your</u> subroutine

Answers to Self-Test

1. A bunch of keyboard characters, typed one after the other. A string can include letters A to Z, numerals 0 to 9, punctuation symbols (.,;etc.), special symbols (# $%etc.), and graphics characters.

2. A place in the computer's memory (RAM) that can hold, or store, a string.

3. The name of a string box. A string variable may be a letter followed by a dollar sign (A$), a letter and a digit followed by a dollar sign (N3$), or a letter followed by letters and/or digits and a dollar sign (AB$, S123$, NAME$).

4. The value of a string variable is the string assigned to the string variable. (frame 3)

5. 110 DIM S$(30), V$(20), P$(40)

6. (a) ZZZZZ (b) DO RE MI

   (c) THE DRAGON ATE THE LAZY KNIGHT.

7. The computer will print a question mark, turn on the cursor, and wait for someone to enter a value or values for the variable or variables in the INPUT statement. When the required information is entered the computer moves on to the next statement.

8. The READ statement will read one value from a DATA statement for each variable in the READ statement(s). The computer will keep track of which values have been read, so each time it comes to a READ statement it will read the next available value or values from the DATA statement(s). If there are no more values, an "error" message is printed — it means OUT OF DATA.

9. We did it this way:

```
100 REM**SOUND OFF FOR SOMEONE
110 DIM NAME$(40)
120 SE. 1,0,0

200 REM**FOR WHOM?
210 GR. 0
220 PR. "YOUR NAME";: IN. NAME$
230 PR. "BORDER (0 TO 15)";: IN. BR
240 PR. "BACKGROUND (0 TO 15)";: IN. BK
250 SE. 4,BR,8
260 SE. 2,BK,8
```

(continued)

```
300 REM**CRESCENDO FOR NAME$
310 PR. CHR$(125)
320 FOR N=255 TO 1 STEP -1
330   PR. NAME$
340   SO. 0,N,10,10
350 NEXT N
360 SO. 0,0,0,0

400 REM**DO IT AGAIN
410 GOTO 210
```

10.

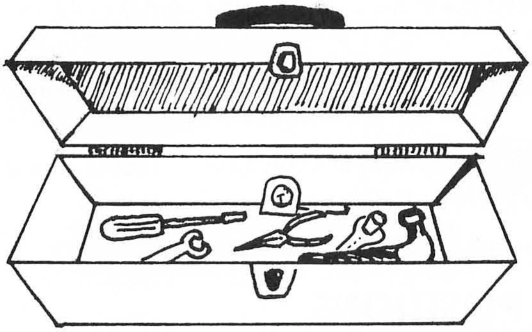
```
100 REM**MUSIC WITH COLOR
110 GR. 0

200 REM**READ TEMPO (HOW FAST TO PLAY)
210 READ TEMPO

300 REM**READ NOTE,TYPE OF NOTE,&COLOR
310 READ N,TN,C

400 REM**PLAY A COLOR NOTE
410 SE. 2,C,8
420 SO. 0,N,10,10
430 FOR Z=1 TO TEMPO/TN: NEXT Z
440 SO. 0,0,0,0

500 REM**GO PLAY AGAIN
510 GOTO 310

1000 REM**TEMPO, N, TN, C
1010 DATA 512
1020 DATA 47,4,12,53,4,7,60,4,5,53,4,7
1030 DATA 47,4,12,47,4,12,47,2,12
1040 DATA 53,4,7,53,4,7,53,2,7
1050 DATA 47,4,12,40,4,4,40,2,4
1060 DATA 47,4,12,53,4,7,60,4,5,53,4,7
1070 DATA 47,4,12,47,4,12,47,4,12,47,4,12
1080 DATA 53,4,7,53,4,7,45,4,3,53,4,7
1090 DATA 60,1,5
```

11. A subroutine is a block of lines (statements) that may be *called* by a GOSUB statement. The last statement in a subroutine is a RETURN statement which sends the computer to the statement following the GOSUB that called the subroutine.

12.
```
300 N = 60: TD = 500: C = 3
420 N = 121: TD = 500: C = 12

800 REM**PLAY A COLOR NOTE SUBROUTINE
810 SE. 2,C,8
820 SO. 0,N,10,10
830 FOR Z=1 TO TD: NEXT Z
840 SO. 0,0,0,0
850 RETURN
```

13. As usual, we leave one or more questions without answers. Good luck on this one.

Before you move on to Chapter 6, add the subroutine in question 12 and the one you wrote for question 13 to your Programmer's Toolbox.



Hmmm. . .why not start a 3x5 card file of subroutines and other tools?

Chapter Six

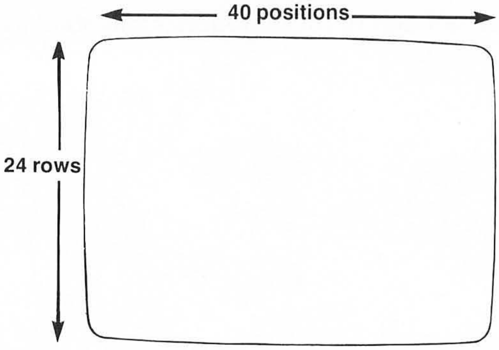
Skipping Around the Screen

In this chapter, you will learn more about controlling and using the screen. You will learn how to tell the computer to print information anywhere on the screen, blink it on or off and move it around. When you finish this chapter, you will be able to:

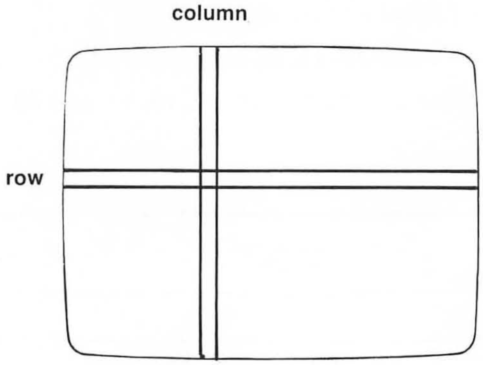
- Interpret screen positions as rows numbered 0 to 23, and columns numbered 0 to 39
- Use the POSITION statement to position information anywhere on the screen
- Write programs to do simple animation — that is, cause things to move from place to place on the screen
- Use POKE to turn the cursor on or off

SCREEN POSITIONS

In GRAPHICS 0, think of the screen as having 24 rows with 40 screen positions in each row:



Rows are numbered from 0 (top of screen) to 23 (bottom of screen). Positions across the screen are numbered from 0 (left of screen) to 39 (right of screen). We will refer to the positions across the screen as *columns*, numbered from 0 to 39.



IMPORTANT NOTICE! Some television sets will not display columns 0 and 1. Therefore, we will do most of our work in columns 2 through 39.

Explore the following screen map. See Appendix D for blank screen maps.

	0	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	2	-	-	-	-	-	-	-	-	-	3	-	-	-	-	-	-	-	-	-
	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9
0			A								H	E	R	E																										
1																																								
2																																								
3																																								
4																																								
5																	T	H	E	R	E																			
6													E																											
7													R																											
8													E																											
9													H																											
10						R	O	W					W																		C									
11													O																		O									
12													N																		L									
13																						*									U									
14																															M									
15																															N									
16																																								
17																																								
18																																								
19																																								
20																																								
21																																								
22																																								
23																																								Z

The word ROW is in row 10.

The word COLUMN is in column 30.

The letter A is in column 2, row 0.

There is a lonely star (*) in column 21, row 13.

The word HERE is in columns 10, 11, 12, and 13, row 0.

The word THERE is in row 5, columns, 16, 17, 18, 19, and 20.

Questions

1. Where is the letter Z?____________________________

2. Where is NOWHERE?____________________________

Answers

1. Column 39, row 23

2. Column 12, rows 12, 11, 10, 9, 8, 7, and 6. Hmmm...we suspect that, at first, you thought NOWHERE was, well, nowhere.

YOU CAN PRINT ALMOST ANYWHERE

It is easy to tell the computer where you want to print something on the screen.

- Use the POSITION statement to tell the computer *where* to print.
- Then use the PRINT statement to tell the computer *what* to print.

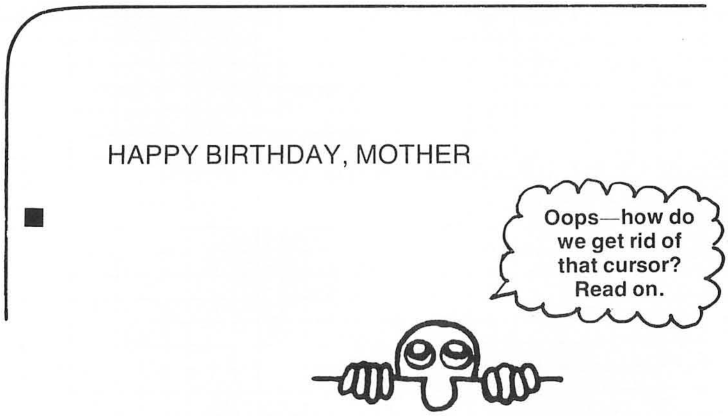
When you want to print something you must tell the computer the column and row. And, of course, you must tell the computer what to print. Here is a short program to print a birthday message for mother near the center of the screen.

```
100 REM**HAPPY BIRTHDAY, MOTHER
110 GR. 0

200 REM**WHERE TO PRINT
210 POSITION 8,12
```

(continued)

```
300 REM**PRINT THE MESSAGE
310 PR. "HAPPY BIRTHDAY, MOTHER"

400 REM**DO NOTHING LOOP
410 GOTO 410
```

Go ahead — try it. You will see this:

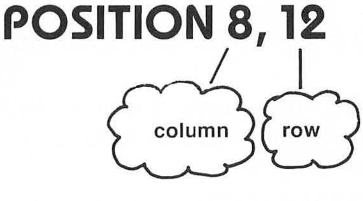


```
HAPPY BIRTHDAY, MOTHER
■
```

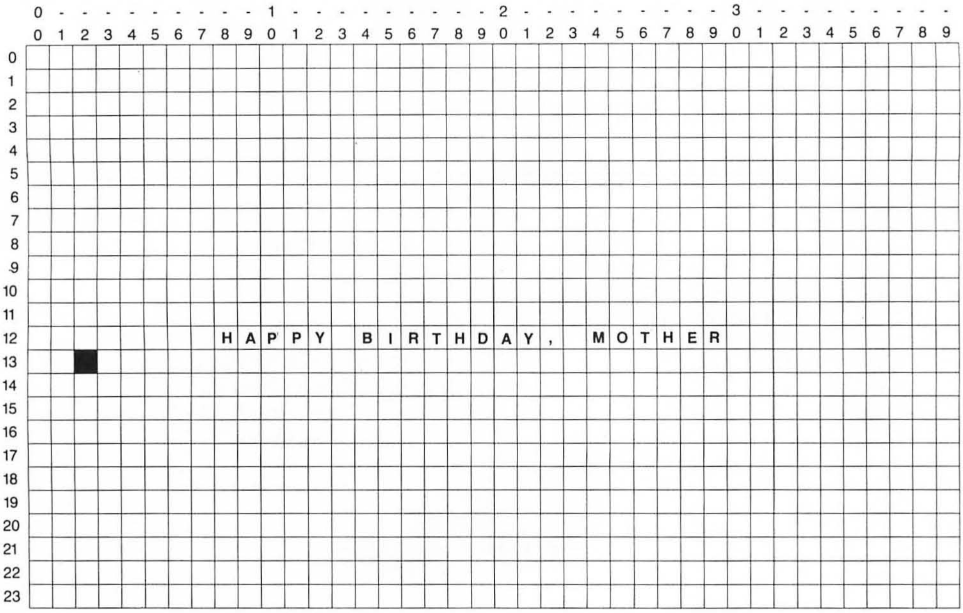
Just what we wanted, except for the cursor cluttering up our electronic birthday card. Soon, we will tell you how to make the cursor disappear. But first, more about line 210:

The line: 210 POSITION 8,12

Will tell the computer to get ready to do something at column 8, row 12 on the screen.



And what do we do at that position? Print the birthday message, of course (line 310).



There is our message, neatly centered on the screen at column 8, row 12. Unfortunately, the cursor is also on the screen at column 2, row 9. Why column 2? Some TV's will not show columns 0 and 1, so the computer will set the left margin at column 2. Later, we will show you how to change the margins.

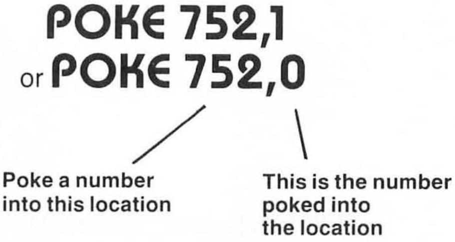
How do we get rid of that pesky cursor? And, if we make it disappear, how do we get it back?

- Press SYSTEM RESET
- Type POKE 752,1 and press RETURN. Poof! The cursor disappears.
- Type POKE 752,0 and press RETURN. The cursor reappears.

Your ATARI computer has thousands of *memory locations*. Each memory location has a numerical address that identifies it. Each memory location can store a number in the range 0 to 255.

Memory location 752 holds a number that tells the computer to show or not show the cursor on the screen. When the number zero (0) is in location 752, the computer shows the cursor. When the number one (1) is in location 752, the computer does not show the cursor.

You use the POKE command to store the number you want (0 or 1) in location 752.



POKE 752,1

or **POKE 752,0**

Poke a number into this location

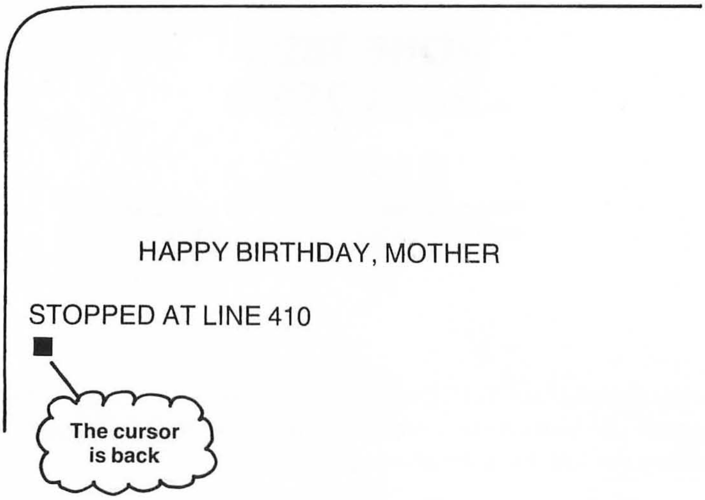
This is the number poked into the location

We will occassionally use POKE to do special things. Let's use POKE in our birthday message program. While we are at it, we will also change the screen colors and darken the letters in the message.

```
100 REM**HAPPY BIRTHDAY, MOTHER
110 GR. 0
120 POKE 752,1 ———— Begone, cursor!

200 REM**SCREEN COLORS
210 SE. 1,0,0
220 SE. 2,5,12
230 SE. 4,10,8

300 REM**PUT MESSAGE AT 8,12
310 POSITION 8,12
320 PR. "HAPPY BIRTHDAY, MOTHER"

400 REM**DO NOTHING LOOP
410 GOTO 410
```

Run this program. You will see HAPPY BIRTHDAY, MOTHER on a violet (light purple) screen with a turquoise border. You will not see the cursor because line 120 pokes it away.

Press the BREAK key.



```
        HAPPY BIRTHDAY, MOTHER

STOPPED AT LINE 410
■
```

Pressing BREAK or SYSTEM RESET returns the cursor to the screen. Try it both ways: run the program and press BREAK to stop it; run the program and press SYSTEM RESET to stop.

EXPERIMENT! Here is a program you can use to print whatever you want, wherever you want it. In this program, we use the abbreviation for POSITION:

POS. is the abbreviation for **POSITION**

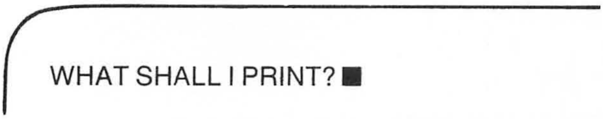
Look for POS. in line 320.

```
100 REM**PRINT WHAT WHERE
110 GR. 0
120 DIM MSG$(40)

200 REM**FIND OUT WHAT & WHERE
210 POKE 752,0
220 PR. CHR$(125)
230 PR. "WHAT SHALL I PRINT";:IN. MSG$
240 PR. "IN WHAT COLUMN    ";:IN. COL
250 PR. "IN WHAT ROW       ";:IN. ROW
```

```
300 REM**PRINT MSG$ AT COL,ROW
310 POKE 752,1
320 POS. COL,ROW
330 PR. MSG$

400 REM**LONG TIME DELAY
410 TD = 3000
420 FOR Z=1 TO TD: NEXT Z

500 REM**GO FOR ANOTHER MESSAGE
510 GOTO 210
```

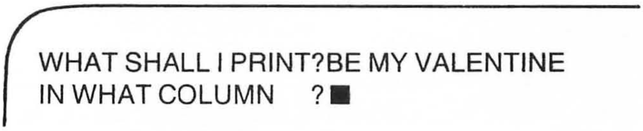
Line 210 turns the cursor ON so you see it while entering information. Line 310 turns the cursor OFF so you don't see it while the message is on the screen. The time delay is a few seconds — shorten it if you wish.

Enter and run the program. First you see:



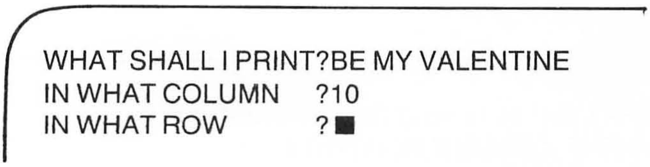
```
WHAT SHALL I PRINT?■
```

Type your message and press RETURN.



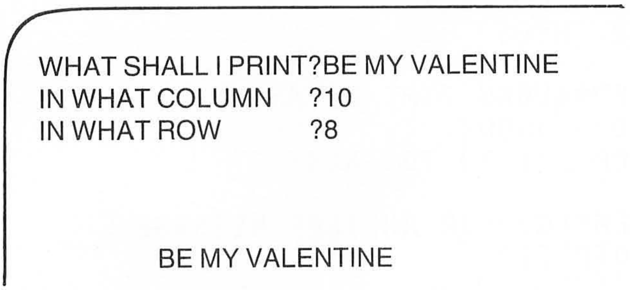
```
WHAT SHALL I PRINT?BE MY VALENTINE
IN WHAT COLUMN    ?■
```

How about column 10? Type **10** and press RETURN.



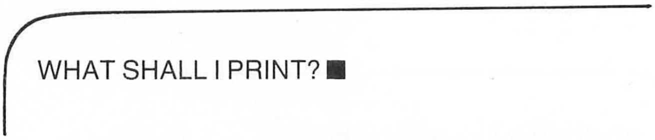
```
WHAT SHALL I PRINT?BE MY VALENTINE
IN WHAT COLUMN    ?10
IN WHAT ROW       ?■
```

Oh, about a third of the way down the screen, in row 8.



```
WHAT SHALL I PRINT?BE MY VALENTINE
IN WHAT COLUMN   ?10
IN WHAT ROW      ?8



            BE MY VALENTINE
```

After a few seconds, you will again see:

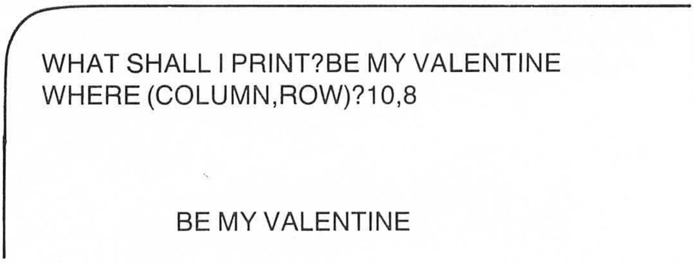


```
WHAT SHALL I PRINT? ■
```

Here is another way to write block 200 of the program.

```
200 REM**FIND OUT WHAT & WHERE
210 POKE 752,0
220 PR. CHR$(125)
230 PR. "WHAT SHALL I PRINT";:IN.MSG$
240 PR. "WHERE(COLUMN,ROW) ";:IN.COL,ROW
```

Line 240 asks for both the column and row.



```
WHAT SHALL I PRINT?BE MY VALENTINE
WHERE (COLUMN,ROW)?10,8



            BE MY VALENTINE
```

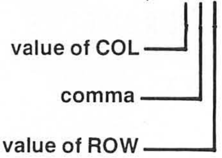
Line 240 asks for *two* things, COL and ROW.

```
240 PR. "WHERE (COLUMN,ROW)";:IN. COL,ROW
```

Must be a
comma here

When you enter the values of COL and ROW, separate them with a comma.

```
WHERE (COLUMN,ROW)?10,8
```



Questions

1. Mother would be pleased to see her happy birthday message blinking on the screen. Write a program to blink HAPPY BIRTHDAY, MOTHER without the cursor showing.
2. Write a program to blink a message anywhere on the screen. The message and the place to blink it are entered in response to INPUT statements. Show *only*the blinking message, nothing else.

Answers

1. Easy! We can use most of out second HAPPY BIRTHDAY, MOTHER program.

```
100 REM** BLINKING BIRTHDAY MESSAGE
110 GR. 0
120 POKE 752,1

200 REM**SCREEN COLORS
210 SE. 1,0,0
220 SE. 2,5,12
230 SE. 4,10,8
```

(continued)

```
300 REM**PUT MESSAGE AT 8,12
310 POS. 8,12
320 PR. "HAPPY BIRTHDAY, MOTHER"
330 GOSUB 910

400 REM**ERASE THE MESSAGE
410 PR. CHR$(125)
420 GOSUB 910

500 REM**GO BLINK AGAIN
510 GOTO 310

900 REM**TIME DELAY SUBROUTINE
910 TD = 500
920 FOR Z=1 TO TD: NEXT Z
930 RETURN
```

2. Another easy one. We can use most of our PRINT WHAT WHERE program.

```
100 REM**BLINK WHAT WHERE
110 GR. 0
120 DIM MSG$(40)

200 REM**FIND OUT WHAT & WHERE
210 POKE 752,0
220 PR. CHR$(125)
230 PR. "WHAT SHALL I PRINT";:IN. MSG$
240 PR. "WHERE (COLUMN,ROW)";:IN. COL,ROW

300 REM**PRINT MSG$ AT COL,ROW
310 POKE 752,1
320 POS. COL,ROW
330 PR. MSG$
340 GOSUB 910

400 REM**ERASE THE MESSAGE
410 PR. CHR$(125)
420 GOSUB 910
```

(continued)

```
500 REM**GO BLINK AGAIN
510 GOTO 310

900 REM**TIME DELAY SUBROUTINE
910 TD = 500
920 FOR Z=1 TO TD: NEXT Z
930 RETURN
```

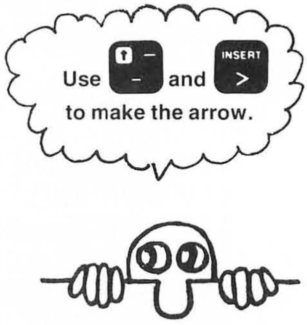
OTHER THINGS TO TRY:

- Add a few choruses of *Happy Birthday to You* to the HAPPY BIRTHDAY, MOTHER program.
- Ask for the screen colors in the BLINK WHAT WHERE program.
- Add a little music to the BLINK WHAT WHERE program.

MAKE THINGS MOVE

You may use the POSITION statement to make things move on the screen. Here is a program to "shoot" an arrow across the screen.

```
100 REM**SHOOT AN ARROW
110 GR. 0
120 SE. 1,0,0
130 SE. 2,0,8
140 POKE 752,1

200 REM**START ARROW AT LEFT
210 POS. 2,12: PR. "- >"
220 TD = 1000
230 GOSUB 910

300 REM**THERE IT GOES!
310 FOR COL=2 TO 37
320   POS. COL,12: PR. "  "
330   POS. COL+1,12: PR. "- >"
340   TD = 20
350   GOSUB 910
360 NEXT COL
```



(continued)

```
400 REM**DO NOTHING LOOP
410 GOTO 410

900 REM**TIME DELAY SUBROUTINE
910 FOR Z=1 TO TD: NEXT Z
920 RETURN
```

Enter and run this program. You will see an arrow (– >) appear in row 12, columns 2 and 3. After a brief pause, it zips across the screen and thunks into the right edge of the screen.

The program will begin with block 100, which will clear the screen, set the screen colors and turn off the cursor.

```
100 REM**SHOOT AN ARROW
110 GR. 0
120 SE. 1,0,0
130 SE. 2,0,8
140 POKE 752,1
```

Block 200 puts the arrow on the screen at column 2, row 12. The tail (–) of the arrow is in column 2 and the head (>) of the arrow is in column 3. We put the arrow at column 2 instead of column 0 because the PRINT statement usually will not print in columns 0 or 1. Lines 220 and 230 cause a long time delay:

```
200 REM**START ARROW AT LEFT
210 POS.2,12:PR. "– >"
220 TD = 1000
230 GOSUB 910
```

Most of the work is done by block 300:

```
300 REM**THERE IT GOES!
310 FOR COL=2 TO 37
320    POS. COL,12: PR. " "
330    POS. COL+1,12: PR. "– >"
340    TD = 20
350    GOSUB 910
360 NEXT COL
```

The *inside* of the FOR-NEXT loop (lines 320, 330, 340, and 350) is done for COL=2, 3, 4, 5, and so on up to 37. Line 320 erases the tail of the arrow at the arrow's current position. Line 330 then prints the arrow one position (COL+1) to the right.

Lines 340 and 350 set the amount of time delay and call the time delay subroutine. Change line 340 to speed up or slow down the arrow. Or, change line 340 to: 340 TD = COL and watch the arrow slow down as it flies across the screen. To slow it down even more, try TD = 2*COL or TD = 3*COL.

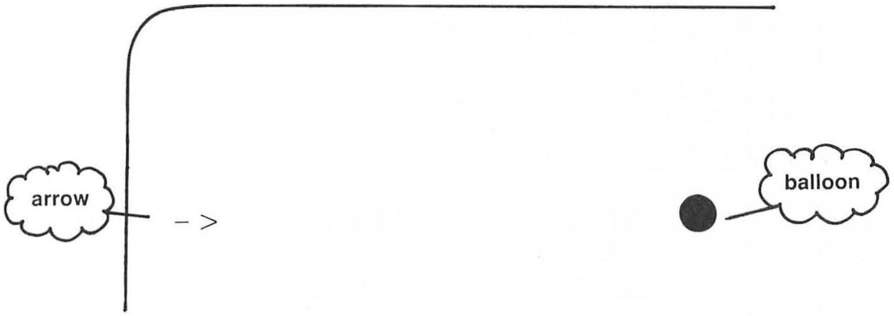
It's more fun with sound. Change line 350 as follows and add line 370:

```
350  SO. 0,COL,10,10:GOSUB 910
370  SO. 0,0,0,0
```

Run this variation and hear the sound as the arrow flies across the screen.

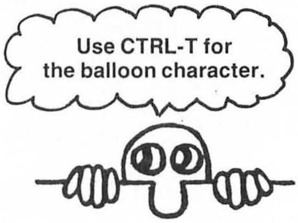
EXPERIMENT! Try several variations of the SOUND statement in line 350. Different distortion? Make the loudness depend on COL? Make the frequency different? Make the arrow go THUNK! as it hits the right side of the screen. EXPERIMENT!

Next, let's put a balloon on the screen as the target for the arrow.



Easy. Change block 200 as follows:

```
200 REM**PUT ARROW & BALLOON ON SCREEN
210 POS. 2,12: PR. "- >"
220 POS. 39,12: PR. "●"
230 TD = 1000
240 GOSUB 910
```



Run the modified program. Watch the arrow fly across the screen and hit the balloon. Pop! The balloon is gone. Try your hand at making a popping sound as the balloon is hit. Begin at line 370. Try a quick FOR-NEXT loop. Experiment with frequency, distortion and loudness.

Questions

1. Modify the SHOOT AN ARROW program so an arrow (< –) flies across the screen from right to left. Start the arrow in column 38, row 12. It should fly to the left and stop with its nose in column 2 and its tail in column

2. Write a program to drop a heart (♥) slowly from the top of the screen to the bottom of the screen. Of course, make it a red heart on a light screen.

3. Fly a ball around the screen from 2,0 to 39,0 to 39,23 to 2,23 to 2,0 — then keep it going around and around.

Answers

1. We included sound in our program:

```
100 REM**SHOOT AN ARROW
110 GR. 0
120 SE. 1,0,0
130 SE. 2,0,8
140 POKE 752,1

200 REM**START ARROW AT RIGHT
210 POS. 38,12: PR. "<-"
220 TD = 1000
230 GOSUB 910

300 REM**THERE IT GOES!
310 FOR COL=38 TO 3 STEP -1
320   POS. COL+1,12: PR. " "
330   POS. COL-1,12: PR. "<-"
340   TD = 20
350   SO. 0,COL,10,10: GOSUB 910
360 NEXT COL
370 SO. 0,0,0,0

400 REM**DO NOTHING LOOP
410 GOTO 410

900 REM**TIME DELAY SUBROUTINE
910 FOR Z=1 TO TD: NEXT Z
920 RETURN
```

Erase tail. Print arrow one position to the left.

Here are some other ways to write lines 310, 320 and 330: There are many more possible!

```
310 FOR COL=37 TO 2 STEP -1
320   POS. COL+2,12: PR. "  "
330   POS. COL,12: PR. "<-"
```

```
310 FOR COL=37 TO 2 STEP -1
320   POS. COL+1,12: PR. "  "
330   POS. COL,12: PR. "<-"
```

2 spaces

```
310 FOR COL=38 TO 3 STEP -1
320   POS. COL,12: PR. "  "
330   POS. COL-1,12: PR. "<-"
```

2 spaces

2.

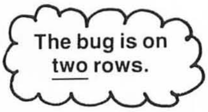
```
100 REM**DROP A HEART
110 GR. 0
120 SE. 1,0,4
130 SE. 2,3,12
140 POKE 752,1

200 REM**START HEART AT TOP
210 POS. 19,0: PR. "♥"
220 TD = 1000
230 GOSUB 920

300 REM**LET IT GO!
310 FOR ROW=0 TO 22
320   POS. 19,ROW: PR. " "
330   POS. 19,ROW+1: PR. "♥";
340   TD = 100
350   GOSUB 910
360 NEXT ROW
370 SO. 0,0,0,0

400 REM**DO NOTHING LOOP
410 GOTO 410

900 REM**SOUND & DELAY SUBROUTINE
910 SO. 0,ROW,10,10
920 FOR Z=1 TO TD: NEXT Z
930 RETURN
```

To prevent scrolling in bottom row.

3. We leave this one for you to do. Also try some of these.

- Run a racing car from the left of the screen to the right. Make the racing car look like this: (@>
- Crawl a bug from the left edge to the right edge of the screen. Make the bug look like the one below.

```
-●●●●●● @
 ))))))
```



- Bounce a ball from left to right, right to left, left to right, and so on.
- Move your name from left to right across the screen so it stops with the last letter of your name touching the right edge of the screen.

NAME MOVER

Karl's name has four letters. George's name has six letters. And there are nine letters in "Aloysious." How many letters in your name? We don't know, but we do know how to find out:

```
100 REM**HOW MANY LETTERS IN A NAME
120 DIM NAME$(25)
120 GR. 0

200 REM**GET A NAME
210 PR. "YOUR NAME";: IN. NAME$

300 REM**COMPUTE NUMBER OF LETTERS
310 NL = LEN(NAME$)

400 REM**PRINT NUMBER OF LETTERS
410 PR. NAME$; " HAS "; NL; " LETTERS"
420 PRINT

500 REM**GO FOR ANOTHER
510 GOTO 210
```

As usual, to find out what it does, ENTER and RUN it.

```
YOUR NAME? TIM
TIM HAS 3 LETTERS

YOUR NAME? BARBARA
BARBARA HAS 7 LETTERS

YOUR NAME? JOHN
JOHN HAS 4 LETTERS

YOUR NAME? FRANCES
FRANCES HAS 7 LETTERS

YOUR NAME? ■
```

and so on. Try it with your name.

Line 310 counts the length (LEN) of the string value of NAME$. The length of a string is the number of characters in the string.

```
310 NL = LEN(NAME$)
```

LENght

of the value of

LEN is a function. You can use it to compute the number of characters in any string. Here are some examples:

```
LEN("A") is 1.
LEN("AB") is 2.
LEN("ABC") is 3.
LEN("12345679") is 8.
```

Did we fool you on that one! Look again and see that the string ″12345679″ has eight characters — there is no 8.

Spaces count. LEN(″ ″) is 1 and LEN(″A B″)is 3.

space **space**

There is a special string called the *empty string* which has no characters — not even one. The empty string consists of two quotation marks with nothing between, not even a space:

LEN(″″) is zero (0).

nothing between quotation marks.

Think about ″bouncing″ a name back and forth across the screen. A one-character name would go from column 0 to column 39 and back. A two-character name would go from column 0 to column 38 and back. A three-character name would go from column 0 to column 37 and back. And so on. In general, a name will go from column 0 to a column that depends on the length of the name.

LENGTH OF NAME	GO TO COLUMN
1	39
2	38
3	37
4	36
N	40 - N

Suppose the name is called NAME$. Then the length of the name is LEN(NAME$). We use this idea in the following program to bounce a name on the screen. Look in lines 310, 510, and 610 for NL, the length of the name.

```
100 REM**NAME BOUNCER
110 DIM NAME$(25)
120 GR. 0

200 REM**GET A NAME
210 POKE 752,0
220 PR. "YOUR NAME";: IN. NAME$
```

```
300 REM**NL IS LENGTH OF NAME
310 NL = LEN(NAME$)

400 REM**PUT NAME NEAR LEFT EDGE
410 PR. CHR$(125)
420 POS.0,12: PR. NAME$
430 POKE 752,1

500 REM**MOVE NAME TO RIGHT EDGE
510 FOR COL=1 TO 40-NL
520   POS. COL-1,12: PR. " "
530   POS. COL,12: PR. NAME$
540   TD = 20
550   GOSUB 910
560 NEXT COL

600 REM**MOVE NAME LEFT
610 FOR COL=40-NL-1 TO 0 STEP -1
620   POS. COL+NL,12: PR. " "
630   POS. COL,12: PR. NAME$
640   TD = 20
650   GOSUB 910
660 NEXT COL

700 REM**GO DO IT AGAIN
710 GOTO 510

900 REM**SOUND & DELAY SUBROUTINE
910 SO. 0,COL,10,10
920 FOR Z=1 TO TD: NEXT Z
930 RETURN
```

Try longer and shorter values of TD in lines 540 and 640.
For example:

Real fast:TD=1
Slow motion:TD=100

Also try
910 SO.0,COL+100,10,10
or
910 SO.0,2*COL+50,10,10

Now try the questions — they will help you understand how the program works.

Questions

1. In the NAME BOUNCER program, suppose you enter CHARLIE as the name.

   (a) What is the value of NL in line 310?____________________.

   (b) In line 510, the value of COL will go from 1 to____________________.

   (c) In line 610, the value of COL will go from ____ to 0.

   (d) Suppose COL is 32. In line 620, where will the space be printed?____

2. VARIATIONS

   (a) Change the screen color to the color of your choice.

   (b) Make the screen color part of the SOUND & DELAY SUBROUTINE. Make it depend on COL.

Answers

1. (a) 7 (b) 33 (c) 32 (d) column39

2. We did (b) like this:

```
900 REM**SOUND, COLOR & DELAY SUBR.
910 SO. 0,COL,10,10
920 SE. 2,COL,8
930 FOR Z=1 TO TD: NEXT Z
940 RETURN
```

   After watching a name bounce for awhile, we added a line to make the letters dark:

```
130 SE. 1,0,0
```

SELF-TEST

What? — another Self-Test? Go jump around a bit. Skip rope, play hopscotch, stretch, relax. Then come and skip through this Self-Test.

1. In GRAPHICS 0, the screen has (a) ____ columns and (b) ____ rows. The columns are numbered from (c) ____ (left of screen) to (d) ____ (right of screen). The rows are numbered from (e) ____ (top of screen) to (f) ____ (bottom of screen).

2.

	0	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	2	-	-	-	-	-	-	-	-	-	3	-	-	-	-	-	-	-	-	-
	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9
0	Z																																							
1																																								
2																																								
3																																								
4																																								
5																																								
6																					1	2	3																	
7																																D								
8																												W	I	Z	A	R	D							
9																																A								
10																																G								
11																															H	O	B	B	I	T				
12																																N								
13												♥																												
14																																								
15																																								
16																																								
17															C																									
18																A																								
19																	T																							
20																																								
21																																								
22																																								
23																																								

(a) What is in column 11, row 13?________

(b) What is in columns 20, 21, and 22, row 6?________

(c) What is in column 31, rows 7 through 12?________

(d) Where is the word WIZARD?________

(e) Where is the word HOBBIT?________

(f) Where is the word RAG?________

(g) Where is the letter Z?________

(h) Where is the word CAT?________

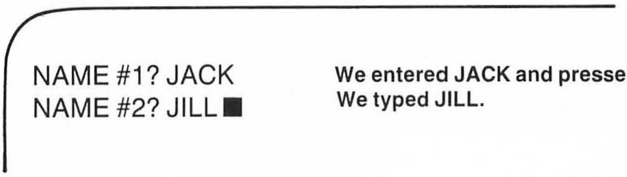
3. How do you tell the computer to:

   (a) Turn off the cursor.________

   (b) Turn on the cursor.________

4. Write a program to put the message MERRY CHRISTMAS, FATHER on line 10. Center the message on the line. Do not show the cursor.

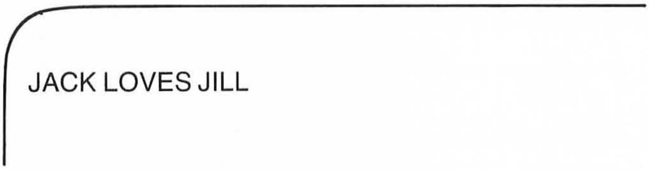
5. Write a program to ask for two names. For example, it might start like this:



```
NAME #1? JACK
NAME #2? JILL■
```

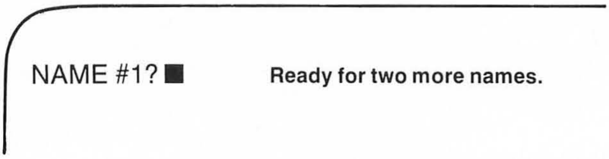
We entered JACK and pressed RETURN.
We typed JILL.

Now we press RETURN and see only this near the middle of the screen:



```
JACK LOVES JILL
```

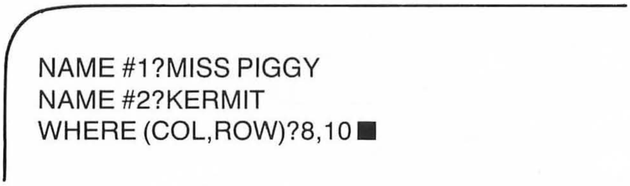
After a few seconds, we again see:



```
NAME #1?■
```

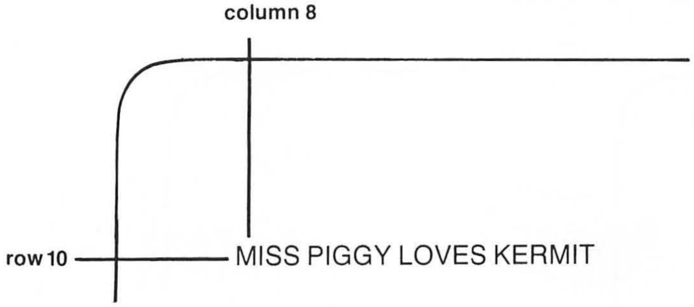
Ready for two more names.

6. Modify your program in question 5 to instruct the computer to blink the JACK LOVES JILL message (or use your two names) wherever you want.



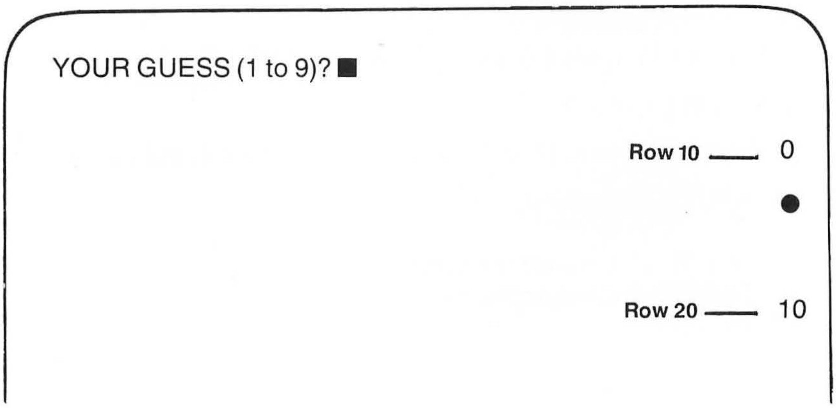
```
NAME #1?MISS PIGGY
NAME #2?KERMIT
WHERE (COL,ROW)?8,10■
```

Now press RETURN and see:

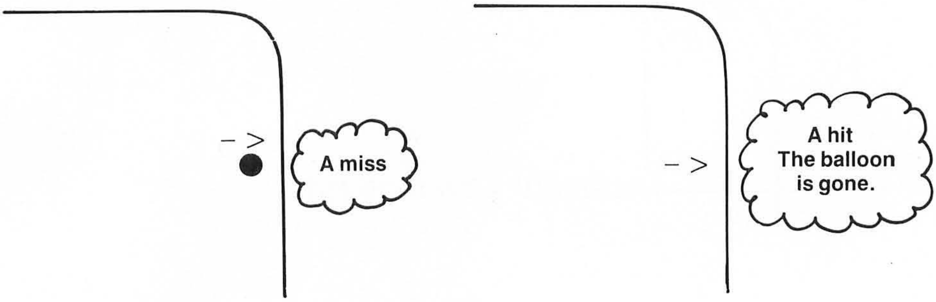


7. Write a program to bounce a ball back and forth on the screen. The ball should go from column 0, row 12 to column 39, row 12, then back to column 0, row 12, then repeat until someone presses BREAK.

8. Write a program so two people can play a simple game. Player A secretly enters a number from 1 to 9 to place a balloon on the right edge of the screen between the numbers 0 and 10. The number 0 is in row 10 and the number 10 is in row 20. The ball is in row 10+N, where N is the number secretly entered by player A. Suppose player A enters the number 3. Player B will now see:



Player B now enters a number from 1 to 9. This is the value of N. An arrow appears at row 10+N and flies across the screen. It either hits or misses the balloon. For example, here are a miss and a hit:



Answers to Self-Test

1. (a) 40 (b) 24 (c) 0 (d) 39 (e) 0 (f) 23

2. (a) a heart (♥)

   (b) The number 123

   (c) The word DRAGON

   (d) Columns 27 through 32, row 8

   (e) Columns 30 through 35, row 11

   (f) Column 31, rows 8,9, and 10. It is part of DRAGON.

   (g) Column 0, row 0

   (h) Column 14, row 17; column 15, row 18; and column 16, row 19

3. Use a POKE instruction:

```
POKE 752,1 turns off the cursor
POKE 752,0 turns on the cursor
```

4. We put our message with dark letters on a green screen with a pink border:

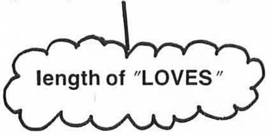
```
100 REM**MERRY CHRISTMAS, FATHER
110 GR. 0
120 POKE 752,1

200 REM**SCREEN COLORS
210 SE. 1,0,0
220 SE. 2,12,8
230 SE. 4,4,8

300 REM**PUT MESSAGE AT 8,10
310 POS. 8,10
320 PR. "MERRY CHRISTMAS, FATHER"

400 REM**DO NOTHING LOOP
410 GOTO 410
```

5. This program will put the message at column 0, row 12:

```
100 REM**SOMEONE LOVES SOMEONE
110 GR. 0
120 DIM N1$(20), N2$(20)
130 SE. 1,0,0

200 REM**GET TWO NAMES
210 POKE 752,0
220 PR. CHR$(125)
230 PR. "NAME #1";: IN. N1$
240 PR. "NAME #2";: IN. N2$

300 REM**PRINT MESSAGE
310 POKE 752,1
320 PR. CHR$(125)
330 POS. 0,12
340 PR. N1$; " LOVES ";N2$

400 REM**LONG TIME DELAY
410 TD = 3000
420 FOR Z=1 TO TD: NEXT Z

500 REM**GO FOR MORE NAMES
510 GOTO 210
```

How long is the message? Let LM be the length of the message.

LM = LEN(N1$) + 7 + LEN(N2$)



We can center the message by printing it in column LM/2. Modify block 300 as follows:

```
300 REM**PRINT MESSAGE
310 POKE 752,1
320 PR. CHR$(125)
330 LM = LEN(N1$) + 7 + LEN(N2$)
335 POS. LM/2,12
340 PR. N1$; " LOVES "; N2$
```

6. We made these changes to our original program:

```
250 PR. "WHERE (COL,ROW)";: IN. COL,ROW

330 POS. COL,ROW
```

7.
```
100 REM**BOUNCING BALL
110 GR.0
120 DIM OBJECT$(1)
130 OBJECT$ = "●"

200 REM**START OBJECT AT LEFT
210 POKE 752, 1
220 POS. 0,12: PR. OBJECT$
230 TD = 1000
240 GOSUB 910

300 REM**MOVE IT TO RIGHT
310 FOR COL=0 TO 38
320   POS. COL,12: PR. " "
330   POS. COL+1,12: PR. OBJECT$
340   TD = 20
350   GOSUB 910
360 NEXT COL
```

(continued)

```
400 REM**MOVE IT TO LEFT
410 FOR COL=38 TO 0 STEP -1
420   POS. COL+1, 12: PR. " "
430   POS. COL,12: PR. OBJECT$
440   TD = 20
450   GOSUB 910
460 NEXT COL

500 REM**KEEP IT BOUNCING
510 GOTO 310

900 REM**TIME DELAY SUBROUTINE
910 FOR Z=1 TO TD: NEXT Z
920 RETURN
```

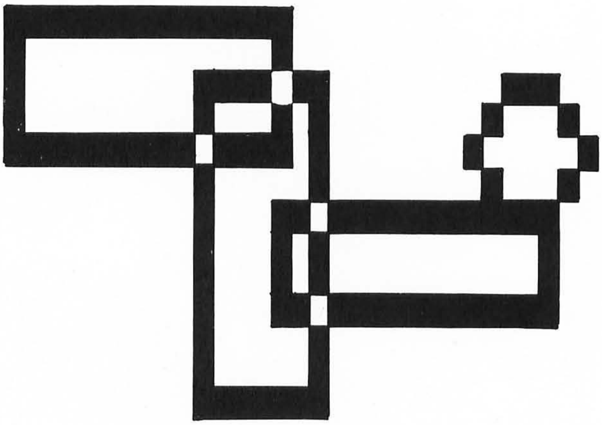
8. As usual, we leave at least one for you to do without our solution.

Chapter Seven

Graphics Galore

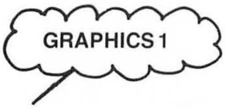
In this chapter you will begin to learn how to put patterns and pictures on the screen. You will learn about GRAPHICS 1, GRAPHICS 2 and beyond. When you finish this chapter, you will be able to:

- Use GRAPHICS 1 and GRAPHICS 2 to put big letters on the screen
- Use GRAPHICS 3 through 8 to PLOT tiny colored rectangles on the screen
- Use DRAWTO to draw lines on the screen
- Use the COLOR command to select colors of plotted points and lines

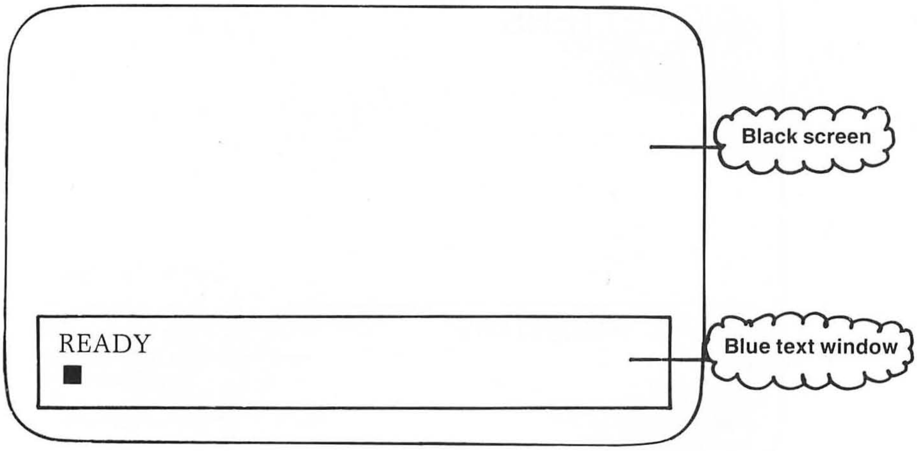


BIG LETTERS

You have been exploring the land of GRAPHICS 0. Now get ready to take a trip to the realm of GRAPHICS 1.



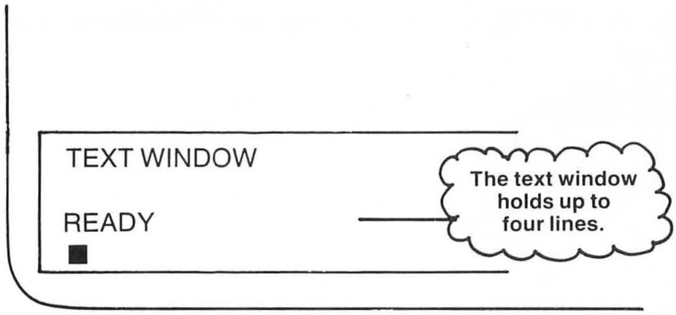
Type GR. 1 and press the RETURN key. This is what you see:



```
READY
■
```

In GRAPHICS 1, the screen is split into two parts. We call the top part the *graphics screen* and the bottom part the *text window*. To print information in the text window, use PRINT as before. Use a slightly different PRINT instruction to print large letters in the top part of the screen. First, print something in the text window.

Type PR."TEXT WINDOW" and press RETURN.



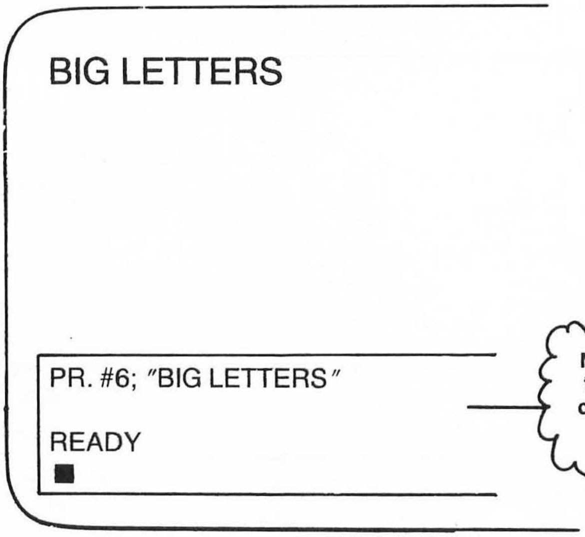
```
TEXT WINDOW

READY
■
```

Next, print some big letters in the top part of the screen.

Type PR #6; "BIG LETTERS" and press RETURN.

semi-colon

Here are your big letters:



```
BIG LETTERS





PR. #6; "BIG LETTERS"

READY
■
```

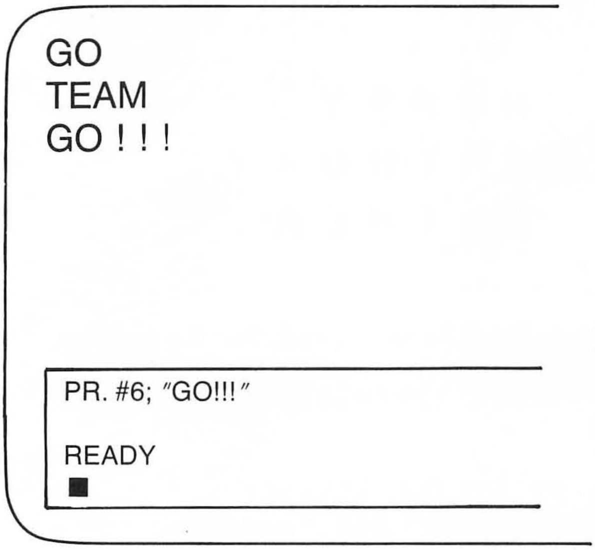
Note that the text window can hold only four lines.

REMEMBER: In GRAPHICS 1 use PR. # 6; to print big letters in the top part of the screen. Try another one.

Type GR. 1 and press RETURN to clear the top part of the screen. Now type the following three lines:

```
PR. #6; "GO"
PR. #6; "TEAM"
PR. #6; "GO!!!"
```

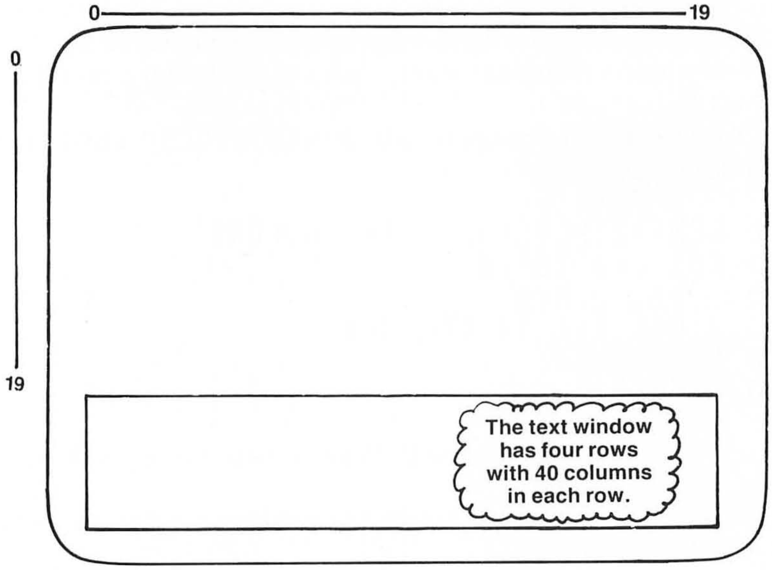
When we did it, the screen looked like this.



```
GO
TEAM
GO ! ! !
```

```
PR. #6; "GO!!!"

READY
■
```

In GRAPHICS 1, the screen is split into two parts. The top part is called the graphics screen; the bottom part is called the text window. The graphics screen has 20 columns and 20 rows. Columns are numbered 0 to 19 and rows are numbered 0 to 19. The text window has four rows with 40 columns in each row.



You can write a program in GRAPHICS O mode and have the program tell the computer to use GRAPHICS 1. Here is a program to put the following message on the screen:

```
  H A P P Y
B I R T H D A Y
 M O T H E R
```

Now go to GR. 0 (press SYSTEM RESET) and enter this program:

```
100 REM**HAPPY BIRTHDAY, MOTHER
110 GR.1

220 REM**PRINT THE MESSAGE
210 POS. 7,8: PR. #6; "HAPPY"
220 POS. 6,10: PR. #6; "BIRTHDAY"
230 POS. 7,12: PR.#6; "MOTHER"

300 REM**DO NOTHING LOOP
310 GOTO 310
```

Enter and run the program. Line 110 puts the computer in GRAPHICS 1 mode, then lines 210, 220, and 230 print the message near the center of the top part of the screen. The text window is blue and empty. Remember, the top part of the screen has 20 columns and 20 rows.

Press SYSTEM RESET. The computer is back in GRAPHICS 0. List the program and make these changes:

```
300 REM**ZIP THROUGH THE COLORS
310 FOR C=0 TO 15
320   SE. 2,C,8
330   FOR Z=1 TO 50: NEXT Z
340 NEXT C
350 GOTO 310
```

Line 320 sets color register 2 to color C. Color register 2 controls the color and luminance of the text window.

Run this variation and mother will see the text window changing color while she enjoys her birthday message.

> REMEMBER: In GRAPHICS 1, color register 2 controls the color and luminance of the text window.

Try another variation. Go to GRAPHICS 0 and make these changes:

```
300 REM**TEXT WINDOW MESSAGE
310 PR. CHR$(125)
320 PR. "HAPPY BIRTHDAY, MOTHER  ";
330 FOR Z=1 TO 200: NEXT Z
340 GOTO 320
```

What do you think will happen when you run this variation? Try it and find out. Then try this program to blink someone's name 10 times.

```
100 REM**BLINK NAME IN GR. 1
110 DIM NAME$(20)
120 GR. 1

200 REM**GET NAME
210 PR. CHR$(125)
220 PR. "YOUR NAME";: IN. NAME$

300 REM**BLINK 10 TIMES
310 FOR BLINK=1 TO 10
320   POS. 0,0: PR. #6; NAME$
330   GOSUB 910
340   GR. 1
350   GOSUB 910
360 NEXT BLINK

400 REM**GO FOR ANOTHER NAME
410 GOTO 210

900 REM**TIME DELAY SUBROUTINE
910 TD = 200
920 FOR Z=1 TO TD: NEXT Z
930 RETURN
```

Clears the entire screen, both top and bottom.

Questions

1. In GRAPHICS 1, the screen is divided into two parts. In the top part, there are (a) ___ columns and (b) ___ rows. In the bottom part, called the text window, there are (c) ___ columns and (d) ___ rows.

2. Change the BLINK NAME IN GR. 1 program so the name is blinked near the center of row 9.

3. Write a program to roll a "wheel" (O) across any row from column 0 to column 19. Your program should ask for the row. Use the letter oh (O) as the wheel.

Answers

1. (a) 20 (b) 20 (c) 40 (d) 4

   In the top part, letters and other characters, including graphics characters, are twice as wide as characters in the text window.

2. Remember, the top part of the screen is 20 characters wide. We changed block 300 this way:

```
300 REM**BLINK 10 TIMES
310 COL = (20 - LEN(NAME$))/2
320 FOR BLINK=1 TO 10
350   POS. COL,9: PR. #6; NAME$
330   GOSUB 910
340   GR. 1
350   GOSUB 910
360 NEXT BLINK
```

   Line 310 computes the column in which to print the name so that it is approximately centered. For example, the length of JASON is 5, so COL = (20 -5)/2 = 15/2 = 7.5 and JASON will be printed in column 7 (the part to the right of the point is ignored).

3.
```
100 REM**ROLL A BIG WHEEL
110 GR. 1

200 REM**GET THE ROW
210 PR. CHR$(125)
220 PR. "WHAT ROW (0 TO 19)";: IN. ROW

300 REM**START THE WHEEL AT 0,ROW
310 POS. 0,ROW: PR. #6; "O"
320 TD = 1000
330 GOSUB 910

400 REM**ROLL THE WHEEL TO 19,ROW
410 FOR COL=1 TO 19
420   POS. COL,ROW: PR. #6; "O"
430   POS. COL-1,ROW: PR. #6; " "
440   TD = 50
450   GOSUB 910
460 NEXT COL

500 REM**GO FOR ANOTHER ROLL
510 GOTO 210

900 REM**TIME DELAY SUBROUTINE
910 FOR Z=1 TO TD: NEXT Z
920 RETURN
```

Use this program to roll a bunch of wheels. Note that the wheels are orange. Soon we will show you how to change the color of big things in GR. 1.

EXPERIMENT! In GRAPHICS 2, the top part of the screen has 20 columns and 10 rows. Columns are numbered 0 to 19 and rows 0 to 9. Letters and other characters are twice as wide and twice as tall as in GRAPHICS O. For example, here is the letter A in all three modes:

GR. 0

GR. 1

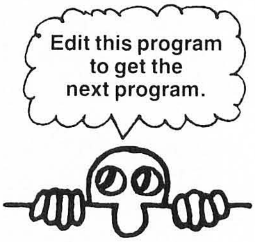
GR. 2

Experiment with GRAPHICS 2. Try all the things you did with GRAPHICS 1. Remember, there are only 10 rows in GR. 2, numbered 0 to 9.

In both GRAPHICS 1 and GRAPHICS 2, color register 2 controls the color and luminance of the text window.

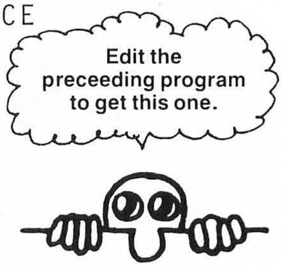
Color register 0 controls the color and luminance of upper case characters (CAPS). Try this program in both GR. 1 and GR. 2. In line 110, try both GR. 1 and GR. 2:

```
100 REM**CHAMELEON IN GR. 1 & 2
110 _______________

200 REM**PUT CHAMELEON ON SCREEN
210 PR. #6; "CHAMELEON"

300 REM**CHANGE COLOR & LUMINANCE
310 FOR C=0 TO 15
320   FOR L=0 TO 15
330     SE. 0,C,L
340     FOR Z=1 TO 100: NEXT Z
350   NEXT L
360 NEXT C

400 REM**GO DO IT AGAIN
410 GOTO 310
```



Next, experiment with color register 4. In GR. 1 and GR. 2, color register 4 controls background color and luminance of the top part of the screen. Try this program in both GR. 1 and GR. 2 (line 110).

```
100 REM**BACKGROUND IN GR. 1 & 2
110 _______________

200 REM**PUT THIS ON SCREEN
210 PR. #6; "BACKGROUND COLORS"

300 REM**CHANGE COLOR & LUMINANCE
310 FOR C=0 TO 15
320   FOR L=0 TO 14 STEP 2
330     SE. 4,C,L
340     FOR Z=1 TO 200: NEXT Z
350   NEXT L
360 NEXT C
```



Color register 2 controls the background color in the text window. It also controls the color of *inverse video* characters in the top part of the screen. Try some inverse video characters in GRAPHICS O.

- Press SYSTEM RESET
- Press the inverse video key. It looks like this:

  Models 400 and 800: Λ

  Models 600XL and 800XL: ◩
- Type some letters. For example, type ABC. You will see them in inverse video.

  ABC
- Press Λ or ◩ to get back to normal video.

In GRAPHICS 0, inverse video characters are darker blue on a lighter blue rectangle. That is, in inverse video the luminances of the foreground and background are reversed.

> REMEMBER: Press Λ (ATARI 400 or 800) or ◩ (ATARI 600XL or 800XL) to get into or out of inverse video.

Now try inverse video in GRAPHICS 2.

- Press SYSTEM RESET.
- Type GR. 2 and press RETURN.
- Type PR. #6; " INVERSE VIDEO " and press RETURN.

  Press Λ or ◩ here. Press Λ or ◩ here.

In the top part of screen, you will see INVERSE VIDEO in dark blue letters.

- Type SE. 2,12,8 and press RETURN.

Presto chango! The big letters (and the text window) become green. Change them to pink.

- Type **SE. 2,4,8** and press RETURN.

The big letters become pink.
Color register 1 controls the color of lower case characters.

- Press SYSTEM RESET.
- Type **GR. 2** and press RETURN.
- Type **PR. #6; "lower case"**

Press CAPS LOWR here

Hold down SHIFT and press CAPS LOWR here.

You will see LOWER CASE in light green letters in the top part of the screen. Why not in lower case letters? Well, that's just the way it is. You can't print lower case letters up there.

- Use SETCOLOR with register 1 to change the color of LOWER CASE.
- Then try lower case inverse video characters. Their color is controlled by color register 3. At first, you will see red.

REMEMBER. In GRAPHICS 1 and GRAPHICS 2, the colors of information in the graphics screen (top part of the split screen) are controlled by color registers 0, 1, 2, 3, and 4, as shown in the following table.

COLOR REGISTER	CONTROLS COLOR & LUMINANCE	NORMAL COLOR*
0	upper case	orange
1	lower case	light green
2	inverse video upper case	dark blue
3	inverse video lower case	red
4	background	black

*These are the colors right after you type GR. 1 or GR. 2. Use SETCOLOR to change any of these.

EXPERIMENT! Use this program:

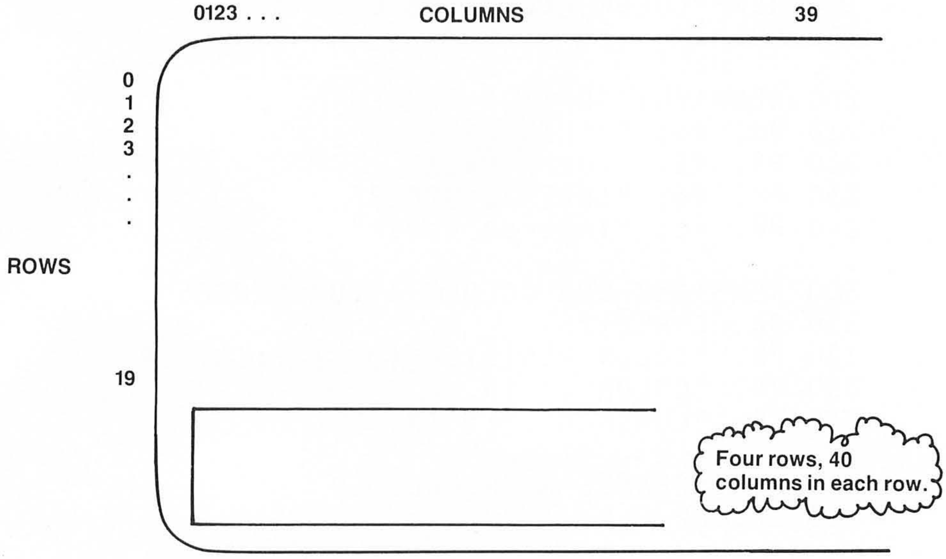
```
100 REM**COLOR REGISTERS 0 TO 3
110 GR. 2

200 REM**PUT INFO ON SCREEN
210 PR. #6; "UPPER CASE"
220 PR. #6; "lower case"
230 PR. #6; "INVERSE UPPER"
240 PR. #6; "inverse lower"

300 REM**ASK FOR COLORS & LUMINANCES
310 PR. CHR$(125)
320 PR. "COLOR REGISTER (0-4)";:IN. CR
330 PR. "COLOR";: IN. C
340 PR. "LUMINANCE";: IN. L

400 REM**CHANGE AS REQUESTED
410 SE. CR,C,L

500 REM**GO ASK AGAIN
510 GOTO 310
```

TINY RECTANGLES

Patterns. That's what you will do next — you will put patterns on the screen, using little rectangles of colored light. To do this, you will use the GRAPHICS 3 mode and two new instructions called COLOR and PLOT.

Type **GRAPHICS 3** and press RETURN.
or type **GR. 3** and press RETURN.

Welcome to GRAPHICS 3. You see a two-part screen as in GRAPHICS 1 or GRAPHICS 2. The top part of the screen has 40 columns (0 to 39) and 20 rows (0 to 19). The text window is the same as before.



You can put a tiny rectangle of colored light anywhere in the top part of the screen — but first you must select a color.

- Type COLOR 1 and press RETURN. You have selected the color orange.
- Type PLOT 20,10 and press RETURN. You will see an orange rectangle appear in column 20, row 10. This is near the center of the screen.

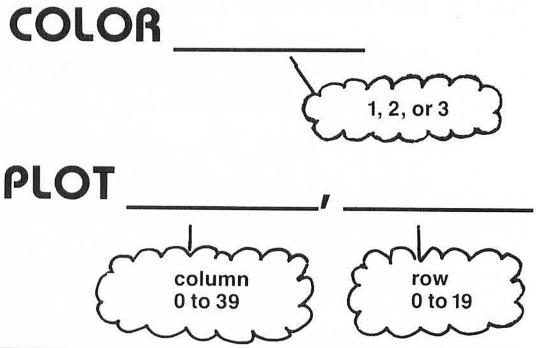
Try two more. Put a light green rectangle in the upper left corner and a dark blue rectangle in the upper right corner.

COLOR 2 —— light green

PLOT 0,0

COLOR 3 —— dark blue

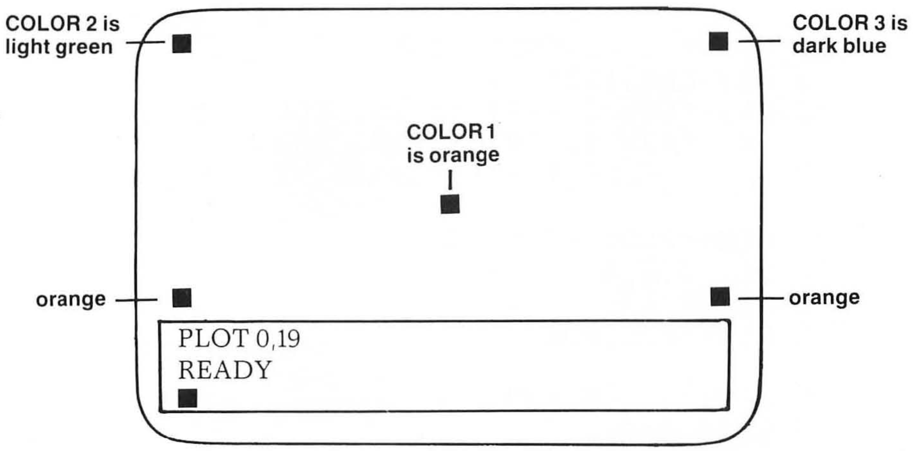
PLOT 39,0



Then put orange rectangles in the two lower corners.

```
COLOR 1

PLOT 39,19

PLOT 0,19
```

You have now put five colored rectangles on the screen.



```
PLOT 0,19
READY
■
```

GRAPHICS 3 colors are controlled by color register 0, 1, 2, and 4. Color register 3 is not used. When you type GRAPHICS 3 (or GR. 3), the computer loads the color register this way:

COLOR REGISTER	COLOR	DESCRIPTION
0	orange	This is COLOR 1
1	light green	This is COLOR 2
2	dark blue	This is COLOR 3
3	-	Not used in GR. 3
4	black	Background color

Please note that COLOR 1 picks the color in color register 0, COLOR 2 picks the color in register 1, and COLOR 3 picks the color in register 2. Strange, but true.

Of course, you may use SETCOLOR to change the colors in any of the color registers. If you change the color in color register 0, all points already plotted in COLOR 1 change to the new color. Try it. Plot a bunch of points using any of the three color registers, then change the color in the color register. You will see all your points change to the new color. Or try the next program, which uses SETCOLOR 0 and COLOR 1. Watch all the points change whenever you enter a color that is different from your previous color.

```
100 REM**PLOT ANY COLOR
110 GR. 3

200 REM**GET INFORMATION
210 PR. CHR$(125)
220 PR. "COL (0-39)";: IN. COL
230 PR. "ROW (0-19)";: IN. ROW
240 PR. "COLOR (0-15)";: IN. C

300 REM**PLOT AS REQUESTED
310 SE. 0,C,8
320 COLOR 1
330 PLOT COL,ROW

400 REM**GO DO ANOTHER
410 GOTO 210
```

KEEP EXPERIMENTING. The next program will plot three points, then let you change the color and luminance of any point:

```
100 REM**COLOR EXPERIMENTS IN GR. 3
110 GR. 3

200 REM**PLOT 3 POINTS IN 3 COLORS
210 COLOR 1: PLOT 5,5
220 COLOR 2: PLOT 10,10
230 COLOR 3: PLOT 15,15

300 REM**ASK FOR A COLOR
310 PR. CHR$(125)
320 PR. "COLOR NO. (1-3)";: IN. CN
330 PR. "COLOR (0-15)";: IN. C
340 PR. "LUMINANCE (0-14)";: IN. L
```

```
400 REM**SET COLOR AS REQUESTED
410 SE. CN-1,C,L

500 REM**GO FOR ANOTHER
510 GOTO 310
```

REMEMBER: how COLOR, SETCOLOR, and color registers go together in GRAPHICS 3.

COLOR REGISTER	TO CHANGE COLOR	TO SELECT REGISTER
0	SETCOLOR 0	COLOR 1
1	SETCOLOR 1	COLOR 2
2	SETCOLOR 2	COLOR 3
3	Not used in GR. 3	Not used in GR. 3
4	SETCOLOR 4	Background color

Also remember that color register 2 controls the color and luminance of the text window.

Questions

1. In GRAPHICS 3, COLOR 1 is the color stored in color register 0. It begins as the color orange. How can you change COLOR 1 to medium pink?________

2. Color 2 is in color register (a) ____. It begins as the color (b) ____. How do you change it to dark green? (c) ____.

3. How do you change COLOR 3 to light purple?________________________

4. How do you change the screen background color to white (lightest gray)?___

5. Write a program to put a tiny rectangle at column 19, row 9. Then make it go through all luminances in all colors.

Answers

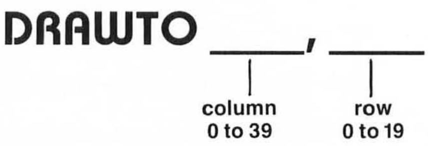
1. SETCOLOR 0,4,8
2. (a) 1 (b) light green (c) SETCOLOR 1,12,0
3. SETCOLOR 2,5,14
4. SETCOLOR 4,0,14
5. We used COLOR 3 (color register 2).

```
100 REM**ALL COLORS IN GR. 3
110 GR. 3

200 REM**PLOT ONE POINT
210 COLOR 3: PLOT 19,9

300 REM**ALL COLORS IN ALL LUMINANCES
310 FOR C=0 TO 15
320   FOR L=0 TO 14 STEP 2
330     SE. 2,C,L
340   NEXT L
350 NEXT C

400 REM**DO IT AGAIN
410 GOTO 310
```

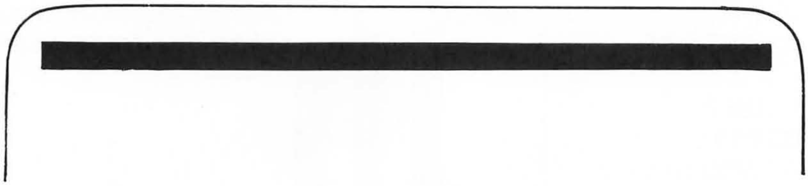
DRAW A LINE

You may plot a tiny rectangle anywhere on the screen. You can also draw a line from that place to any other place on the screen. Draw an orange line across the top of the screen this way:

```
GR. 3
COLOR 1
PLOT 0,0
DRAWTO 39,0
```



If all goes well, you will see an orange line across the screen from column 0, row 0 to column 39, row 0.



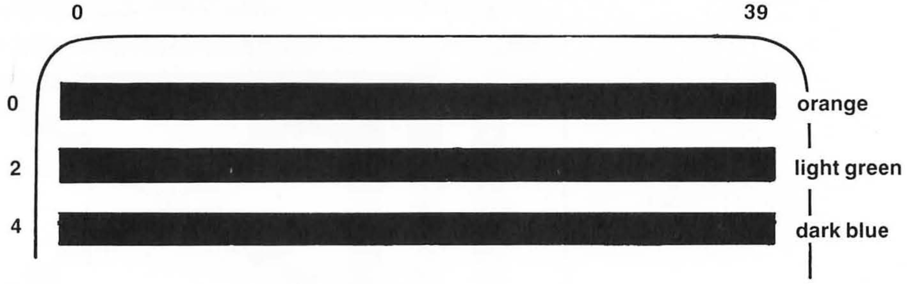
Keep drawing.

```
COLOR 2
PLOT 0,2
DRAWTO 39,2
```

a light green line.

```
COLOR 3
PLOT 0,4
DRAWTO 39,4
```

A dark blue line.

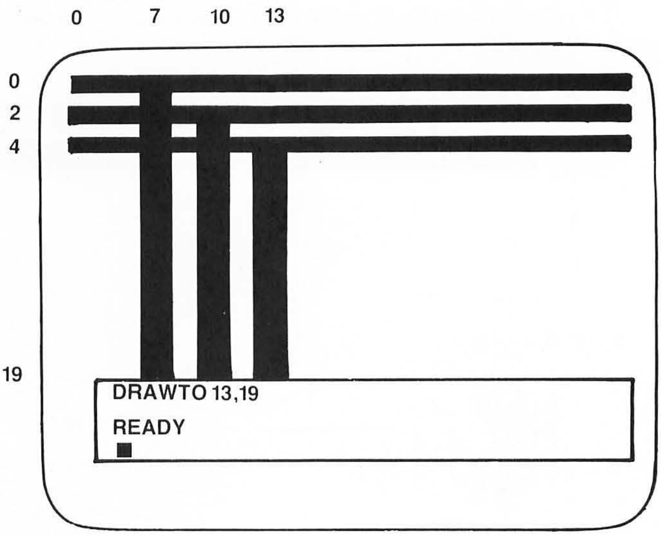


Yes, you can use SETCOLOR to change the color of any of these lines.

- Change the top line to medium green: SETCOLOR 0,12,8
- Change the middle line to darkish pink: SETCOLOR 1,4,4
- Change the bottom line to medium purple: SETCOLOR 2,5,8
- Change the background to lightest gray: SETCOLOR 4,0,14

Add some vertical lines.

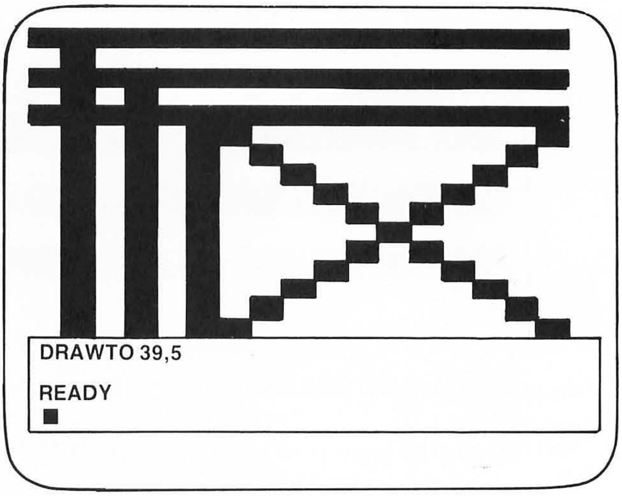
```
COLOR 1
PLOT 7,0
DRAWTO 7,19

COLOR 2
PLOT 10,2
DRAWTO 10,19

COLOR 3
PLOT 13,4
DRAWTO 13,19
```



```
DRAWTO 13,19
READY
■
```

Horizontal and vertical lines look smooth. Other lines look a little jagged. Add some diagonal lines.

```
COLOR 1
PLOT 14,5
DRAWTO 39,19

COLOR 2
PLOT 14,19
DRAWTO 39,5
```



```
DRAWTO 39,5
READY
■
```

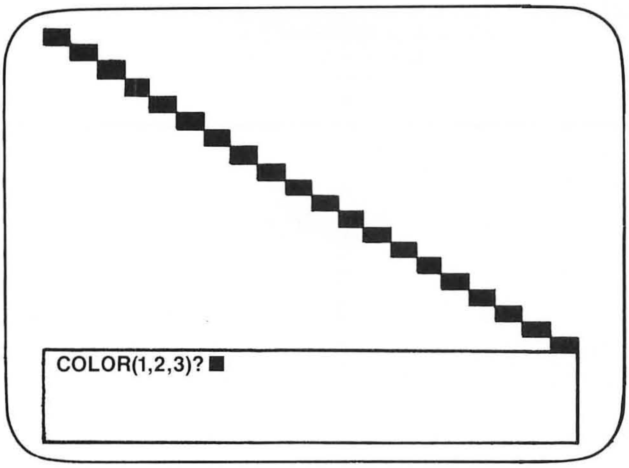
Now use this program to draw patterns of lines in orange, light green, and dark blue. For each line select COLOR 1, COLOR 2 or COLOR 3, the place to PLOT the beginning of the line and the place to DRAWTO.

```
100 REM**DRAW LINES IN GR. 3
110 GR. 3

300 REM**GET INFO ABOUT LINE
310 PR. CHR$(125)
320 PR. "COLOR (1,2,3)";:IN. C
330 PR. "PLOT (COL,ROW)";: IN. C1,R1
340 PR. "DRAWTO (COL,ROW)";: IN. C2,R2

400 REM**DRAW THE LINE
410 COLOR C
420 PLOT C1,R1
430 DRAWTO C2,R2

500 REM**GO FOR MORE INFORMATION
510 GOTO 310
```

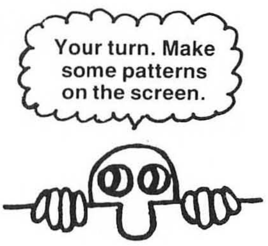
Enter and run the program. Enter the information for each line as asked for in the text window. For example, here is the information to draw an orange line from the top left corner to the bottom right corner of the screen:

```
COLOR (1,2,3)?1
PLOT (COL,ROW)?0,0
DRAWTO (COL,ROW)?39,19 ■
```

Press RETURN and see it happen.



```
COLOR(1,2,3)? ■
```



Questions

1. Write the commands to tell the computer to draw a light green line from the bottom left corner to the top right corner of the screen.

   ____

   ____

   ____

   ____

2. Write the commands to tell the computer to draw a medium purple line from 6,6 to 24,6.

   ____

   ____

   ____

   ____

3. Add block 200 to the DRAW LINES IN GR. 3 program so the computer will first ask these questions:

```
COLOR 1 (COLOR,LUM)?
COLOR 2 (COLOR,LUM)?
COLOR 3 (COLOR,LUM)?
BACKGROUND (COLOR,LUM)?
```

   This will let the user select the colors and luminances for COLOR 1, COLOR 2, COLOR 3, and the background. The rest of the program is the same.

Answers

1.

```
GR.3
COLOR 2
PLOT 0,19
DRAWTO 39,19
```

2. We used COLOR 3, controlled by color register 2.

```
GR.3
SE.2,5,8
COLOR 3
PLOT 6,6
DRAWTO 24,6
```

3.

```
200 REM**GET COLORS & LUMINANCES
210 PR. CHR$(125)
220 PR. "COLOR 1 (COLOR,LUM)";: IN. C1,L1
230 PR. "COLOR 2 (COLOR,LUM)";: IN. C2,L2
240 PR. "COLOR 3 (COLOR,LUM)";: IN. C3,L3
250 PR. "BACKGROUND (COLOR,LUM)";: IN. CB,LB
260 SE. 0,C1,L1
270 SE. 1,C2,L2
280 SE. 2,C3,L3
290 SE. 4,CB,LB
```

USE READ & DATA TO PAINT THE SCREEN

Here is a program to put a familiar pattern on the screen:

```
100 REM**SOMETHING IN THE SKY
110 GR. 3

200 REM**READ STAR POSITION
210 READ COL,ROW

300 REM**PUT A STAR IN THE SKY
310 COLOR 1
320 PLOT COL,ROW

400 REM**GO FOR ANOTHER ONE
410 GOTO 210

1000 REM**STAR POSITIONS
1010 DATA 9,9, 13,9, 16,10, 21,12
1020 DATA 22,15, 28,15, 29,11
```

Run the program. In the top part of the screen you see the Big Dipper with seven orange stars. In the text window you see: ERROR-6 AT LINE 210. This is an out-of-data message – just ignore it and watch the Big Dipper.

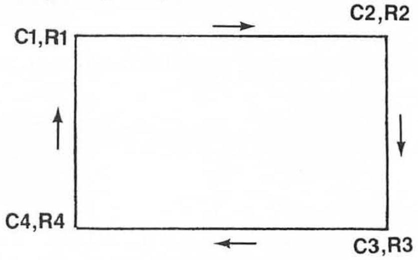
The star positions are in the DATA statements. Each star position is a pair of numbers for COL and ROW.

```
1010 DATA 9,9, 13,9, 16,10, 21,12
       /       \
   COL,ROW   COL,ROW      and so on.
```

Go ahead. Replace our constellation with your own. Put your star position in DATA statements beginning at line 1010. Look in Appendix D for a blank screen map to help you plot your star positions.

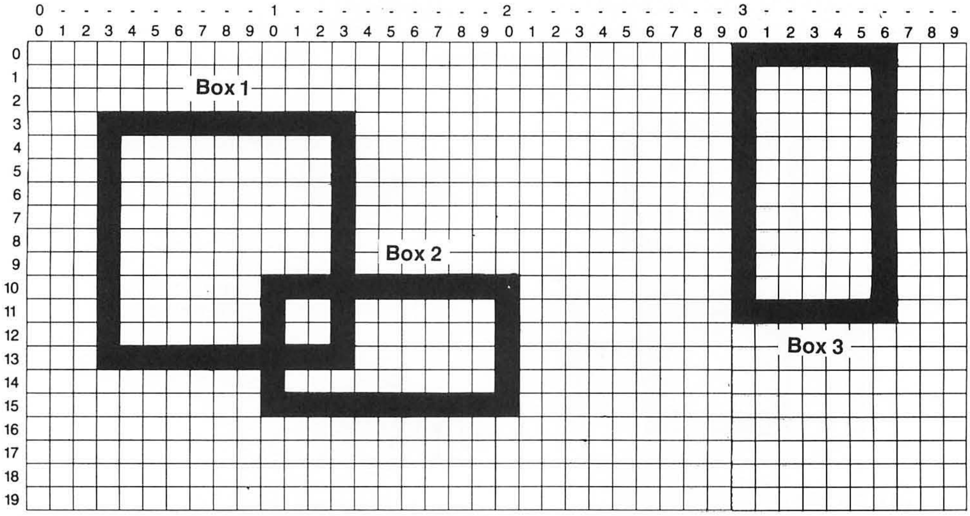
Next, a program to draw boxes on the screen. Each box is defined by a PLOT followed by four DRAWTO's. You will even be able to include the color (1, 2, or 3) in DATA statements:

```
100 REM**BOXES IN GR. 3
110 GR. 3

200 REM**READ BOX DESCRIPTION
210 READ CN,C1,R1,C2,R2,C3,R3,C4,R4

300 REM**DRAW THE BOX
310 COLOR CN
320 PLOT C1,R1
330 DRAWTO C2,R2
340 DRAWTO C3,R3
350 DRAWTO C4,R4
360 DRAWTO C1,R1
```



```
400 REM**GO DRAW ANOTHER
410 GOTO 210

1000 REM**BOX DESCRIPTIONS
1010 DATA 1,3,3,13,3,13,13,3,13       Box 1
1020 DATA 2,10,10,20,10,20,15,10,15   Box 2
1030 DATA 3,30,0,36,0,36,11,30,11     Box 3
```

We show data for three boxes. They will look this way on the screen:



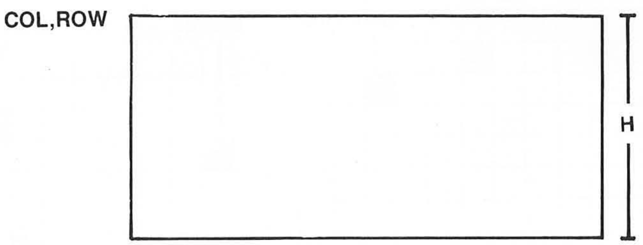
Ignore the out-of-data error message in the text window.

Questions

1. Change the SOMETHING IN THE SKY program to instruct the computer to read the color number (1, 2, or 3) along with the star position for each star. Put color numbers of your choice in the DATA statements.

2. Write a program to put lines on the screen. Read the color (1, 2, or 3), starting position and end position for each line from DATA statements.

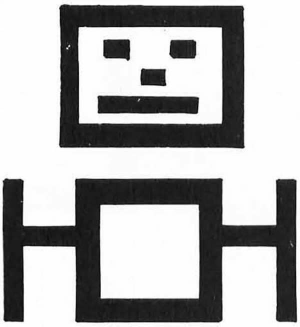
3. Change our BOXES IN GR. 3 program so each box is defined by the screen position of its color (C), upper left corner (COL,ROW), and its width (W) and height (H):



A READ statement might look like this way:

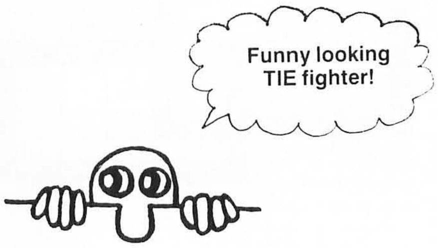
```
READ CN,COL,ROW,W,H
```

Use your program to make these patterns:



The eyes and nose are single points.

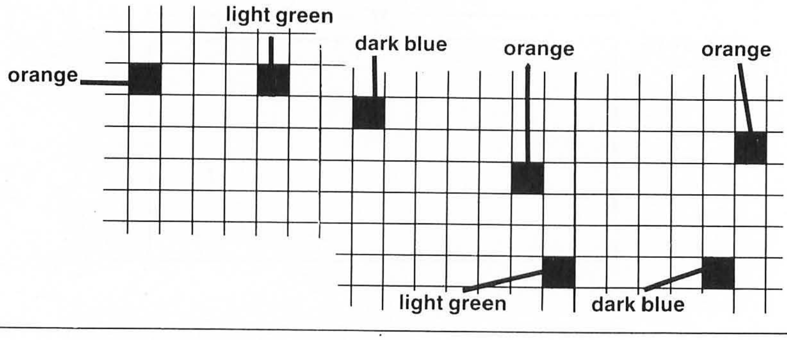
The mouth is a line.



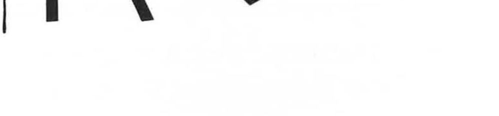
Answers

1. Change lines 210, 310, and the DATA statements. We did it this way:

```
210 READ COL,ROW,C

310 COLOR C

1000 REM**STAR POSITIONS & COLORS
1010 DATA 9,9,1
1020 DATA 13,9,2
1030 DATA 16,10,3
1040 DATA 21,12,1
1050 DATA 22,15,2
1060 DATA 28,15,3
1070 DATA 29,11,1
```



2.
```
100 REM**PAINT LINES IN GR. 3
110 GR. 3

200 REM**GET DESCRIPTION
210 READ CN,C1,R1,C2,R2

300 REM**PAINT A LINE
310 COLOR CN
320 PLOT C1,R1
330 DRAWTO C2,R2

400 REM**GO FOR ANOTHER LINE
410 GOTO 210

1000 REM** CN,C1,R1,C2,R2
1010 DATA 1,0,0,39,0
1020 DATA 2,0,1,39,10
1030 DATA 3,0,2,39,17
1040 DATA 1,0,3,26,19
1050 DATA 2,0,4,13,19
1060 DATA 3,0,5,0,19
```



3. Change blocks 200, 300, and 1000 as follows:

```
200 REM**READ BOX DESCRIPTION
210 READ CN,COL,ROW,W,H

300 REM**DRAW THE BOX
310 COLOR CN
320 PLOT COL,ROW
330 DRAWTO COL+W-1,ROW
340 DRAWTO COL+W-1,ROW+H-1
350 DRAWTO COL,ROW+H-1
360 DRAWTO COL,ROW

1000 REM**BOX DESCRIPTIONS:CN,COL,ROW,W,H
1010 DATA 1,3,3,11,11
1020 DATA 2,10,10,11,6
1030 DATA 3,30,0,7,12
```

To draw a horizontal line make H=0.

To draw a vertical line, make W=0.

To plot a single point, make W=0 and H=0.

BEYOND GRAPHICS 3

GRAPHICS 3 is a *low-resolution* mode. You can paint with a finer brush in GRAPHICS 5 or GRAPHICS 7, using the same colors as in GRAPHICS 3.

- GRAPHICS 5 (or GR. 5) gives you 80 columns (0 to 79) and 40 rows (0 to 39) in the top part of the screen, plus a text window.
- GRAPHICS 7 (or GR. 7) gives you 160 columns (0 to 159) and 80 rows (0 to 79) in the top part of the screen, plus a text window.

For both GR. 5 and GR. 7, PLOT, DRAWTO, COLOR, and SETCOLOR work the same as in GR. 3. The normal foreground colors are orange (COLOR 1), light green (COLOR 2), and dark blue (COLOR 3). The normal background color is black.

EXPERIMENT! In Gr. 5 and GR. 7, put an orange point in the upper left corner, a light green point near the center, and a dark blue point in the bottom right corner of the screen.

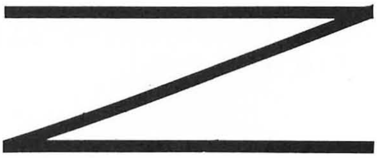
```
GR.5
COLOR 1:PLOT 0,0
COLOR 2:PLOT 40,20
COLOR 3:PLOT 79,39
```

```
GR.7
COLOR 1:PLOT 0,0
COLOR 2:PLOT 80,40
COLOR 3:PLOT 159,79
```

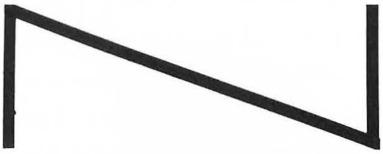
Look carefully to see the dark blue tiny rectangle in the bottom right corner of the graphics part of the screen!

Draw some lines:

```
GR.5
COLOR 3:PLOT 0,0
DRAWTO 79,0
COLOR 2:DRAWTO 0,39
COLOR 3:DRAWTO 79,39
```



```
GR.7
COLOR 3:PLOT 0,79
DRAWTO 0,0
COLOR 2:DRAWTO 159,79
COLOR 3:DRAWTO 159,0
```



REMEMBER:

	columns	rows
GRAPHICS 3	40	20
GRAPHICS 5	80	40
GRAPHICS 7	160	80

Now try all our (and your) GRAPHICS 3 programs in GRAPHICS 5 and GRAPHICS 7.

Questions

1. Here is a program to paint alternate orange and light green horizontal stripes on the entire top part of the screen:

```
100 REM**HORIZONTAL STRIPES
110 GR. 3

200 REM**PAINT STRIPES
210 FOR ROW=0 TO 18 STEP 2
220   COLOR 1
230   PLOT 0,ROW: DRAWTO 39,ROW
240   COLOR 2
250   PLOT 0,ROW+1: DRAWTO 39,ROW+1
260 NEXT ROW

300 REM**THAT'S ALL
310 END
```

(a) If you change line 110 to 110 GR. 5 and run the program, what will the screen look like?____________________

(b) If you change line 110 to 110 GR. 7 and run the program, what will the screen look like?____________________

2. Write a program to paint alternating orange and light green vertical stripes across the entire top part of the screen in GRAPHICS 3.

3. Modify the HORIZONTAL STRIPES program so the computer fills the entire top part of the screen with horizontal orange and light green stripes:

(a) in GRAPHICS 5

(b) in GRAPHICS 7

4. Modify your VERTICAL STRIPES program so the computer fills the entire top part of the screen with vertical orange and light green stripes:

(a) in GRAPHICS 5

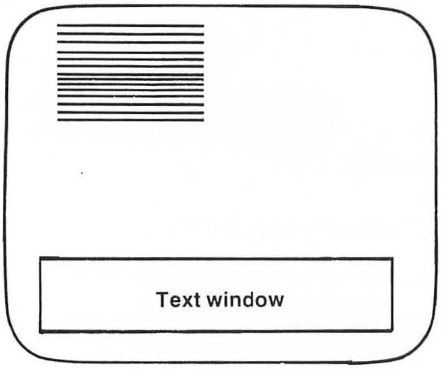
(b) in GRAPHICS 7

5. Write a program to fill the screen with alternating diagonal orange and light green stripes.

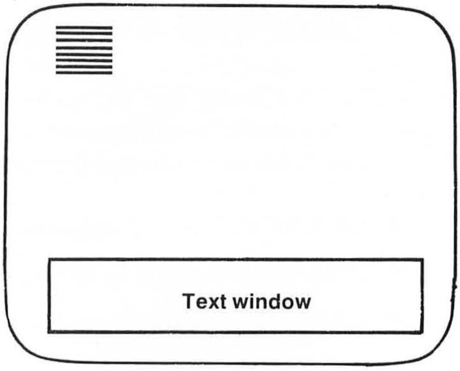
Answers

1. The striped part of the screen will be smaller in both cases.

(a)



(b)



2.

```
100 REM**VERTICAL STRIPES
110 GR. 3

200 REM**PAINT STRIPES
210 FOR COL=0 TO 38 STEP 2
220   COLOR 1
230   PLOT COL,0: DRAWTO COL,19
240   COLOR 2
250   PLOT COL+1,0: DRAWTO COL+1,19
260 NEXT COL

300 REM**THAT'S ALL
310 END
```

3. Here are the *changes* to HORIZONTAL STRIPES:

```
(a) 110 GR. 5
    210 FOR ROW=0 TO 38 STEP 2
    230   PLOT 0,ROW: DRAWTO 79,ROW
    250   PLOT 0,ROW+1: DRAWTO 79,ROW+1


(b) 110 GR. 7
    210 FOR ROW=0 TO 78 STEP 2
    230   PLOT 0,ROW: DRAWTO 159,ROW
    250   PLOT 0,ROW+1: DRAWTO 159,ROW+1
```

4 and 5. No answers. You will know when your programs work by seeing the stripes on the screen!

GRAPHICS 4 and 6

GRAPHICS 3,5 and 7 are four-color modes: background color and three foreground colors available at any one time. GRAPHICS 4 and 6 are two-color modes: background color and only one foreground color.

- GRAPHICS 3 has 40 columns (0-39) and 20 rows (0-19).
- GRAPHICS 4 and 5 have 80 columns (0-79) and 40 rows (0-39).
- GRAPHICS 6 and 7 have 160 columns (0-159) and 80 rows (0-79).

You can use GRAPHICS 4 and 6 in the same way as GRAPHICS 3,5 and 7 except that you are limited to COLOR 1. Use SETCOLOR 0 to change the color. Here is a handy table showing the colors in GRAPHICS 3 through 7:

GRAPHICS SCREEN	GR.3,5,7	GR. 4,6
SE. 0,____,____	COLOR 1	COLOR 1
SE. 1,____,____	COLOR 2	Not available
SE. 2,____,____	COLOR 3	Not available
SE. 4,____,____	Background	Background

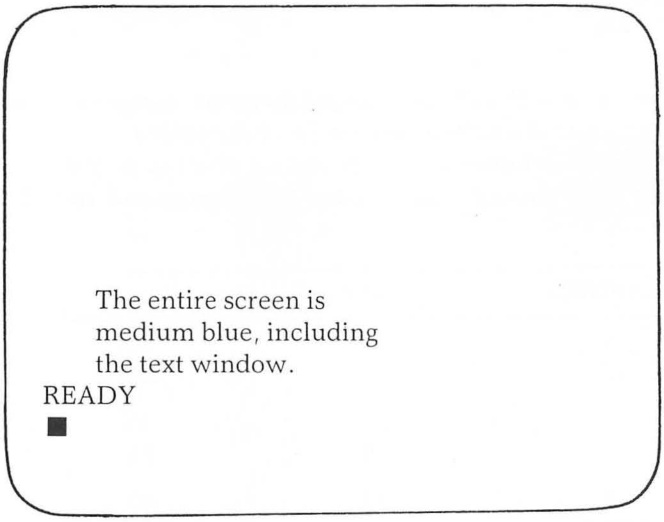
TEXT WINDOW		
SE. 1,____,____	Luminance of characters	Luminance of characters
SE. 2,____,____	Color & luminance	Color & luminance

Perhaps you wonder why anyone would use GRAPHICS 4 or 6. The reason is somewhat technical. The computer uses less memory in GRAPHICS 4 than in GRAPHICS 5. It uses less memory in GRAPHICS 6 than in GRAPHICS 7. Nevermind — use GRAPHICS 5 & 7 for now. When you become more advanced and write complex programs, you will sometimes have to use GRAPHICS 4 and 6.

GRAPHICS 8

GRAPHICS 8 is a high resolution graphics mode with 320 columns (0-319) and 160 rows (0-159). The price you pay is loss of color. GRAPHICS 8 is a one-color mode with the same color for background and foreground. Make the luminances different in order to see what you are plotting or drawing. Try this:

Type GR. 8 and press RETURN.



The entire screen is
medium blue, including
the text window.

```
READY
■
```

Now do the following, one line at a time, and watch what happens:

PLOT 0,0: DRAWTO 319,159	
SE. 1,0,14	Brightest line
SE. 1,0,0	Darkest line
SE. 2,0,14	White screen
SE. 2,5,14	Light purple screen
SE. 2,12,8	Medium green screen
SE. 1,0,14	Brightest line
SE. 4,4,8	Pink border

In GRAPHICS 8,

SETCOLOR 1,____,____ Background color and luminance
SETCOLOR 2,0,____ Foreground luminance only
SETCOLOR 4,____,____ Border color and luminance

YOU CAN ELIMINATE THE TEXT WINDOW

In GRAPHICS 1 through 8, you can eliminate the text window and use the entire screen for graphics. To do so, add 16 to the graphics mode number. You can do this only within a program. Try it with GR. 19, the full screen counterpart of GR. 3:

```
100 REM**LOOK! NO TEXT WINDOW
110 GR. 19
120 COLOR 1: PLOT 0,0
130 DRAWTO 39,23
140 GOTO 140
```

Run this program. You will see a very jagged diagonal orange line from the top left corner of the screen to the bottom left corner of the screen.

GRAPHICS 19 has 40 columns (0 to 39) and 24 rows (0 to 23) with no text window. Here is a table showing the number of columns and rows for all our graphics modes:

GRAPHICS	Columns	Rows
1	20	20
17	20	24
2	20	10
18	20	12
3	40	20
19	40	24
4	80	40
20	80	48
5	80	40
21	80	48
6	160	80
22	160	96
7	160	80
23	160	96
8	320	160
24	320	192

Modify our LOOK! NO TEXT WINDOW program for GRAPHICS 20,21,22,23 and 24. Also write a program to try out GRAPHICS 17 and 18, using PRINT #6.

The GTIA Chip

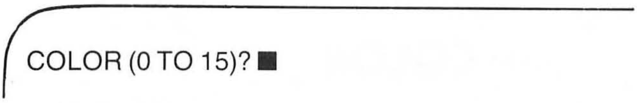
Older ATARI computers used a graphics chip called CTIA, which provided the graphics modes discussed up till now. New computers, including the 600XL and 800XL, contain a more powerful graphics chip called the GTIA chip.

The GTIA chip provides three more graphics modes: 9, 10 and 11. Each mode has 80 columns numbered 0 to 79 and 192 rows numbered 0 to 191. We will briefly touch on modes 9 and 11, but not mode 10, which requires more advanced programming techniques using the POKE statement.

In GRAPHICS 9, you can use any one of 16 colors at a time. After you choose a color, you can put up to 16 different shades (luminances) of the color on the screen. To see it, try this program:

```
100 REM**GRAPHICS 9 DEMO
110 GR. 0

200 REM**ASK FOR A COLOR NUMBER
210 PR. "COLOR (0 TO 15)";: IN. CN

300 REM**PUT 16 SHADES ON SCREEN
310 GR. 9
320 SE. 4,CN,0
330 FOR SHADE=0 TO 15
340   COLOR SHADE
350   W = 5*SHADE
360   FOR COL=W TO W+4
370     PLOT W,0: DRAWTO W, 191
380   NEXT COL
390 NEXT SHADE

400 REM**DO NOTHING LOOP
410 GOTO 410
```

When you run the program, it begins this way:



```
COLOR (0 TO 15)? ■
```

When you enter a color number and press RETURN, the entire screen becomes the color you chose. Then the computer draws vertical bands of that color with the darkest stripes on the left side of the screen and the lightest on the right side. Each strip is five columns wide.

In GRAPHICS 9, use the COLOR statement to select the shade (see line 340). Use SETCOLOR 4 to select the color.

SE. 4,____,O

color, 0 to 15

In GRAPHICS 11, you can put 16 different colors on the screen at the same time. You can also control the luminance. Try this program:

```
100 REM**GRAPHICS 11 DEMO
110 GR. 11

200 REM**COLOR BARS
210 FOR C=0 TO 15
220   COLOR C
230   W = 12*C
240   FOR ROW=W TO W+11
250     PLOT 0,ROW: DRAWTO 79,ROW
260   NEXT ROW
270 NEXT C

300 REM**CONTINUALLY CHANGE LUMINANCE
310 FOR LUM=0 TO 14 STEP 2
320   SE. 4,0,LUM
330   TD = 40
340   FOR Z=1 TO TD: NEXT Z
350 NEXT LUM
360 GOTO 310
```

Also try TD=1 and TD=106

RUN this program. First, the computer draws 16 color bars (the top one is black). The it zips through 16 luminances again and again and again—until you press BREAK.

In GRAPHICS 11, use COLOR to select any of 16 colors and SETCOLOR 4 to select only the luminance of all colors on the screen:

COLOR ____

color number, 0 to 15

SETCOLOR 4, 0, ____

luminance, 0 to 14

SELF-TEST

Well, it looks as if you have painted yourself into another Self-Test. Have a colorful time!

1. In graphics modes 1 through 8, the screen is split into two parts.

   (a) How many columns does the bottom part have? ____

   (b) How many rows does the bottom part have? ____

   (c) What is the bottom part of the screen called? ________

2. For graphics modes 1 through 8, complete the following table showing the number of columns and rows in the top part of the screen:

GRAPHICS	1	2	3	4	5	6	7	8
Columns	20							
Rows	20							

3. In GRAPHICS 17 through 24, there is no text window. The entire screen is used for big letters (GR. 17 and GR. 18) or rectangles and lines of colored light (GR. 19 through 24). Complete the following table showing the number of columns and rows in GRAPHICS 17 through 24:

GRAPHICS	17	18	19	20	21	22	23	24
Columns	20							
Rows	24							

4. Write a GRAPHICS 1 program to ask for a name, "bounce" the name for the left edge of the screen to the right edge and back and then go back for a new name. Modify your program for GRAPHICS 2.

5. Write a GRAPHICS 18 program to print ABCD on the screen. Use these colors:

Background	A	B	C	D
white	orange	light green	dark blue	red

6. Modify your program for question 5 to alternate the above colors with these colors:

A	B	C	D
light blue	medium pink	medium green	light purple

7. Write direct statements to draw an orange line connecting the top left corner of the graphics screen to the bottom right corner:

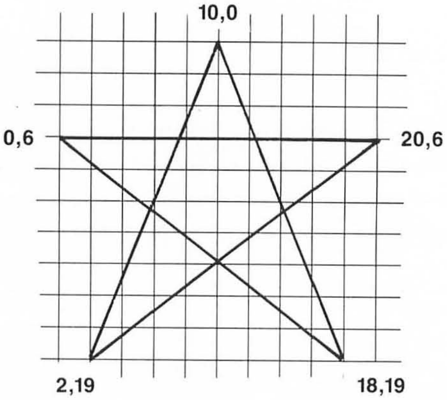
(a) GR. 3
PLOT ____,____
DRAWTO ____,____

(c) GR. 7 (OR GR. 6)
____
____

(b) GR. 5 (or GR. 4)
PLOT ____,____: DRAWTO ____,____

(d) GR. 8
____

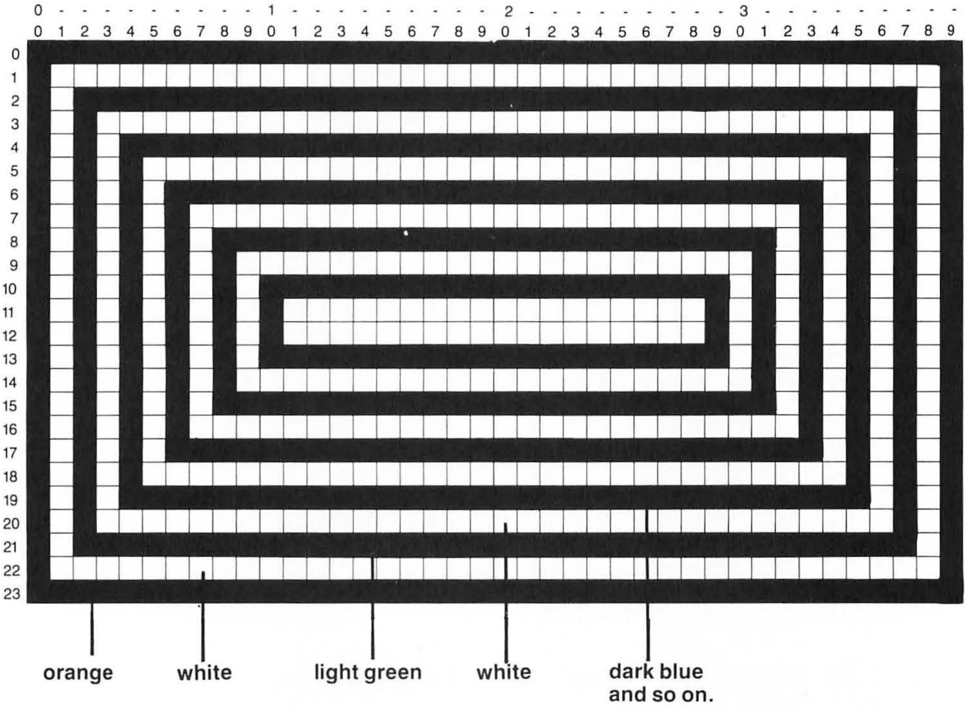
8. Write a program to draw the star below in GRAPHICS 3,5 and 7. Make the star medium purple on a white background. Use a time delay of a few seconds to allow you to see the star in each graphics mode.



9. Write a program to draw triangles on the screen. You choose the graphics mode. Information for the triangles is contained in DATA statements. For each triangle, the data consists of the column and row for each vertex and the color number (1, 2 or 3). For example:

```
1010 DATA 3,4, 10,8, 2,11, 1
                           |
                     color number
```

10. Write a program to fill the screen with nested rectangles that alternate in color: orange, white, light green, white, dark blue, white and so on. Write programs for graphics modes 3,5,7,19,21 and 23. In GR. 19 the screen should look this way:



Answers to Self-Test

1. (a) 40 (b) 4 (c) text window

2.

GRAPHICS	1	2	3	4	5	6	7	8
Columns	20	20	40	80	80	160	160	320
Rows	20	10	20	40	40	80	80	120

3.

GRAPHICS	17	18	19	20	21	22	23	24
Columns	20	20	40	80	80	160	160	320
Rows	24	12	24	48	48	96	96	192

4. Here is our GRAPHICS 1 program:

```
100 REM**BOUNCE NAME RIGHT & LEFT
110 GR. 1
120 DIM NAME$(19)

200 REM**GET THE NAME
210 PR. CHR$(125)
220 PR. "YOUR NAME";: IN. NAME$

300 REM**COMPUTE LENGTH OF NAME
310 LN = LEN(NAME$)

400 REM**PUT NAME AT LEFT EDGE
410 ROW = 10
420 POS. 0,ROW: PR. #6; NAME$
430 GOSUB 910
```

(continued)

```
500 REM**MOVE NAME TO RIGHT EDGE
510 FOR COL=1 TO 20-LN
520   POS. COL-1,ROW: PR. #6; " "
530   POS. COL,ROW: PR. #6; NAME$
540   GOSUB 910
550 NEXT COL

600 REM**MOVE NAME TO LEFT EDGE
610 FOR COL=19-LN TO 0 STEP -1
620   POS. COL+LN,ROW: PR. #6; " "
630   POS. COL,ROW: PR. #6; NAME$
640   GOSUB 910
650 NEXT COL

700 REM**GO GET ANOTHER NAME
710 GOTO 210

900 REM**SOUND DELAY SUBROUTINE
910 SOUND 0,100-5*COL
920 TD = 20
930 FOR Z=1 TO TD: NEXT Z
940 SOUND 0,0,0,0
950 RETURN
```

To try our program in GRAPHICS 2, make these changes:

```
110 GR. 2
410 ROW = 5
```

```
5.  100 REM**ABCD IN GRAPHICS 18
    110 GR. 2

    200 REM**BACKGROUND COLOR
    210 SE. 4,0,14

    300 REM**ABCD IN 4 COLORS
    210 PR. #6; "A";
    320 PR. #6. "b";  lower case
    330 PR. #6; "C";  inverse video caps
    340 PR. #6; "d";  inverse video lower case

    400 REM**DO NOTHING LOOP
    410 GOTO 410
```

6. Make changes beginning at block 400.

```
400 REM**CHANGE COLORS
410 SE. 0,9,8
420 SE. 1,4,8
430 SE. 2,12,8
440 SE. 3,5,14
450 GOSUB 910

500 REM**ORIGINAL COLORS
510 SE. 0,2,8
520 SE. 1,12,14
530 SE. 2,8,0
540 SE. 3,2,6
550 GOSUB 910

600 REM**GO AROUND AGAIN
610 GOTO 410

900 REM**TIME DELAY SUBROUTINE
910 TD = 1000
920 FOR Z=1 TO TD: NEXT Z
930 RETURN
```

7. (a) GR. 3
PLOT 0,0
DRAWTO 39,19

(b) GR5
PLOT 0,0
DRAWTO 79,39

(c) GR. 7
PLOT 0,0
DRAWTO 159,79

(d) GR. 8
PLOT 0,0: DRAWTO 319,159

In GR. 8 you might also want to make the foreground luminance very light or very dark.

8.
```
100 REM**STAR IN GR. 3,5,7
110 FOR GR=3 TO 7 STEP 2
120   GR. GR
130   SE. 4,0,14:SE.0,5,8
140   COLOR 1: PLOT 10,0
150   DRAWTO 18,19
160   DRAWTO 0,6
170   DRAWTO 20,6
180   DRAWTO 2,19
190   DRAWTO 10,0
200   GOSUB 910
210 NEXT GR

300 REM**GO DO IT AGAIN
310 GOTO 110

900 REM**TIME DELAY SUBROUTINE
910 TD = 2000
920 FOR Z=1 TO TD: NEXT Z
930 RETURN
```

9. We wrote our program for GRAPHICS 7.

```
100 REM**TRIANGLES
110 GR. 7

200 REM**READ TRIANGLE DESCRITPION
210 READ C1,R1,C2,R2,C3,R3,CN

300 REM**DRAW TRIANGLE
310 COLOR CN
320 PLOT C1,R1
330 DRAWTO C2,R2
340 DRAWTO C3,R3
350 DRAWTO C1,R1

400 REM**GO DRAW ANOTHER
410 GOTO 210
```

(continued)

```
1000 REM**TRIANGLE DESCRIPTIONS
1010 DATA 3,4,10,8,2,11,1
1020 DATA 10,7,14,1,7,2
1030 DATA 20,0,25,5,15,5,3
```

10. As usual, we leave at least one question without an answer. Good luck on this one. We know you can do it.

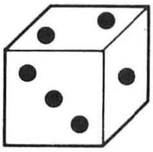
Chapter
Eight

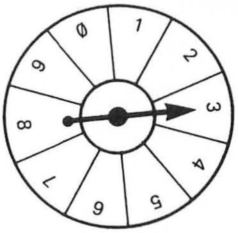
Meandering with Random Numbers

In this chapter, you will learn to use the mysterious and unpredictable RND function. You will use the RND *function* to compute *random numbers* and use them in programs to surprise and delight yourself and your friends. With RND, your programs can take on an air of friendly unpredictability, and people will wonder. . .what next?

When you finish this chapter, you will be able to:

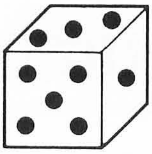
- Use the RND function to compute random whole numbers
- Tell the computer to make random music that sounds awful
- Make names and other strings of characters meander aimlessly about the screen
- Invent your own random constellations (you get to name them)
- Splash random blips of color here, there, anywhere on the screen
- Create ever-changing symmetric mandalas
- Put random color stripes or rectangles on the screen
- Simulate coin flipping and dice rolling
- Create adventurers for games such as *Dungeons & Dragons*, *RuneQuest*, and *Tunnels & Trolls*





RANDOM NUMBERS

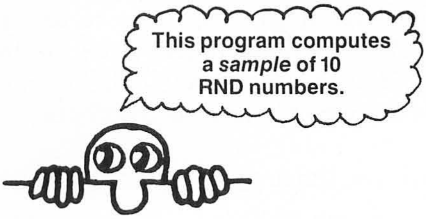
Random numbers are numbers that are chosen at random from a given set of numbers. A die has six sides, numbered 1 through 6. Roll one die to get a random number from 1 to 6.



Flip a coin — two possibilities: heads or tails. If it comes up HEADS, call it 1; if it comes up TAILS, call it 2. Flipping a coin gives a random number: 1 or 2.

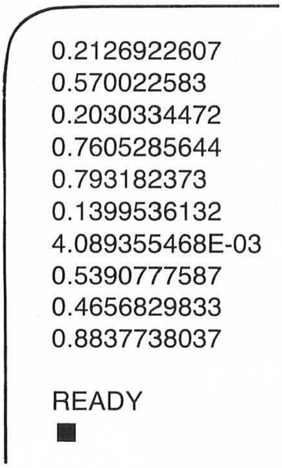
In ATARI BASIC you may use the RND function to get a random number. The following program uses the RND function to print random numbers on the screen:

```
10 REM**RND NUMBERS
20 GR. 0
30 FOR K=1 TO 10
40   PRINT RND(0)
50 NEXT K
```



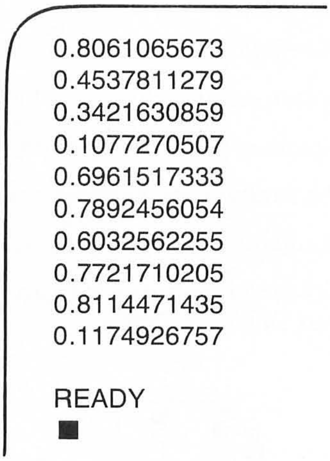
We ran the program twice. Here is what happened:

First RUN:



```
0.2126922607
0.570022583
0.2030334472
0.7605285644
0.793182373
0.1399536132
4.089355468E-03
0.5390777587
0.4656829833
0.8837738037

READY
■
```

Second RUN:



```
0.8061065673
0.4537811279
0.3421630859
0.1077270507
0.6961517333
0.7892456054
0.6032562255
0.7721710205
0.8114471435
0.1174926757

READY
■
```

Two runs produced very different sets of numbers. That's the idea of random numbers. They are, well, random!

Look over the two lists of numbers. Every number is greater than zero. Yes, even 4.089355468E-03. This is a *floating point number*. Every number is less than one. Another way to read 4.089355468E-03 is 0.00489355468. (See Appendix B for more information on floating point numbers.)

From the evidence, it seems that an RND number is greater than zero and less than one. However, we haven't shown much evidence — only 20 numbers. We suggest you run a bunch of RND numbers on your ATARI to get more evidence. (But remember! Evidence is not proof.)

It's true. RND numbers *are* greater than zero and less than one. Another way to say it: RND numbers are *between* 0 and 1. Or, in still another way:

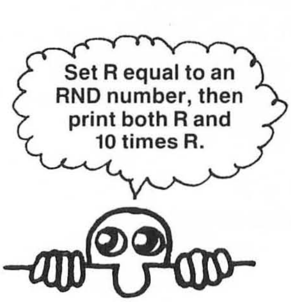
0 < RND(0) <1

Use any number in parentheses following RND. This number has no effect, but still must be there. We will usually put zero (0) in parentheses.

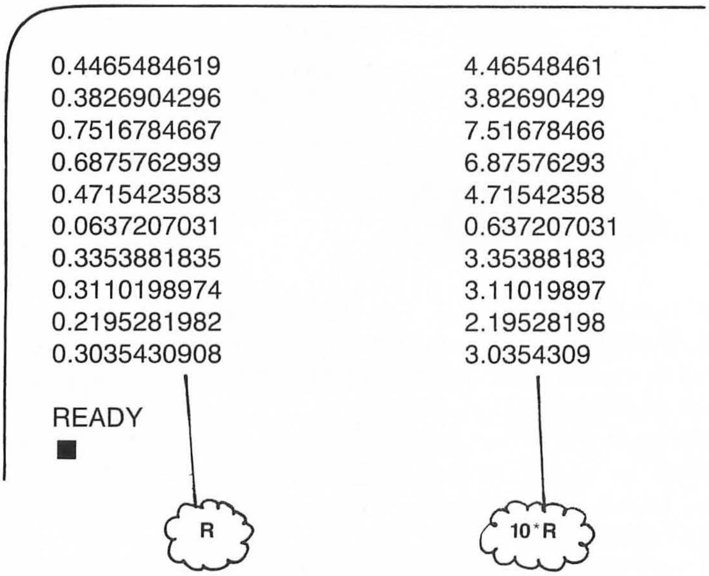
RND numbers between zero and one are not always convenient. Usually, we want whole numbers in some range. For example, we might want the numbers 1,2,3,4,5 and 6 at random; or decimal digits 0 through 9; or numbers from 1 to 100.

Hmmm...RND(0) is between 0 and 1, but is never 0 or 1. Therefore, 10*RND(0) must be a number between 0 and 10, but never 0 and never 10. Do you agree? If not, run this program a few times:

```
10 REM**10 TIMES RND
20 GR. 0
30 FOR K=1 TO 10
40   R = RND(0)
50   PR. R, 10*R
60 NEXT K
```


Set R equal to an RND number, then print both R and 10 times R.

Here is a sample:



```
0.4465484619          4.46548461
0.3826904296          3.82690429
0.7516784667          7.51678466
0.6875762939          6.87576293
0.4715423583          4.71542358
0.0637207031          0.637207031
0.3353881835          3.35388183
0.3110198974          3.11019897
0.2195281982          2.19528198
0.3035430908          3.0354309

READY
■
```

All values of 10*R are greater than zero and less than ten. Each of these values can be thought of as having an *integer* part to the left of the decimal point and a *fractional* part to the right of the decimal point.

4.46548461
integer part
fractional part
0.637207031
integer part
fractional part

For each number between 0 and 10, the integer (whole number) part is a single digit. Wouldn't it be nice if you could tell the computer to throw away the fractional part and keep the integer part?

Well, you can. ATARI BASIC has a clever and useful function called INT. Here are some examples:

```
INT(3) = 3                      INT(7) = 7
INT(4.46548461) = 4             INT(0.637207031) = 0
```

If X is any positive number (or zero), INT(X) is the integer part of X.

Instead of a number, you can put any numeric variable, function or expression in parentheses following INT.

INT(_____)

Any ATARI BASIC number, numeric variable, function, or expression.

So, it is OK to write INT(10*RND(0)).

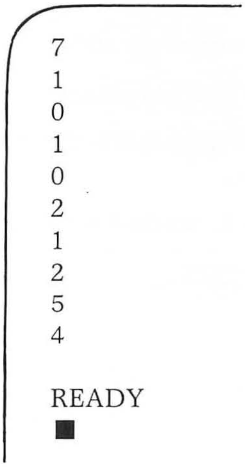
RND(0)	is a random number between 0 and 1.
10*RND(0)	is a random number between 0 and 10.
INT(10*RND(0))	is a random integer from 0 to 9.

Now run this program to put random digits (0 to 9) on the screen:

```
10 REM**RANDOM DIGITS, 0 TO 9
20 GR. 0
30 FOR K=1 TO 10
40   PR. INT(10*RND(0)
50 NEXT K
```

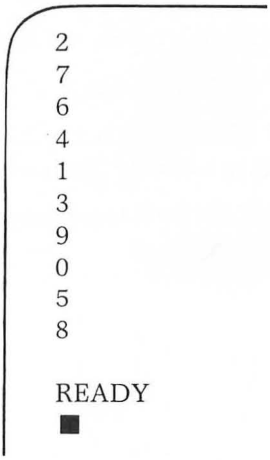
We ran the program twice and got these results:

First RUN:



```
7
1
0
1
0
2
1
2
5
4

READY
■
```

Second RUN:



```
2
7
6
4
1
3
9
0
5
8

READY
■
```

Questions

1. The RND function gives a random number between ____ and ____.
2. Complete the following:

   (a) 2*RND(0) is between ____ and ____.

   (b) 6*RND(0) is between ____ and ____.

   (c) 100*RND(0) is between ____ and ____.
3. Complete the following:

   (a) INT(2*RND(0)) is ____ or ____.

   (b) INT(6*RND(0)) + 1 is ____ or ____ or ____ or ____ or ____ or ____.

   (c) INT(100*RND(0)) is an integer from ____ to ____.

Answers

1. The RND function gives a random number between 0 and 1. RND(0) is never 0 or 1, but always somewhere between.
2. (a) 2*RND(0) is between 0 and 2. Never 0 and never 2.

   (b) 6*RND(0) is between 0 and 6. Never 0 and never 6.

   (c) 100*RND(0) is between 0 and 100. Never 0 and never 100.
3. (a) INT(2*RND(0)) is 0 or 1. These are the only possible values.

   (b) INT(6*RND(0)) + 1 is 1 or 2 or 3 or 4 or 5 or 6. Later in this chapter we will use this to simulate (imitate) rolling one die.

   (c) INT(100*RND(0)) is an integer from 0 to 99. It can be 0 or 1 or 2 or 3, and so on, up to 99.

EXPERIMENT! Use the following program to print random numbers from 1 to N, where you supply the value of N:

```
100 REM**RND NUMBERS, 1 TO N
110 GR. 0

200 REM**TELL WHAT TO DO
210 PR. CHR$(125)
220 PR. "I'LL PRINT RND NUMBERS FROM"
230 PR. "1 TO N. YOU SELECT N."
240 PR.: PR. "N=";: IN. N

300 REM**PRINT THE RND NUMBERS
310 PR.
320 PR. INT(N*RND(0))+1,
330 FOR Z=1 TO 50: NEXT Z
340 GOTO 320
```

Just to see what happens, also try this variation of line 320, with a semicolon at the end:

```
320 PR. INT(N*RND(0))+1;
```

RANDOM MUSIC

OK, so RND gives random numbers. But what are they good for? How about some random music?

```
100 REM**RANDOM MUSIC
110 GR. 0
120 N = INT(255*RND(0))+1
130 SO. 0,N,10,10
140 TD = 100
150 FOR Z=1 TO 100: NEXT Z
160 SO. 0,0,0,0
170 GOTO 120
```

Line 120 assigns to N a value that is a random integer in the range 1 to 255. Enter and run the program. You will hear "music" consisting of a succession of random tones. To eliminate high tones you can hardly hear, change line 120 as follows:

```
120 N = INT(200*RND(0))+56
```

This gives random integers from 56 to 255. If you want a few more of the higher notes, change line 120 again:

```
120 N = INT(220*RND(0))+36
```

Now N can be a random integer from 36 to 255.

Let's hear how it sounds with a random duration for each tone. Change line 140 as follows:

```
140 TD=INT(190*RND(0))+11
```

This gives a random time delay from 11 to 200.

One more thing. Add some color. Make the screen color and luminance random. Add the following lines:

```
133 C = INT(16*RND(0))
135 L = INT(16*RND(0))
136 SE.2,C,L
```

Now, if you like chaos, run the modified program and enjoy what you see and hear.

Questions

1. For each of the following, describe the possible random numbers.

   (a) INT(3*RND(0))+2
   Possible numbers: ____________________

   (b) INT(10*RND(0))+10
   Possible numbers: ____________________

   (c) 2*INT(8*RND(0))
   Possible numbers: ____________________

____

2. Complete the following program to do random "bursts" of music. Each burst is a rapid sequence of tones from a LO tone to a HI tone. (Remember, LO tones have higher *numbers* than HI tones.) Let the HI tone be a random number from 10 to 100 and the LO tone be a number from 101 to 255.

```
100 REM**RANDOM BURSTS OF MUSIC
110 GR. 0

200 REM**COMPUTE LO & HI TONE NUMBERS

210 LO = ________________________

220 HI = ________________________

300 REM**PLAY SOME MUSIC
310 FOR N = ________________________
320   SO. 0,N,10,10
330 NEXT N

400 REM**GO FOR NEW LO & HI
410 GOTO 210
```

Answers

1. (a) 2,3,4 (b) 10,11,12,13,14,15,16,17,18,19

   (c) 0,2,4,6,8,10,12,14

2.
```
210 LO=INT(155*RND(0))+101
220 HI=INT(91*RND(0))+10
310 FOR N=LO TO HI STEP -1
```

Did you forget STEP − 1? Remember, low notes have high numbers and high notes have low numbers. The value of LO is always more than the value of HI, so you must use STEP − 1.

SKITTERY NAME

Use random numbers to put a name here, there, anywhere on the screen. Here for a moment, there for a moment, somewhere else for a moment.

```
100 REM**SKITTERY NAME
110 DIM NAME$(20)
120 GR. 0

200 REM**GET A NAME
210 POKE 752,0
220 PR. "WHAT IS YOUR NAME";:IN. NAME$
230 POKE 752,1

300 REM**RANDOM SCREEN POSITION
310 COL = INT(40*RND(0))
320 ROW = INT(24*RND(0))

500 REM**PRINT NAME AT COL,ROW
510 PR. CHR$(125)
520 POS. COL,ROW: PR. NAME$

600 REM**DELAY, THEN GO AROUND
610 TD = 200
620 FOR Z=1 TO TD: NEXT Z
630 GOTO 310
```

When you run the program, it begins this way:

```
WHAT IS YOUR NAME? ■
```

Type your name, press RETURN, and watch your name cavort about the screen. Press BREAK to stop the action and delete *only* line 510.

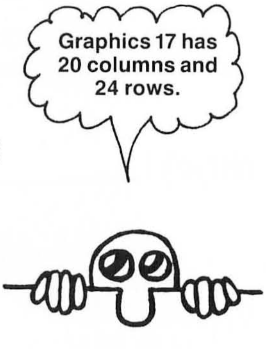
Type 510 and press RETURN. Run it again to see what happens.

Now try the program in GRAPHICS 17. We tried it with this program:

```
100 REM**SKITTERY NAME IN GR. 17
110 DIM NAME$(20)
120 GR. 0

200 REM**GET A NAME
220 PR. "WHAT IS YOUR NAME";:IN. NAME$
```

(continued)

```
300 REM**RANDOM SCREEN POSITION
310 COL = INT(20*RND(0))
320 ROW = INT(24*RND(0))

500 REM**PRINT NAME AT COL,ROW
510 GR. 17
520 POS. COL,ROW: PR. #6; NAME$

600 REM**DELAY, THEN GO AROUND
610 TD = 200
620 FOR Z=1 TO TD: NEXT Z
630 GOTO 310
```



Run this program, enter someone's name, and watch the name scamper about the screen in big letters. Then make these changes:

Delete line 510.
Add this line: 230 GR. 17

With these changes, the computer will enter GRAPHICS 17 at line 230. Since line 510 is gone, the screen will not be cleared each time before the name is printed. So the name will remain. The screen will slowly fill with the name of your choice.

Questions

1. Change the SKITTERY NAME program so that a random screen color is chosen each time before the name is printed. Make the name itself either very light or very dark (your choice).
2. Change the SKITTERY NAME IN GR. 17 program to a SKITTERY NAME IN GR. 18 program. Now the letters will be even bigger!
3. Change the SKITTERY NAME IN GR. 17 program so the name is printed in a random color with random luminance.
4. Sometimes a name is printed so close to the right edge of the screen that it is broken and part of the name appears on the left edge of the next line down. For example, here is Annalee's name printed at column 38:

```
                  Column 38
                      I
                      AN

   NALEE
    I
Column 2
```

(a) Change the Graphics 0 SKITTERY NAME program so the name is not printed so close to the right edge that it is broken.

(b) Change the SKITTERY NAME IN GR. 17 program so the name is not printed so close to the right edge that it is broken.

Answers

1. We make the name very light: 230 SE. 1,0,14. We put our random screen color in block 400.

```
400 REM**RANDOM SCREEN COLOR
410 C = INT(16*RND(0))
420 SE. 2,C,8
```

More variations: Random luminance, random border color and/or luminance, random sound.

2. Make these changes:

```
320 ROW = INT(12*RND(0))
510 GR. 18
```

3. Delete line 510 and add these lines:

```
400 REM**RANDOM COLOR & LUMINANCE
410 GR. 17
420 C = INT(16*RND(0))
430 L = INT(16*RND(0))
440 SE. 0,C,L
```

Also try a white background: 450 SE. 4,0,14

4. We leave this one for you to ponder. Here's a hint: Use the LEN function in the line that computes the value of COL.

RANDOM COLOR BLIPS

The following program plots tiny rectangles on the screen in random places and in random colors:

```
100 REM**RANDOM COLOR BLIPS
110 GR. 19

300 REM**RANDOM SCREEN POSITION
310 COL = INT(40*RND(0))
320 ROW = INT(24*RND(0))

400 REM**RANDOM COLOR
410 C = INT(3*RND(0))+1
420 COLOR C

500 REM**PLOT THE BLIP
510 PLOT COL,ROW

600 REM**DELAY, THEN DO ANOTHER
610 TD = 100
620 FOR Z=1 TO TD: NEXT Z
630 GOTO 310
```

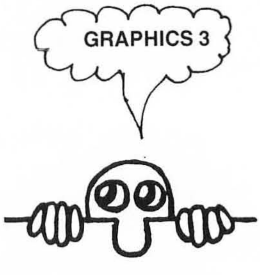
Run this program and watch the screen fill up slowly with orange, light green and dark blue tiny rectangles. Then, add some sound. Change block 600 as follows:

```
600 REM**SOUND,DELAY,GO AROUND
610 TD = 1
620 N = INT(255*RND(0))+1
630 SO. 0,N,10,10
640 FOR Z=1 TO TD: NEXT Z
650 GOTO 310
```

Try this variation with different values for TD in line 610. Experiment with different values of N in line 620.

Next, change the program as follows:

```
100 GR. 3

600 REM**SOUND,DELAY,GO AROUND
610 TD = 1
620 N = INT(255*RND(0))+1
630 SO. 0,N,10,10
640 FOR Z=1 TO TD: NEXT Z
650 SO. 0,0,0,0
660 GOTO 310
```



Note we didn't ask you to change line 320, even though you will now have a split screen. The computer will still "plot" points in rows 20,21,22 and 23. However, you won't see them because they are obscured by the text window. Go ahead and run this program. While the program is running, press BREAK. The computer stops and you can look at a static pattern.

Type CONT and press RETURN.

The pattern continues. Remember: You can CONTinue a program after pressing BREAK by typing CONT. While the program is stopped, change the value of TD in line 610, then type CONT and press RETURN. Do this several times until you get just the right sound for your ears.

Press BREAK to stop the program.

Type CONT to continue it from the place it stopped.

A mandala is a symmetrical pattern, nice to look at. A giant snowflake is beautifully symmetric about its center. Snowflakes are great mandalas but melt all too soon. Use this program to put an everchanging mandala on the screen.

```
100 REM**MANDALA, EVERCHANGING
110 GR. 19

300 REM**HORIZONTAL & VERTICAL OFFSET
310 H = INT(20*RND(0))
320 V = INT(12*NRD(0))
```

(continued)

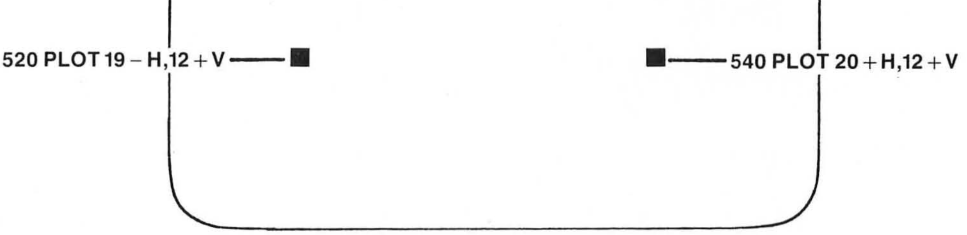
```
400 REM**RANDOM COLOR
410 C = INT(3*RND(0))+1
420 COLOR C

500 REM**TURN ON FOUR BLIPS
510 PLOT 19-H,11-V
520 PLOT 19-H,12+V
530 PLOT 20+H,11-V
540 PLOT 20+H,12+V

600 REM**DELAY, THEN GO AROUND
610 TD = 100
620 FOR Z=1 TO TD: NEXT Z
630 GOTO 310
```

Run the program. The computer turns on four blips at a time, symmetrical in the center of the screen. If it is happening too fast for you to see this, increase the time delay (TD) in line 610. If you want things to happen more rapidly, decrease the value of TD.

Lines 310 and 320 compute random values for H (horizontal offset) and V (vertical offset). H can be any integer from 0 to 19; V can be any integer from 0 to 11. H and V are used in block 500 to plot the four blips.

In line 510, the possible values of 19 – H are 0 (when H is 19) to 19 (when H is 0). The possible values of 11 – V are 0 (when V is 11) to 11 (when V is 0). So line 510 plots a blip in the upper left quadrant of the screen.

You can probably figure out that line 520 puts a blip in the lower left quadrant, line 530 puts a blip in the upper right quadrant, and line 540 puts a blip in the lower right quadrant.



EXPERIMENT! Try these variations.

VARIATION 1. Change block 300 this way:

```
300 REM**HORIZONTAL & VERTICAL OFFSET
310 H = INT(20*RND(0))
320 V = INT(13*RND(0))
325 V = INT(V*RND(0))
```

V will still be a number from 0 to 11, but smaller values are more likely — the smaller, the more likely! This will change the shape of the mandala, make it look wider and less tall.

VARIATION 2. This one biases the value of H toward smaller values. Use this block 300 to make the mandala seem taller and narrower. When we ran it, we saw "faces" appear early in the run.

```
300 REM**HORIZONTAL & VERTICAL OFFSET
310 H = INT(21*RND(0))
315 H = INT(H*RND(0))
320 V = INT(12*RND(0))
```

VARIATION 3. Combine the previous two variations to get a "round" looking mandala.

```
300 REM**HORIZONTAL & VERTICAL OFFSET
310 H = INT(21*RND(0))
315 H = INT(H*RND(0))
320 V = INT(13*RND(0))
325 V = INT(V*RND(0))
```

VARIATION 4. This leaves a "black cross" in the center.

```
300 REM**HORIZONTAL & VERTICAL OFFSET
310 H = INT(20*RND(0))
315 H = INT(H*RND(0))+1
320 V = INT(12*RND(0))
325 V = INT(V*RND(0))+1
```

MORE VARIATIONS. Try these:

VARIATION 5. Instead of a black background, use a white background, or a background of the color of your choice.

VARIATION 6. Select your own colors for COLOR 1, COLOR 2 and COLOR 3.

Questions

1. In the RANDOM COLOR BLIPS program, you choose the screen colors. Add block 200 to select colors for COLOR 1, COLOR 2, COLOR 3 and the background.
2. Modify the RANDOM COLOR BLIPS program for (a) GRAPHICS 21 and (b) GRAPHICS 23.
3. Add sound to the MANDALA, EVERCHANGING program. Make the tone number equal to H*V+30.
4. In the MANDALA, EVERCHANGING PROGRAM, change block 400 so the three foreground screen colors and luminances are selected at random.

Answers

1. We choose dark purple, medium green and light pink on a white background:

```
200 REM**SELECT SCREEN COLORS
210 SE. 0,5,2
220 SE. 1,12,8
230 SE. 2,4,12
240 SE. 4,0,14
```

You may also choose random colors and luminances. Whatever colors you choose, if you let the program run for several minutes, the screen colors will change every few seconds. Try it — it's a nice effect.

2. Make these changes:

```
(a) 110 GR. 21
    310 COL = INT(80*RND(0))
    320 ROW = INT(48*RND(0))
(b) 110 GR. 23
    310 COL = INT(160*RND(0))
    320 ROW = INT(96*RND(0))
```

3. We did it by changing block 600:

```
600 REM**SOUND,DELAY,GO AROUND
610 TD = 100
615 SO. 0,H*V+30,10,10
620 FOR Z=1 TO TD: NEXT Z
625 SO. 0,0,0,0
630 GOTO 310
```

Also try: **610 TD = 1**
And try some values of TD between 1 and 100.

4. We want random colors and luminances for color registers 0,1 and 2. Look for our new block 400 in this complete program:

```
100 REM**MANDALA, EVERCHANGING
110 GR. 19

300 REM**HORIZONTAL & VERTICAL OFFSET
310 H = INT(21*RND(0))
315 H = INT(H*RND(0))
320 V = INT(13*RND(0))
325 V = INT(V*RND(0))

400 REM**RANDOM COLORS
410 SE.0,INT(16*RND(0)),INT(16*RND(0))
420 SE.1,INT(16*RND(0)),INT(16*RND(0))
430 SE.2,INT(16*RND(0)),INT(16*RND(0))
440 COLOR INT(3*RND(0))+1

500 REM**TURN ON FOUR BLIPS
510 PLOT 19-H,11-V
520 PLOT 19-H,12+V
530 PLOT 20+H,11-V
540 PLOT 20+H,12+V

600 REM**SOUND DELAY,GO AGAIN
610 TD = 1
615 SO. 0,H*V+30,10,10
620 FOR Z=1 TO TD: NEXT Z
625 SO. 0,0,0,0
630 GOTO 310
```

We like this last program a lot. We like it even better with a white background:

```
120 SE.4,0,14
```

After watching it on a white background, we thought, Hmmm...how about making the background change randomly?" So we did:

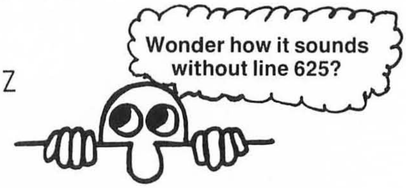
```
450 SE.4,INT(16*RND(0)),14
```

Try all variations: black background, white background, random background. And try other values of TD in line 610.

610 TD = 2	Slower,
610 TD = 4	and slower,
610 TD = 8	and slower,
and so on	and so on.

Try some different sound effects. For example:

```
600 REM**SOUND DELAY, GO AGAIN
610 TD = 1
613 FOR K=30 TO 32 ←——or 33 or 34 or . . .
615   SO. 0,H*V+K,10,10
617 NEXT K
620 FOR Z=1 TO TD: NEXT Z
625 SO. 0,0,0,0
630 GOTO 310
```



My, how time flies. We began watching MANDALA, EVERCHANGING on Tuesday and here it is Sunday already. How did that happen?

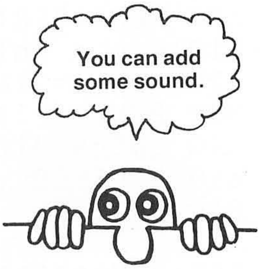
ZAPPY ARTIST

Zappy Artist loves to paint stripes. He began by painting horizontal stripes from somewhere on the screen to somewhere else on the screen. He likes to paint on a white screen. He also likes to see the colors change a lot.

```
100 REM**ZAPPY ARTIST #1
110 GR. 19
120 SE. 4,0,14

300 REM**RANDOM COORDINATES
310 ROW = INT(24*RND(0))
320 COL1 = INT(40*RND(0))
330 COL2 = INT(40*RND(0))

400 REM**RANDOM COLORS
410 SE.0,INT(16*RND(0)),INT(16*RND(0))
420 SE.1,INT(16*RND(0)),INT(16*RND(0))
430 SE.2,INT(16*RND(0)),INT(16*RND(0))
440 COLOR INT(3*RND(0))+1
```

(continued)

```
500 REM**DRAW A LINE
510 PLOT COL1,ROW
520 DRAWTO COL2,ROW

600 REM**DELAY & GO AGAIN
610 TD = 1
620 FOR Z=1 TO TD: NEXT Z
630 GOTO 310
```



OK, run it and watch Zappy Artist zap horizontal lines all over the screen. Sometimes he zaps from left to right, sometimes from right to left, depending on the values of COL1 and COL2. Try other time delays (TD) in line 610.

Well, as you might expect, Zappy got tired of horizontal lines and switched to vertical lines for a while. Then he got tired of that and dallied for a time with alternate horizontal and vertical lines. He does it like this:

```
100 REM**ZAPPY ARTIST #2
110 GR. 19
120 SE. 4,0,14

200 REM**DRAW TWO LINES
210 GOSUB 710
220 GOSUB 810

600 REM**DELAY & GO AGAIN
610 TD = 1
620 FOR Z=1 TO TD: NEXT Z
630 GOTO 210
```

Try other values of TD in line 610.

```
700 REM**HORIZONTAL LINE SUBROUTINE
710 ROW = INT(24*RND(0))
720 COL1 = INT(40*RND(0))
730 COL2 = INT(40*RND(0))
740 GOSUB 910
750 PLOT COL1,ROW: DRAWTO COL2,ROW
760 RETURN
```

This subroutine calls another subroutine.

(continued)

```
800 REM**VERTICAL LINE SUBROUTINE
810 COL = INT(40*RND(0))
820 ROW1 = INT(24*RND(0))
830 ROW2 = INT(24*RND(0))
840 GOSUB 910
850 PLOT COL,ROW1: DRAWTO COL,ROW2
860 RETURN

900 REM**RANDOM COLORS SUBROUTINE
910 SE.0,INT(16*RND(0)),INT(16*RND(0))
920 SE.1,INT(16*RND(0)),INT(16*RND(0))
930 SE.2,INT(16*RND(0)),INT(16*RND(0))
940 COLOR INT(3*RND(0))+1
950 RETURN
```

This subroutine calls another subroutine.

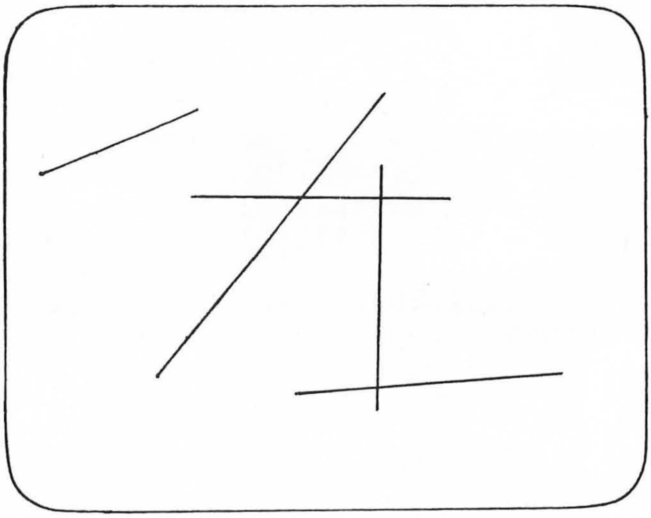
Questions

1. In ZAPPY ARTIST #2, block 200 calls two subroutines. Briefly explain what happens as the computer obeys the instructions in block 200.

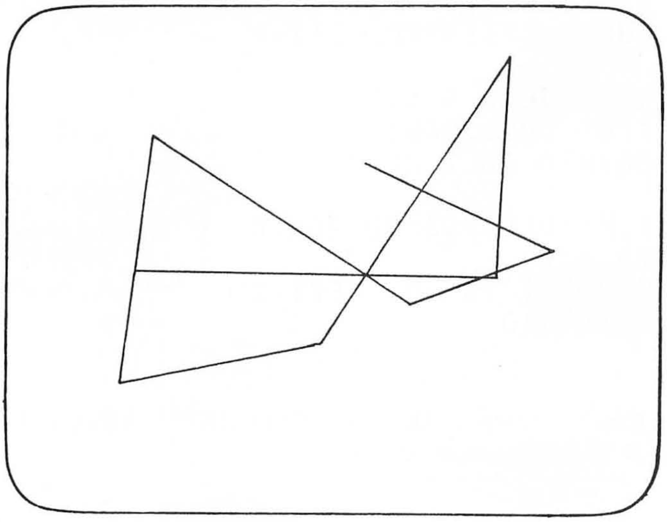
   ____

   ____

2. Write ZAPPY ARTIST #3, in which Zappy Artist draws random lines from anywhere on the screen to anywhere else. These lines can be horizontal, vertical, or at an angle.



3. Write ZAPPY ARTIST #4, in which Zappy Artist draws random lines where each line is connected to the previous line.



Answers

1. Line 210 (210 GOSUB 710) calls the HORIZONTAL LINE SUBROUTINE which draws a random horizontal line. This subroutine calls the RANDOM COLORS SUBROUTINE. After the line is drawn, the computer returns to line 220 and calls the VΞRTICAL LINE SUBROUTINE.

2. Fortunately, we had stored ZAPPY ARTIST #1 on both tape cassette and diskette. We loaded it and screen-edited it to get ZAPPY ARTIST #3. Time: less than two minutes:

```
100 REM**ZAPPY ARTIST #3
110 GR. 19
120 SE. 4,0,14

300 REM**RANDOM COORDINATES
310 COL1 = INT(40*RND(0))
320 ROW1 = INT(24*RND(0))
330 COL2 = INT(40*RND(0))
350 ROW2 = INT(24*RND(0))
```

```
400 REM**RANDOM COLORS
410 SE.0,INT(16*RND(0)),INT(16*RND(0))
420 SE.1,INT(16*RND(0)),INT(16*RND(0))
430 SE.2,INT(16*RND(0)),INT(16*RND(0))
440 COLOR INT(3*RND(0))+1

500 REM**DRAW A LINE
510 PLOT COL1,ROW1
520 DRAWTO COL2,ROW2

600 REM**DELAY & GO AGAIN
610 TD = 1
620 FOR Z=1 TO TD: NEXT Z
630 GOTO 310
```

Try other values of TD in line 610

3. We leave this one for you. Do it by editing ZAPPY ARTIST #1 or ZAPPY ARTIST #3. The changes are minor.

FANTASY ROLE PLAYING GAMES

Millions of people are playing fantasy role playing games. A role playing game is a game in which one or more players create and control characters (adventurers) who live their imaginary lives in a specially made game world. The game world is created, managed, and operated by a GameMaster (GM), also called a referee, adventure master, or Dungeon Master (DM).

Most people who play role playing games use a formal rule system. Some of the best known are listed below:

Dungeons & Dragons (D&D). From TSR, P.O. Box 756, Lake Geneva, WI 53147.

RuneQuest (RQ). From Chaosium, P.O. Box 6302, Albany, CA 94706.

Tunnels & Trolls (T&T). From Blade, Box 1467, Scottsdale, AZ 85252.

Beware! The rule books are difficult to read and understand. For beginners, we suggest two books written specifically for beginners:

Adventurer's Handbook: A Guide to Role Playing Games by Bob Albrecht and Greg Stafford.

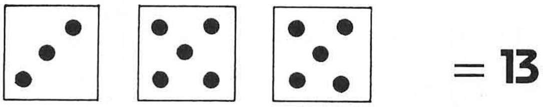
Through Dungeons Deep by Robert Plamondon.

Both are available from Reston Publishing Company, 11480 Sunset Hills Road, Reston, VA 22090.

To play a role playing game, you create one or more characters, then guide them through adventures in a universe created by a GameMaster. To create a character, you roll three six-sided dice several times. Each roll assigns a value to one of the basic characteristics of your character.

For starters, you will learn to use your computer to roll a beginning character as described in *Adventurer's Handbook* or *RuneQuest.* Your character will have seven basic characteristics: strength (STR), constitution (CON), size (SIZ), intelligence (INT), power (POW), dexterity (DEX), and charisma (CHA). These characteristics determine a character's ability to use weapons, fight, learn, use magic, sustain damage, lead others, survive and thrive in the GM's world.

A six-sided die has six sides numbered 1 to 6. To determine a characteristic, you roll three dice and add the numbers. For example:



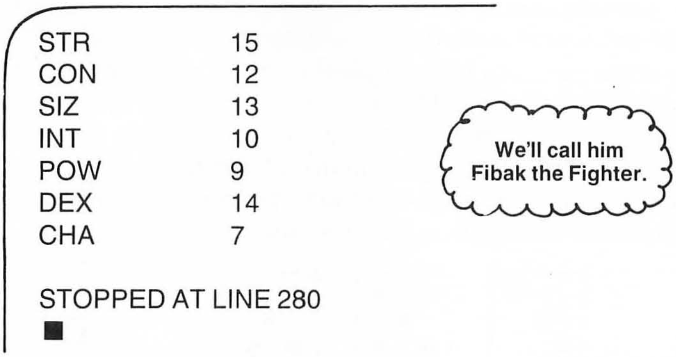
The smallest possible roll is 3 (1+1+1). The largest is 18 (6+6+6). An "average" roll is 10 or 11. You will get 10 or 11 about 25% of the time. A roll of 9 to 12 occurs about 48% of the time. Expect to get 9,10,11 or 12 about half of the time.

Instead of rolling dice, use your ATARI computer to roll a character.

```
110 REM**CREATE A CHARACTER
110 GR. 0

200 REM**ROLL CHARACTERISTICS
210 GOSUB 910: PR. "STR", DICE
220 GOSUB 910: PR. "CON", DICE
230 GOSUB 910: PR. "SIZ", DICE
240 GOSUB 910: PR. "INT", DICE
250 GOSUB 910: PR. "POW", DICE
260 GOSUB 910: PR. "DEX", DICE
270 GOSUB 910: PR. "CHA", DICE
280 STOP

900 REM**DICE SUBROUTINE
910 D1 = INT(6*RND(0))+1
920 D2 = INT(6*RND(0))+1
930 D3 = INT(6*RND(0))+1
940 DICE = D1 + D2 + D3
950 RETURN
```

We hope you use screen editing when you enter this program. Note how similar to each other are lines 210 through 270. After typing line 210, and pressing RETURN, edit line 210 to get line 220, then press RETURN. Edit line 220 to get line 230. And so on—edit line 910 to get line 920, then edit line 920 to get line 930.

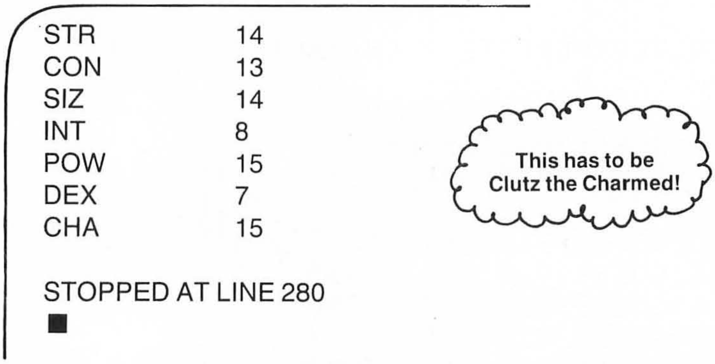
Use the program to roll a character. Your character might look like this (but probably won't).



```
STR        15
CON        12
SIZ        13
INT        10
POW        9
DEX        14
CHA        7

STOPPED AT LINE 280
■
```

Here is your first adventurer. He is big (SIZ=13), strong (STR=15), above average in ability to soak up damage (CON=12), has good dexterity (D=14), has average intelligence (INT=10). He will not be a magic user (POW=9) or a leader (CHA=7).

Roll another character.

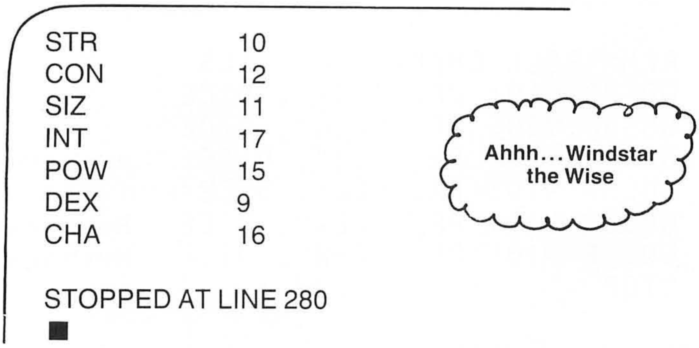


```
STR        14
CON        13
SIZ        14
INT        8
POW        15
DEX        7
CHA        15

STOPPED AT LINE 280
■
```

This is indeed a strange character! Big (SIZ=14), strong (STR=14), able to take damage (CON=13). But clumsy (DEX=7) and not very bright (INT=8). Look at power (POW=15). Very high. If our character had high intelligence, he could be a magic-user. However, he probably can't remember a spell and, even if he did, might use it on himself by accident. His power will show up as luck. Our character is very lucky.

Watch out! This character has high charisma (CHA=15). He will convince others to follow him into...what? A quandry...do we follow Clutz and trust to luck? Perhaps we follow a league behind.

Help! Roll another adventurer.



```
STR          10
CON          12
SIZ          11
INT          17
POW          15
DEX          9
CHA          16

STOPPED AT LINE 280
■
```

Saved! Our group is saved! Windstar the Wise wandered by, saw our forlorn little group of adventurers, and decided to take charge. She is a magic-user and leader — someone to trust, learn from, and follow.

The group of characters now numbers three: Fibak, Clutz, and Windstar. Too small a group. They need at least four more characters to survive and thrive in the GameMaster's world. So, you roll four more characters. More are OK. Tell who each character is and how he or she relates to and works with the other adventurers.

Questions

1. In *Tunnels & Trolls* (T&T), a character has six basic characteristics: strength (STR), constitution (CON), intelligence (IQ), luck (LK), dexterity (DEX), and charisma (CHR). Modify our CREATE A CHARACTER program to roll and print a T&T character, using the above abbreviations.

2. In *Dungeons & Dragons* (D&D), a character has six basic characteristics: strength (STR), constitution (CON), intelligence (INT), wisdom (WIS), dexterity (DEX), and charisma (CHA). Put the abbreviations in a DATA statement such as the one below.

```
1000 REM**CHARACTERISTIC ABBREVIATIONS
1010 DATA STR,CON,INT,WIS,DEX,CHA
```

Write a program to roll a D&D character. Read an abbreviation, roll dice, and print the result for each characteristic.

Answers

1. We did it the easy way, by screen editing the original program. Change only block 200 as follows.

```
200 REM**ROLL CHARACTERISTICS
910 GOSUB 910: PR. "STR", DICE
220 GOSUB 910: PR. "CON", DICE
230 GOSUB 910: PR. "IQ", DICE          Here is a change.
240 GOSUB 910: PR. "LK", DICE
250 GOSUB 910: PR. "DEX", DICE         Here is a change.
260 GOSUB 910: PR. "CHR", DICE         Here is a change.
270 STOP
```

2. Here are two ways. First way:

```
100 REM**CREATE A CHARACTER
110 DIM ABBR$(3)
120 GR. 0

200 REM**ROLL CHARACTERISTICS
210 READ ABBR$
220 GOSUB 910: PR. ABBR$, DICE
230 GOTO 210

900 REM**DICE SUBROUTINE
910 D1 = INT(6*RND(0))+1
920 D2 = INT(6*RND(0))+1
930 D3 = INT(6*RND(0))+1
940 DICE = D1 + D2 + D3
950 RETURN

1000 REM**CHARACTERISTIC ABBREVIATIONS
1010 DATA STR,CON,INT,WIS,DEX,CHA
```

When you run this program, it stops with an ERROR 6 message. That's OK, since all the information is on the screen.

Another way. Change only block 200.

```
200 ROLL CHARACTERISTICS
210 FOR K=1 TO 6
220   READ ABBR$
230   GOSUB 910: PR. ABBR$, DICE
240 NEXT K
250 STOP
```

SELF-TEST

Well, it looks like you have meandered into another Self-Test.

1. For each of the following, what are the *possible* values of X?

   (a) X = INT(3*RND(0)) ____________________
   (b) X = INT(3*RND(0)) + 1 ____________________
   (c) X = INT(3*RND(0)) − 1 ____________________
   (d) X = 2*INT(3*RND(0)) ____________________
   (e) X = 2*(INT(3*RND(0)) + 1 ____________________

2. Complete the following.

   (a) If N=100, then INT(N*RND(0)) will be a random integer in the range ____ to ____, inclusive.

   (b) If N=20, then INT(N*RND(0)) + 11 will be a random integer in the range ____ to ____, inclusive.

3. Complete the following so X will be a random number in the range shown.

   (a) X=10,11,12, or 13. X= ____________
   (b) X=2,5, or 8. X= ____________

4. Where will the wandering question mark wander if you run the following program? ____________________________________

```
100 REM**WANDERING CHARACTER
110 DIM CH$(1)
120 CH$ = "?"
130 GR. 0

200 REM**RANDOM SCREEN POSITION
210 COL = INT(20*RND(0))
220 ROW = INT(12*RND(0))

300 REM**PRINT CH$ IN RANDOM PLACE
310 PR. CHR$(125)
320 POS. COL,ROW: PR. CH$
```

(continued)

```
400 REM**DELAY & GO AGAIN
410 TD = 200
420 FOR Z=1 TO TD: NEXT Z
430 GOTO 210
```

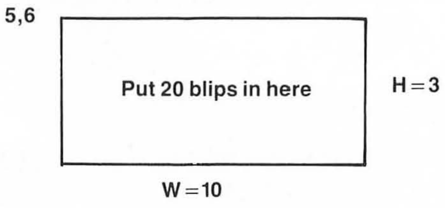
5. Write a program to make a "ball" roll left or right randomly in row 11. Start the ball in column 19. Then, each time through a loop, roll it one place left or one place right or leave it where it is. You can do this by adding − 1,0, or 1 to the present column position. Use RND to get − 1,0, or 1 randomly.

6. Write a program to make random sounds with random voice (V = 0,1,2, or 3), random tone number (N = 0 to 255), random distortion (D = 0 to 15), and random loudness (L = 0 to 15). Also try only even numbers, 0 to 14, for distortion. Include a time delay. Make the time delay easy to adjust.

7. Write a GRAPHICS 3,5 or 7 program to plot random blips of light in a rectangular block of the screen. You enter the location (COL,ROW) of the upper left corner of the block, the width (W) of the block, the height (H) of the block, and the number of blips to plot. For example, the text window might look like this.

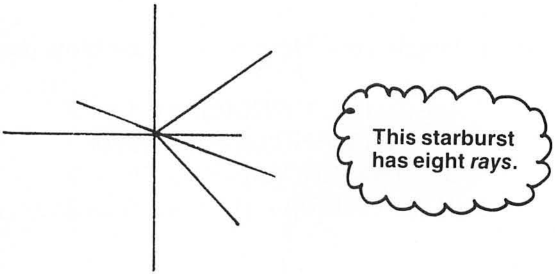
```
CORNER (COL,ROW)?5,6
WIDTH, HEIGHT?10,3
NUMBER OF BLIPS? 20■
```

If you now press RETURN, the computer will put 20 random color blips in the rectangular area of the screen from column 5 to column 15, row 6 to row 9.



After plotting the required number of blips, the computer asks (in the text window) for more information.

8. Write a GRAPHICS 3,5 or 7 program to put random "starbursts" on the screen. A starburst looks like this:



You enter the location (COL,ROW) of the center of starburst and the number of rays emanating from the center. The computer computes random end points for the rays. For example:

```
CENTER (COL,ROW)?10,10
NUMBER OF RAYS?8■
```

Now press RETURN and the computer will put a starburst with eight rays centered at column 10, row 10.

9. In the CREATE A CHARACTER program, we used a DICE SUBROUTINE to simulate (imitate) rolling three six-sided dice. Rewrite this dice rolling subroutine. Use a FOR-NEXT loop to roll three dice and compute their sum.

10. Write a DICE SUBROUTINE to roll ND dice, each with NS sides. For example, suppose you want to roll four dice, each with eight sides. Then ND is 4 and NS is 8. The possible rolls range from 4 to 32.

Answers to Self-Test

1. (a) 0,1, or 2
   (b) 1,2, or 3
   (c) − 1,0, or 1
   (d) 0,2, or 4
   (e) 2,4, or 6

2. (a) 0 to 99
   (b) 11 to 30

3. (a) X = INT(4*RND(0))+10
   (b) X = 3*(INT(3*RND(0))+1)−1

Did this one boggle you? Here is a blow by blow description.

INT(3*RND(0)) is 0,1, or 2.
INT(3*RND(0))+1 is 1,2, or 3.
3*(INT(3*RND(0))+1) is 3,6, or 9.
3*INT(3*RND(0))+1)−1 is 2,5, or 8.

4. The question mark will wander in the upper left quarter of the screen. The value of COL is a random integer from 0 to 19 and the value of ROW is a random integer from 0 to 11.

5.

```
100 REM**ROLL A BALL RANDOMLY
110 GR. 0
120 POKE 752,1

200 REM**START THING AT 19,11
210 COL = 19: ROW = 11
220 POS. COL,ROW: PR. "●"

300 REM**MOVE THING
310 D = INT(3*RND(0))-1
320 POS. COL,ROW: PR. " "
330 COL = COL + D
340 POS. COL,ROW: PR. "●"

400 REM**DELAY, THEN GO AROUND
410 TD= 20
420 FOR Z=1 TO TD: NEXT Z
430 GOTO 310
```

Use CTRL T for the ball

Try other values of TD.

Run this program and watch the ball roll indecisively right or left. If it tries to roll off the screen (COL less than zero or more than 39), the computer will stop with an ERROR 3 or ERROR 141 message.

6.
```
100 REM**VERY RANDOM SOUND
110 GR. 0

200 REM**MAKE EVERYTHING RANDOM
210 V = INT(4*RND(0))
220 N = INT(256*RND(0))
230 D = INT(16*RND(0))
240 L = INT(16*RND(0))

300 REM**NOW MAKE A SOUND
310 SO. V,N,D,L

400 REM**DELAY & GO AGAIN
410 TD = 1
420 FOR Z=1 TO TD; NEXT Z
430 GOTO 210
```

Also try: 230 D = 2*INT(8*RND(0))

7. We arbitrarily picked GRAPHICS 5. The program will also work in GRAPHICS 3 or 7. Simply change line 110.

```
100 REM**RANDOM BLIPS
110 GR. 5

200 REM**GET INFORMATION
210 PR. CHR$(125)
220 PR. "CORNER (COL,ROW)";:IN.COL,ROW
230 PR. "WIDTH,HEIGHT";:IN.W,H
240 PR. "NUMBER OF BLIPS";:IN.NB

300 REM**PUT NB BLIPS ON SCREEN
310 FOR BLIP=1 TO NB
320   COLB = INT((W+1)*RND(0))+COL
330   ROWB = INT((H+1)*RND(0))+ROW
340   COLOR INT(3*RND(0))+1
350   PLOT COLB,ROWB
360 NEXT BLIP

400 REM**GO FOR MORE
410 GOTO 210
```

8,9,10. We leave these without answers for you to solve. Question 10 may make you wonder. Eight-sided dice? Yes, in Fantasy Role Playing games there are 4-sided, 6-sided, 8-sided, 10-sided, 12-sided and 20-sided dice. For information about dice, write to:

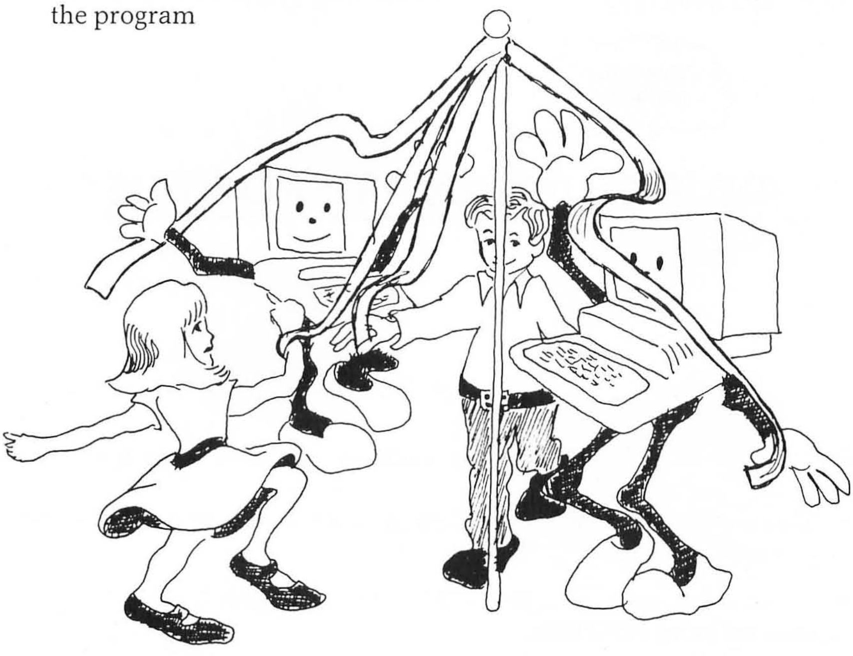
Zocchi Distributors
Middle Earth Hobbies
01956 Pass Road
Gulfport, MS 39501

Chapter Nine

Playtime Junction

Welcome, wayfarer, to Worlds of IF. In this chapter, you will learn how to teach your ATARI computer to make decisions and play games. When you finish this chapter, you will be able to:

- Use the IF-THEN statement to tell the computer to make a decision
- Create simple games using color, sound, shapes, letters, words and numbers
- Make Zappy Artist meander about the screen
- Race colors across the screen
- Use ON...GOTO and ON...GOSUB to select one of several paths through the program



The IF-THEN Statement

The IF-THEN statement tells the computer to make a very simple decision. It tells the computer to do a certain operation *if* a given *condition* is true. However, if the condition is *false* (not true), the operation will not be done. Here is an IF-THEN statement:

```
410 IF X>0 THEN PR. "POSITIVE"
```

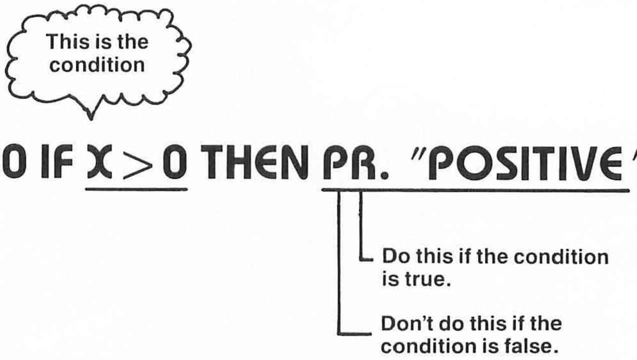
This IF-THEN statement tells the computer:

- If the value of X is greater than zero (IF X>0), then print the word POSITIVE.
- If the value of X is *not* greater than zero, *don't* print the word.

Here is another way to think about it:

- If the value of X is greater than zero, execute the statement following the word THEN.
- If the value of X is *not* greater than zero, *don't* execute the statement following THEN.

Hmmm...one more time.


This is the condition
410 IF X>0 THEN PR. "POSITIVE"
Do this if the condition is true.
Don't do this if the condition is false.

In the above IF-THEN statement, the condition is X>0 or X is greater than zero.

- Suppose the value of X is 3. Then the condition is *true* and the computer prints POSITIVE.
- Suppose the value of X is −7. Now the condition is *false*. The computer does *not* print POSITIVE.

- Suppose the value of X is 0. In this case, the condition is *false*. The word POSITIVE is *not* printed.

The following program has three IF-THEN statements to tell the computer "to do or not to do."

```
100 REM**NEGATIVE, ZERO, OR POSITIVE
110 GR. 0

200 REM**TELL WHAT TO DO
210 PR. "ENTER A NUMBER AND I'LL TELL"
220 PR. "YOU WHETHER YOUR NUMBER IS"
230 PR. "NEGATIVE, ZERO, OR POSITIVE."

300 REM**ASK FOR A NUMBER
310 PR.
320 PR. "YOUR NUMBER";: IN. X

400 REM**TELL ABOUT THE NUMBER
410 IF X<0 THEN PR. "NEGATIVE"
420 IF X=0 THEN PR. "ZERO"
430 IF X>0 THEN PR. "POSITIVE"

500 REM**GO BACK FOR ANOTHER NUMBER
510 GOTO 310
```

"I'm positive about warm weather and negative about cold weather."

A run might go this way:

```
ENTER A NUMBER AND I'LL TELL
YOU WHETHER YOUR NUMBER IS
NEGATIVE, ZERO, OR POSITIVE.

YOUR NUMBER?3
POSITIVE

YOUR NUMBER? - 7
NEGATIVE

YOUR NUMBER?0
ZERO

YOUR NUMBER? ■          Your turn.    Carry on!
```

If you enter a negative number (such as − 7), the condition in line 410 is true and the conditions in lines 420 and 430 are false. So the computer prints NEGATIVE.

If you enter zero (0), the condition in line 420 is true and the conditions in lines 410 and 430 are false. The computer prints ZERO.

If you enter a positive number (such as 3), the condition in line 430 is true and the conditions in lines 410 and 420 are false. The computer prints POSITIVE.

In general, the IF-THEN statement has the following form:

IF condition THEN statement

The *statement* can be almost any ATARI BASIC statement. The *condition* is usually a comparison between a variable and a value, between two variables, or between two more complicated expressions. Here is a handy table showing ATARI BASIC symbols and math symbols:

Math Symbol	Comparison	ATARI BASIC Symbol
=	is equal to	=
<	is less than	<
>	is greater than	>
≤	is less than or equal to	< =
≥	is greater than or equal to	> =
≠	is not equal to	< >

Questions

1. In each IF-THEN statement, underline the condition.

(a) IF ROW <0 THEN ROW=0
(b) IF COL=39 THEN GOTO 610
(c) IF X> =0 THEN PR. "NON-NEGATIVE"
(d) IF G< >X THEN PR. "HA! THAT'S NOT IT."

2. Write a program to ask for a value of COL. If the value is less than 0 or greater than 39, print OUT OF BOUNDS!, then go back for another value.

Answers

1. (a) IF *ROW<0* THEN ROW=0
(b) IF *COL=39* THEN GOTO 610
(c) IF *X> =0* THEN PR. "NON-NEGATIVE"
(d) IF *G< >X* THEN PR "HA! THAT'S NOT IT."

2.

```
100 REM**OUT OF BOUNDS
110 GR. 0

200 REM**ASK FOR VALUE OF COL
210 PR. "WHAT IS COL";: IN. COL

300 REM**TELL ABOUT VALUE
310 IF COL<0 THEN PR. "OUT OF BOUNDS!"
320 IF COL>39 THEN PR. "OUT OF BOUNDS!"

400 REM**GO FOR ANOTHER VALUE
410 PR.
420 GOTO 210
```

Lines 310 and 320 can be combined as follows:

```
310 IF COL<0 OR COL>39 THEN PR. "OUT OF BOUNDS!"
```

The condition

The two simple conditions have been combined into a *compound condition*. If you try this variation, remember to delete line 320.

GAME TIME!

Here is our first computer game, a simple number-guessing game:

```
100 REM**FIRST NUMBER GUESSING GAME
110 GR. 0

200 REM**TELL HOW TO PLAY
210 PR. "I WILL THINK OF A NUMBER."
220 PR. "MY NUMBER IS 1 OR 2."

300 REM**X IS SECRET NUMBER
310 X = INT(2*RND(0))+1

400 REM**G IS PLAYER'S GUESS
410 PR.
420 PR. "OK, I HAVE A NEW NUMBER."
430 PR. "WHAT IS YOUR GUESS (1 OR 2)";: IN. G

500 REM**TELL IF RIGHT OR WRONG
510 IF G=X THEN PR."THAT'S IT. YOU MUST HAVE ESP!"
520 IF G<>X THEN PR. "HA! THAT'S NOT IT."

600 REM**GO PLAY AGAIN
610 PR. "LET'S PLAY AGAIN."
620 GOTO 310
```

Enter the game and play for a short time. Well, it's not the world's most exciting computer game, but it's a beginning. Better things are coming!

A condition can also be a relationship between two strings or two string variables, as shown in lines 510 and 520 of the following program. In this game, the computer "flips" a coin. You guess H for heads or T for tails.

```
100 REM**COIN FLIP GAME
110 GR. 0
120 DIM X$(1),G$(1)
```

(continued)

```
200 REM**TELL HOW TO PLAY
210 PR. "I WILL FLIP A COIN. GUESS"
220 PR. "H FOR HEADS OR T FOR TAILS."

300 REM**COMPUTER "FLIPS" COIN, X$
310 X = INT(2*RND(0))
320 IF X=0 THEN X$="H"
330 IF X=1 THEN X$="T"

400 REM**G$ IS PLAYER'S GUESS
410 PR.
420 PR. "OK, I FLIPPED IT."
430 PR. "WHAT IS YOUR GUESS (H OR T)";: IN. G$

500 REM**TELL IF RIGHT OR WRONG
510 IF G$=X$ THEN PR."THAT'S IT.YOU MUST HAVE ESP!"
520 IF G$<>X$ THEN PR. "HA! THAT'S NOT IT."

600 REM**GO PLAY AGAIN
610 PR. "LET'S PLAY AGAIN."
620 GOTO 310
```

Enter the program and play the game. The computer simulates flipping a fair coin. It will give heads (H) or tails (T) with about equal probability, just as in flipping a real coin.

Questions

1. Answer these questions about the FIRST NUMBER GUESSING GAME:

   (a) In line 310, what are the possible values of X?________

   (b) What variable holds the player's guess?________

   (c) Which lines compare the guess with the computer's secret number?

   ____

   (d) What is the condition in line 510?________

   (e) Suppose this condition is *true*. What happens?

   ____

   (f) Suppose the condition is *false*? What happens?

   ____

   (g) What is the condition in line 520?________

   (h) Suppose this condition is true. What happens?

   ____

   (i)Suppose the condition is false. What happens?

   ____

2. Answer these questions about the COIN FLIP GAME.

   (a) In line 310, what are the possible values of X?

   ____

   (b) Suppose X is 0. What will be the value of X$?

   ____

   (c) Suppose X is 1. What will be the value of X$?

   ____

   (d) What variable holds the player's guess?________

   (e) What is the condition in line 510?________

   (f) What is the condition in line 520?________

   (g) Got it?________

Answers

1. (a) 1 or 2.

   (b) G (used in lines 430, 510, and 520).

   (c) Lines 510 and 520.

   (d) G=X (G is equal to X).

   (e) The computer prints THAT'S IT. YOU MUST HAVE ESP!

   (f) Nothing. The computer moves on to line 520.

   (g) G< >X (G is not equal to X).

   (h) The computer prints HA! THAT'S NOT IT.

   (i) Nothing. The computer moves on to line 610.

2. (a) 0 or 1.

   (b) "H" This happens in line 320 whenever X=0 is true.

   (c) "T" This happens in line 330 whenever X=1 is true.

   (d) G$ You see it in lines 120, 430, 510, and 520.

   (e) G$=X$ or G$ is equal to X$.

   (f) G$< >X$ or G$ is not equal to X$.

   (g) We hope you said yes. If not, compare this program with the FIRST NUMBER GUESSING GAME program. They are quite similar. Differences include the use of (in the COIN FLIP GAME) string variables to hold the computer's coin flip and the player's guess. The conditions in lines 510 and 520 compare the string values of variables G$ and X$.

When you play the COIN FLIP GAME, you will win about half the time and lose about half the time. Play it a bunch of times and keep track. What? You played it 100 times and won 73 times? Well, maybe you *do* have ESP!

Have some fun with a friend. Make these changes to the COIN FLIP GAME:

```
100 REM**UNFAIR COIN FLIP GAME

300 REM**FLIP AN UNFAIR COIN
310 X = INT(3*RND(0))
320 IF X<=1 THEN X$="H"
330 IF X=2  THEN X$="T"
```

Now the computer will "flip" heads (H) about twice as often as tails (T). Have your friend keep track of wins and losses. Perhaps she or he will discover that the computer is flipping an unfair coin. Here is another unfair variation.

```
300 REM**COMPUTER "FLIPS" A COIN, X$
310 PH = 60
320 X = INT(100*RND(0))+1
330 IF X<=PH THEN X$="H"
340 IF X>PH  THEN X$="T"
```

Think of PH as the probability of getting heads. We have set it to 60 to mean a 60% chance of getting heads. In line 320, the possible values of x are 1 to 100, inclusive. So, in line 330, the condition will be true about 60% of the time. If you want heads to appear 30% of the time, change line 310 to: 310 PH = 30. Or select your own value of PH.

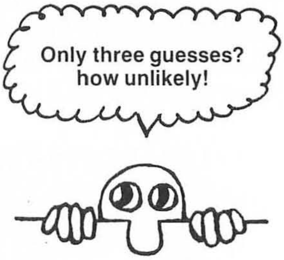
GUESSING WITH SOUND

Next, a game of GUESS MY TONE. The computer computes a random tone number, 1 to 255, then plays the tone. You guess the tone number. A run might go like this.

```
I'LL PLAY A TONE. GUESS MY TONE NUMBER.

MY LOWEST TONE HAS TONE NUMBER 255.
MY HIGHEST TONE HAS TONE NUMBER 1.

GUESS THIS TONE
WHAT IS YOUR GUESS?100
HERE IS YOUR TONE

GUESS THIS TONE
WHAT IS YOUR GUESS?50
HERE IS YOUR TONE

GUESS THIS TONE
WHAT IS YOUR GUESS?73
THAT'S IT. YOU GUESSED MY TONE!
```

Computer plays these tones.

Computer sounds mystery tone before each guess and player's tone after each guess.



After a short time delay, the game begins again with a new tone. Now we will show you the program in a few easy pieces. The first piece is especially easy. Nothing new here.

```
100 REM**GUESS MY TONE
110 GR. 0

200 REM**TELL HOW TO PLAY
210 PR. CHR$(125)
220 PR. "GUESS MY TONE NUMBER."
230 PR.
240 PR. "MY LOWEST TONE HAS TONE NUMBER 255."
250 PR. "MY HIGHEST TONE HAS TONE NUMBER 1."

300 REM**COMPUTE SECRET TONE NUMBER
310 N = INT(255*RND(0))+1
```

Next, we want the computer to play the secret tone.

```
400 REM**PLAY SECRET TONE
410 PR.: PR. "GUESS THIS TONE"
420 SO. 0,N,10,10
430 TD = 500: GOSUB 910
```

As you will soon see, block 900 is a time delay subroutine which also shuts off the sound. Having heard the secret sound, the player is asked to enter a guess.

```
500 REM**GET GUESS & COMPARE
510 PR. "YOUR GUESS";: IN. G
520 IF G=N THEN GOTO 710
```

Line 520 compares the guess (G) with the secret tone number (N). If they are equal, the computer goes to line 710.

```
   Instead of: 520 IF G=N THEN GOTO 710
You can write: 520 IF G=N THEN 710
```

If the guess is not correct, on to block 600. It plays the tone corresponding to the player's guess, then goes back for a new guess.

```
600 REM**GUESS NOT CORRECT
610 PR. "HERE IS YOUR TONE"
620 SO. 0,G,10,10
630 TD = 500: GOSUB 910
640 GOTO 410
```

Suppose, back in line 520, the computer discovered that the guess was correct (G = N was true). In that case, here we are at block 700.

```
700 REM**WINNER!
710 PR. "THAT'S IT! YOU GUESSED MY TONE."
720 SO. 0,N,10,10
730 TD = 2000: GOSUB 910
740 GOTO 210
```

You hear the secret tone for a few seconds, then the game starts anew.

One more block, the time delay subroutine. Please note that it also shuts off the sound for voice zero.

```
900 REM**TIME DELAY SUBROUTINE
910 FOR Z=1 TO TD: NEXT Z
920 SO. 0,0,0,0
930 RETURN
```

Here is a variation to try. Make these changes and additions:

```
   Add:  120 HT = 100
         130 LT = 200
Change:  240 PR. "MY LOWEST TONE HAS TONE NUMBER "; LT
         250 PR. "MY HIGHEST TONE HAS TONE NUMBER "; HT
         310 N = INT((LT-HT+1)*RND(0)) + HT
```

Now the tone number (N) will be in the range, 100 to 200, inclusive. If you want another range, choose your own values of High Tone (HT) and Low Tone (LT). Remember: low tones have high numbers and high tones have low numbers. Tone numbers will be in the range HT or LT, inclusive. For our numbers above:

(1) HT=100 and LT=200

(2) LT−HT+1 = 101

(3) INT((LT−HT+1)*RND(0)) is 0 to 100.

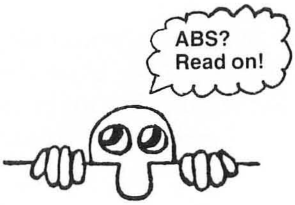
(4) INT((LT−HT+1)*RND(0)) + HT is 100 to 200.

If you choose values of HT and LT, remember to make HT smaller then LT. Yes, we too wonder why ATARI made tone numbers backwards!

In our next game, we use sound as a clue to help you guess a number. The computer "thinks" of a number. You guess, then listen. The computer plays a tone. A high tone means you are close; the closer you are, the higher the tone. A low tone means you are far away.

```
100 REM**LISTEN, AND GUESS MY NUMBER
110 GR. 0
120 LO = 1
130 HI = 255

200 REM**TELL HOW TO PLAY
210 PR. CHR$(125)
220 PR. "I'LL THINK OF A NUMBER"
230 PR. "FROM "; LO; " TO "; HI
240 PR.
250 PR. "GUESS MY NUMBER, THEN LISTEN."
260 PR. "HIGH TONE MEANS YOU ARE CLOSE."
270 PR. "LOW TONE MEANS YOU ARE FAR AWAY."

300 REM**COMPUTER "THINKS" OF A NUMBER
310 X = INT(HI-LO+1)*RND(0)) + LO

400 REM**GET A GUESS
410 PR.
420 PR. "YOUR GUESS";: IN. G
```

(continued)



```
500 REM**D IS DISTANCE FROM X
510 D = ABS(X-G)

600 REM**CHECK FOR A WIN
610 IF D=0 THEN 810

700 REM**NO WIN. SOUND HINT & GO AROUND
710 IF D>255 THEN D=255
720 SO. 0,D,10,10
730 TD = 500
740 FOR Z=1 TO TD: NEXT Z
750 GOTO 410

800 REM**WINNER!
810 PR. "CONGRATULATIONS! YOU GOT IT."

900 REM**TIME DELAY, THEN PLAY AGAIN
910 TD = 2000
920 FOR Z=1 TO TD: NEXT Z
930 GOTO 210
```

In line 510, we use the ABS function. ABS means "ABSolute value." In this program, we use ABS to compute the *distance* from the guess (G) to the computer's secret number (X). Here are some examples:

X	G	X−G	ABS(X−G)	
73	30	43	43	
73	100	−27	27	
73	73	0	0	Winner!

Questions

1. In the variation to GUESS MY TONE, suppose we set the high tone and low tone numbers as follows:

```
120 HT =  60    (One octave above Middle C)
130 LT = 121     (Middle C)
```

Complete the following.

(a) LT−HT+1 = _______.
(b) INT((LT−HT+1)*RND(0)) is ____ to ____.
(c) INT((LT−HT+1)*RND(0))+HT is ____ to ____.

2. Answer these questions about LISTEN, AND GUESS MY NUMBER.

(a) As the game is written, the computer's secret number (X) is in the range ____ to ____.
(b) Suppose X is 120 and G is 100. What is D? ____
(c) Suppose X is 97 and G is 150. What is D? ____
(d) What is the purpose of line 710?

Answers

1. (a) LT−HT+1 = 121−60+1 = 62

(b) INT((LT−HT+1)*RND(0)) is 0 to 61.

(c) INT((LT−HT+1)*RND(0))+HT is 60 to 121.

2. (a) 1 to 255. These are the values of LO and HI which are assigned in lines 120 and 130.

   (b) D = ABS(X−G) = ABS(120−100) = ABS(20) = 20.

   (c) D = ABS(X−G) = ABS(97−150) = ABS(−53) = 53.

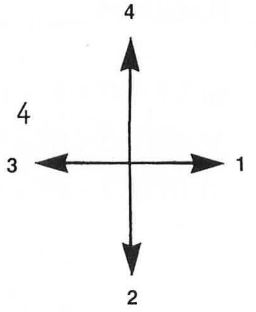
   (d) Suppose X is 100 and someone guesses 357. Then D is 257, larger than 255. This guess is far away and someone should hear a low tone. If we used 257 as the tone number in line 720, you would hear a very high note — 257 produces the same tone as 1. So line 710 checks for this and sets the tone to the lowest possible tone with tone number 255.

   Oops! Some people might not be able to hear the tones for tone number 1, 2 or even 3. You might want to add a line to set the smallest tone number used to 2 or 3.

ZAPPY ARTIST MEANDERS

Zappy Artist meanders right, left, up or down on the screen.

```
100 REM**ZAPPY ARTIST MEANDERS
110 GR. 19

300 REM**ZAPPY ARTIST APPEARS
310 COL = 20
320 ROW = 12
330 COLOR INT(3*RND(0))+1
340 PLOT COL,ROW

500 REM**WHICH WAY? CHOOSE 1 OF 4
510 WAY = INT(4*RND(0))+1
520 IF WAY=1 THEN COL=COL+1
530 IF WAY=2 THEN ROW=ROW+1
540 IF WAY=3 THEN COL=COL-1
550 IF WAY=4 THEN ROW=ROW-1
```



```
700 REM**ZAPPY ARTIST GOES THAT WAY
710 COLOR INT(3*RND(0))+1
720 PLOT COL,ROW
```

(continued)

```
900 REM**DELAY, THEN MEANDER MORE
910 TD = 100
920 FOR Z=1 TO TD: NEXT TD
930 GOTO 510
```

Zappy Artist first appears (block 300) near the center of the GRAPHICS 19 screen at col 20, row 12. He or she then meanders right (WAY=1) or down (WAY=2) or left (WAY=3) or up (WAY=4).

Eventually, Zappy Artist will wander off the screen and you will see: ERROR 3 AT LINE 720. This happens if the value of COL becomes less then zero or more than 39. It also happens if the value of ROW becomes less then zero or more than 23. Add block 600 to prevent this from happening.

```
600 REM**KEEP ZAPPY ON SCREEN
610 IF COL>39 THEN COL=39
620 IF ROW>23 THEN ROW=23
630 IF COL<0  THEN COL=0
640 IF ROW<0  THEN ROW=0
```

Questions

1. Change the ZAPPY ARTIST MEANDERS program so it runs in (a) GRAPHICS 21, (b) GRAPHICS 23.
2. Add block 200 to set the background white and color registers 0, 1, and 2 to colors of your choice.

Answers

1. Remember, we want Zappy Artist to start near the center of the screen. Here are the *changes* for (a) GR. 21 and (b) GR. 23.

```
(a) 110 GR. 21
    310 COL = 40
    320 ROW = 24
    610 IF COL>79 THEN COL=79
    620 IF ROW>47 THEN ROW=47
    910 TD = 1
```

```
(b) 110 GR. 23
    310 COL = 80
    320 ROW = 48
    610 IF COL>159 THEN COL=159
    620 IF ROW>95  THEN ROW=95
    910 TD = 1
```

2.
```
200 REM**CHOOSE COLORS
210 SE. 4,0,14          white background
230 SE. 1,4,8           pink
240 SE. 2,5,8           purple
```

Now add some eerie sound to the GRAPHICS 23 program.

```
800 REM**STRANGE SOUNDS
810 SO. 0,COL,10,10
820 SO. 1,ROW,10,10
```

Also try it with the GR. 19 and GR. 21 programs.

RACING COLORS

In GRAPHICS 3,5 and 7, you can use three different colors at any one time. Let's pick two colors and race them across the screen. We will race orange and light green in GRAPHICS 3.

First, start the two colors at the left edge of the screen in rows 5 and 10:

```
100 REM**RACING COLORS
110 GR. 3

300 REM**RACERS TO STARTING POSITION
310 COLOR 1
320 COL1 = 0: PLOT COL1,5
330 COLOR 2
340 COL2 = 0: PLOT COL2,10
```

The colors are on the screen. In the text window you see:

```
400 REM**READY, SET, GO!
410 PR. CHR$(125)
420 PR. "GET READY"
430 TD = 500: GOSUB 1010
440 PR. "GET SET"
450 TD = 500: GOSUB 1010
460 PR. "GO!"
```

They're off and racing. How will we do that? We'll do it like this: Each time around, we will advance each color one or two spaces. The second space depends on a random number. For each color, we assign a probability from 0 to 100 of advancing the second space.

```
500 REM**CHANCE OF 2ND SPACE
510 P1 = 60
52C P2 = 40
```

Color number one (orange) has a better chance (60%) of moving the second space. Color number two (light green) has only a 40% chance. You can think of color number one as being "faster" then color number two. Of course, you can change these numbers to suit yourself.

Next we compute the actual number of spaces each color moves.

```
600 REM**COMPUTE HOW FAR TO MOVE
610 COL1 = COL1 + 1
620 X = INT(100*RND(0))+1
630 IF X<=P1 THEN COL1=COL1+1
640 COL2 = COL2 + 1
650 X = INT(100*RND(0))+1
660 IF X<=P2 THEN COL2=COL2+1
```

Lines 610 and 640 move each color one space. Lines 620 and 630 give a P1 per cent chance of color one moving another space. Lines 650 and 660 give a P2 per cent chance of color two moving another space. Now paint the colors from column 0 to their new positions.

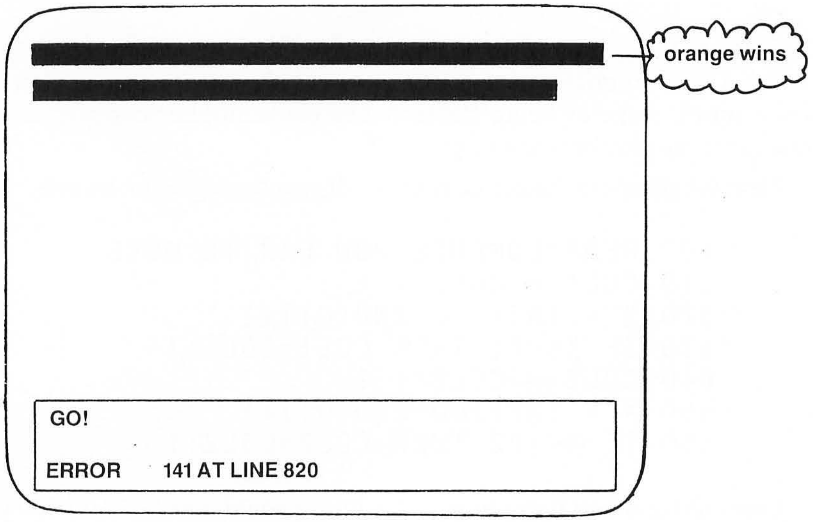
```
800 REM**PAINT COLCR TO NEW POSITIONS
810 COLOR 1
820 PLOT 0,5: DRAWTO COL1,5
830 COLOR 2
840 PLOT 0,10: DRAWTO COL2,10
```

Keep em racing. After a short time delay, go back for another move:

```
900 REM**DELAY, THEN GO AROUND
910 TD = 100: GOSUB 1010
920 GOTO 610

1010 REM**TIME DELAY SUBROUTINE
1020 FOR Z=1 TO TD: NEXT Z
1030 RETURN
```

OK, run the program and watch them race. Orange will usually win, unless you change the values of P1 and P2 in lines 510 and 520. Every race will end with one of these messages in the text window:

```
       ERROR 141 AT LINE 820
    or ERROR 141 AT LINE 840
```



```
GO!

ERROR   141 AT LINE 820
```

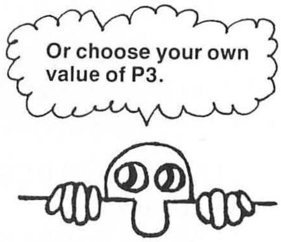
Questions

1. Add a third racing color, dark blue in row 15.
2. A challenge: Add block 700 to prevent any color from trying to go off the screen. Then change block 900 so the computer announces the winner or, if there is a tie, says so. You might also want to add sound to the program.

Answers

1. Add these lines.

```
350 COLOR 3
360 COL3 = 0: PLOT COL3,15

530 P3 = 50

670 COL3 = COL3 + 1
680 X = INT(100*RND(0))
690 IF X<=P3 THEN COL3=COL3+1

850 COLOR 3
860 PLOT 0,15: DRAWTO COL3,15
```



2. We'll leave this one for you. We also suggest you modify the program for GRAPHICS 5 or 7. May the best color win!

A TREE HAS MANY BRANCHES

This chapter began with a program called NEGATIVE, ZERO, OR POSITIVE. Here it is again with some changes:

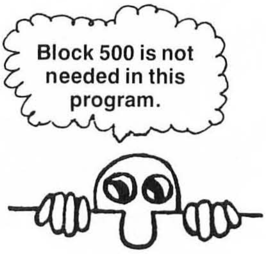
```
100 REM**NEGATIVE, ZERO, OR POSITIVE
110 GR. 0
120 DIM N$(8) ,Z$(4) ,P$(80
130 N$ = "NEGATIVE"
140 Z$ = "ZERO"
150 P$ = "POSITIVE"

200 REM**TELL WHAT TO DO
210 PR. "ENTER A NUMBER AND I'LL TELL"
220 PR. "YOU WHETHER YOUR NUMBER IS"
230 PR. "NEGATIVE, ZERO, OR POSITIVE."

300 REM**ASK FOR A NUMBER
310 PR.
320 PR. "YOUR NUMBER";: IN. X
```

(continued)

```
400 REM**TELL ABOUT THE NUMBER
410 IF X<0 THEN PR. N$
420 IF X=0 THEN PR. Z$
430 IF X>0 THEN PR. P$

500 REM**GO BACK FOR ANOTHER NUMBER
510 GOTO 310
```

One of the nicest things about computers: There is always another way to solve a problem, write a program. Here is another way to write the above program:

```
100 REM**NEGATIVE, ZERO, OR POSITIVE
110 GR. 0
120 DIM N$(8),Z$(4),P$(8)
130 N$ = "NEGATIVE"
140 Z$ = "ZERO"
150 P$ = "POSITIVE"

200 REM**TELL WHAT TO DO
210 PR. "ENTER A NUMBER AND I'LL TELL"
220 PR. "YOU WHETHER YOUR NUMBER IS"
230 PR. "NEGATIVE, ZERO, OR POSITIVE."

300 REM**ASK FOR A NUMBER
310 PR.
320 PR. "YOUR NUMBER";: IN. X

400 REM**TELL ABOUT THE NUMBER
410 W = SGN(X) + 2
420 ON W GOTO 430,440,450
430 PR. N$: GOTO 310
440 PR. Z$: GOTO 310
450 PR. P$: GOTO 310
```



OK! OK! We'll explain lines 410 and 420. In line 410, the SGN function has three possible values: −1, 0, or 1.

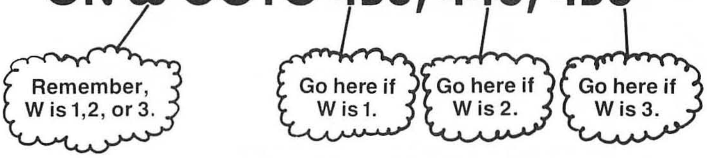
If X is negative, SGN(X) is −1.
If X is zero, SGN(X) is 0.
If X is positive, SGN(X) is 1.

The value of W is SGN(X) + 2.

```
If X is negative, W is 1.
If X is zero,     W is 2.
If X is positive, W is 3.
```

Line 420 sends the computer to line 430, line 440 or 450 depending on the value of W.

ON W GOTO 430, 440, 450



If W is 1, the computer goes to line 430, prints the value of N$, then goes to line 310. If W is 2, the computer goes to line 440, prints the value of Z$, then goes to line 310. And so on.

Lines 410 and 420 can be combined into a single line, as follows:

```
410 ON SGN(X) + 2 GOTO 430,440,450
```

If you do this, remember to delete the old line 420.

There is a cousin to ON...GO called ON...GOSUB. Rewrite block 400 this way:

```
400 REM**TELL ABOUT THE NUMBER
410 ON SGN(X)+2 GOSUB 430,440,450
420 GOTO 310
430 PR. N$: RETURN
440 PR. Z$: RETURN
450 PR. P$: RETURN
```

It works this way:

If X is negative, SGN(X)+2 is equal to 1. The computer GOSUBs to line 430, prints the value of N$, and RETURNs to line 420.

If X is zero, SGN(X)+2 is equal to 2. The computer GOSUBs to line 440, prints the value of Z$, and RETURNs to line 420.

If X is positive, SGN(X)+2 is equal to 3. The computer GOSUBs to line 450, prints the value of P$, and RETURNs to line 420.

You will see even more ways to solve this problem in Chapters 10 and 11! In the meantime, here is another Zappy Artist escapade. Zappy draws horizontal and vertical stripes.

```
100 REM**ZAPPY ARTIST DRAWS STRIPES
110 GR. 19

200 REM**RANDOM STRIPE LOOP
210 HV = INT(2*RND(0))+1
220 ON HV GOSUB 310,410
230 TD = 100
240 FOR Z=1 TO TD: NEXT Z
250 GOTO 210

300 REM**HORIZONTAL STRIPE
310 ROW = INT(24*RND(0))
320 COL1 = INT(40*RND(0))
330 COL2 = INT(40*RND(0))
340 COLOR INT(3*RND(0))+1
350 PLOT COL1,ROW: DRAWTO COL2,ROW
360 RETURN

400 REM**VERTICAL STRIPE
410 COL = INT(40*RND(0))
420 ROW1 = INT(24*RND(0))
430 ROW2 = INT(24*RND(0))
440 COLOR INT(3*RND(0))+1
450 PLOT COL,ROW1: DRAWTO COL,ROW2
460 RETURN
```

Zappy will draw about the same number of horizontal stripes as vertical stripes. To change the mix, change lines 210 and 220.

- Twice as many horizontal as vertical stripes.

```
210 HV = INT(3*RND(0))+1
220 ON HV GOSUB 310,310,410
```

- Three times as many horizontal as vertical stripes.

```
210 HV = INT(4*RND(0))+1
220 ON HV GOSUB 310,310,310,410
```

Questions

1. Complete the following.

   (a) SGN(5) = ____.
   (b) SGN(− 7) = ____.
   (c) SGN(0) = ____.
   (d) SGN(− 7) + 2 = ____.
   (e) If X is 5, then SGN(X) + 2 is ____.

2. Write lines 210 and 220 so Zappy Artist draws four times as many vertical stripes as horizontal stripes.

   210 ______________________________
   220 ______________________________

Answers

1. (a) 1 (b) − 1 (c) 0 (d) 1 (e) 3

2.
```
210 HV = INT(5*RND(0))+1
220 ON HV GOSUB 310,410,410,410,410
```

   In line 210, the possible values of HV are 1, 2, 3, 4 and 5. If HV is 1, the computer GOSUBs to line 310. If HV is 2, 3, 4 or 5, the computer GOSUBs to line 410.

EXPERIMENT! Use this program to find out more about ON...GOTO:

```
100 REM**ON...GOTO EXPERIMENT
110 GR. 0

200 REM**ASK FOR A NUMBER
210 PR.
220 PR. "NUMBER, PLEASE";: IN. X

300 REM**ON...GO REPLIES
310 ON X GOTO 330,340,350
320 PR. "NOT IN MY RANGE": GOTO 210
330 PR. "ONE": GOTO 210
340 PR. "TWO": GOTO 210
350 PR. "THREE": GOTO 210
```

Try these values of X: − 1, 0, .5, 1, 1.5, 2, 2.4, 3, 3.7, 4. Write a similar program to experiment with ON...GOSUB.

SELF-TEST

IF your mind is fuzzy from cogitating on this chapter, THEN take a break — sing, dance, play conga drums, listen to music — then ON...GOTO this Self-Test:

1. In each IF-THEN statement, underline the condition.

   (a) IF COL<39 THEN COL=COL+1

   (b) IF N>255 THEN N=255

   (c) IF X=0 THEN PR. "ZERO"

   (d) IF G<X THEN PR. "TRY A BIGGER NUMBER": GOTO 210

   (e) IF G=X THEN 610

2. Complete each sentence with *true* or *false*.

   (a) If X is 73 and G is 50, then G<X is ____.

   (b) If X is 73 and G is 80, then G<X is ____.

   (c) If X is 73 and G is 80, then G=X is ____.

   (d) If X is 73 and G is 80, then G>X is ____.

   (e) If X is 73 and G is 50, then G>X is ____.

3. For each verbal statement, write a corresponding BASIC IF-THEN statement:

   (a) If the value of COL is less then zero, print OUT OF BOUNDS and go to line 310.______________

   (b) If D is equal to 1, print seven stars (*).________

   (c) If K$ is not equal to "S", then go to line 420. ______________

4. Here is a guessing game called STARS. The program is not complete — blocks 200, 500, 600, 800 and 900 remain for you to write:

```
100 REM**STARS - A GUESSING GAME
110 GR. 0
120 DIM AN$(10)

200 REM**TELL HOW TO PLAY
```

(continued)

```
300 REM**COMPUTER 'THINKS' OF NUMBER
310 X = INT(100*RND(0))+1

400 REM**GET GUESS
410 PR.
420 PR. "YOUR GUESS";: IN. G

500 REM**D IS DINSTANCE FROM X

600 REM**CHECK FOR A WIN

700 REM**NO WIN. PRINT HINT.
710 IF D>=64 THEN PR."*": GOTO 410
720 IF D>=32 THEN PR."**": GOTO 410
730 IF D>=16 THEN PR."***": GOTO 410
740 IF D>=8  THEN PR."****": GOTO 410
750 IF D>=4  THEN PR."*****": GOTO 410
760 IF D>=2  THEN PR."******": GOTO 410
770 PR. "*******": GOTO 410

800 REM**WINNER!

900 REM**PLAY AGAIN?
```

In block 900 the computer should ask PLAY AGAIN (Y OR N)? If someone types Y and presses RETURN, start the game again. If the answer is N, stop. If the answer is neither Y nor N, ask again.

5. Write a program to play random music selected from the following tone numbers. These are the tone numbers for Middle C, D, E, F, G, A, B and C.

Tone numbers	1	2	3	4	5	6	7	8
	C	D	E	F	G	A	B	C
	121	108	96	91	81	72	64	60

6. Write a program to play random chords using tone numbers selected at random as in question 5. Each chord consists of three tones played simultaneously by voices 0, 1 and 2.

7. Write a program to play random music. Use the eight tone numbers shown in question 5. Start at any of these and play it. For example, suppose you start at the 7th tone (64). Play it. Then add − 1, 0 or 1, at random. Suppose you add − 1 and get 6. Play tone number 72, the 6th tone. Add − 1, 0 or 1 and play the corresponding tone. And so on. However, the computer might compute 0 or 9 as the next to play. Instead of 0, play the 8th tone (60). Instead of 9, play the 1st tone (121).

8. Rewrite the program ZAPPY ARTIST MEANDERS without using IF-THEN statements. Use ON...GOTO or ON...GOSUB instead.

Answers to Self-Test

1. (a) IF COL<39 THEN COL=COL+1

   (b) IF N>255 THEN N=255

   (c) IF X=0 THEN PR. "ZERO"

   (d) IF G<X THEN PR. "TRY A BIGGER NUMBER": GOTO 210

   (e) IF G=X THEN 610

2. (a) true (b) false (c) false (d) true (e) false

   (b) IF D=1 THEN PR. "*******"

   (c) IF K$< > "S" THEN GOTO 420

   IF K$< > "S"THEN 420

4. We did it this way: Have fun playing.

```
200 REM**TELL HOW TO PLAY
210 PR. CHR$(125)
220 PR. "WELCOME TO MY GALAXY. I'LL THINK OF"
230 PR. "A NUMBER, 1 to 100. YOU GUESS MY"
240 PR. "NUMBER. IF YOU MISS, I'LL PRINT SOME"
250 PR. "STARS. THE CLOSER YOU ARE, THE MORE"
260 PR. "STARS YOU WILL SEE. IF YOU SEE SEVEN"
270 PR. "STARS (*******), YOU ARE VERY CLOSE!"

500 REM**D IS DISTANCE FROM X
510 D = ABS(X-G)
```

(continued)

```
600 REM**CHECK FOR A WIN
610 IF D=0 THEN 810

800 REM**WINNER!
810 FOR K=1 TO 100
820   COL = INT(40*RND(0))
830   ROW = INT(22*RND(0))
840   POS. COL,ROW: PR. "*"
850 NEXT K
860 POS.0,22: PR."YOU GOT IT.MY NUMBER IS "; X

900 REM**PLAY AGAIN?
910 POS.0,23: PR."PLAY AGAIN (Y OR N)";:IN.AN$
920 IF AN$="Y" THEN 210
930 IF AN$="N" THEN STOP
940 GOTO 910
```

5. You can use IF-THEN or ON...GOTO or ON...GOSUB. We used ON...GOSUB.

```
100 REM**RANDOM MUSIC, KEY OF C
110 GR. 0

200 REM**SELECT ONE OF EIGHT NOTES
210 N = INT(8*RND(0))+1
220 ON N GOSUB 410,420,430,440,450,460,470,480

300 REM**PLAY THE TONE & GO AROUND
310 SO. 0,TN,10,10
320 TD = 100
330 FOR Z=1 TO TD; NEXT Z
340 SO. 0,0,0,0
350 GOTO 210

400 REM**TONE NUMBER SUBROUTINE
410 TN = 121: RETURN
420 TN = 108: RETURN
430 TN = 96 : RETURN
440 TN = 91 : RETURN
450 TN = 81 : RETURN
460 TN = 72 : RETURN
470 TN = 64 : RETURN
480 TN = 60 : RETURN
```

6,7,8. We leave these as challenges to you.

Chapter
Ten

String Magic

You have already used strings and string variables in simple ways. Now you will learn how to use several *string functions* to do interesting and fun things with strings.

When you finish this chapter, you will be able to:

- Compare two strings
- Recognize and use ATASCII codes for characters
- Join two strings
- Convert strings to numbers
- Convert numbers to strings
- Find and use substrings (strings within strings)
- Use the functions: ASC, CHR$, LEN and VAL
- OPEN a channel to the keyboard and GET information from any key you press (you don't have to press RETURN)

SOMETHING OLD, SOMETHING NEW

Just to make things easy, we'll begin with a program that has many familiar features plus a couple of new ideas. This program rolls *Tunnels and Trolls* characters — as many as you want.

```
100 REM**CREATE A CHARACTER
110 GR. 0
120 DIM GAME$(20),ABBR$(3),KY$(1)
```

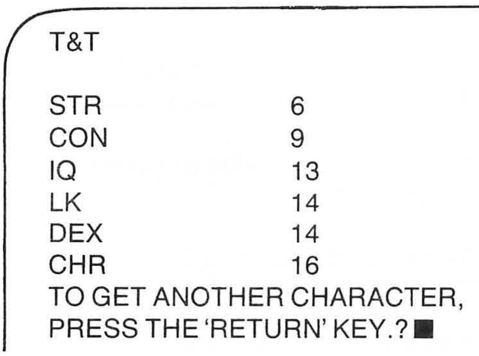
(continued)

```
200 REM**GAME AND ABBREVIATIONS
210 DATA T&T,STR,CON,IQ,LK,DEX,CHR,ZZZ

300 REM**PRINT NAME OF GAME
310 PR. CHR$(125)
320 RESTORE
330 READ GAME$: PR. GAME$

400 REM**ROLL & PRINT CHARACTERISTICS
410 PR.
420 READ ABBR$
430 IF ABBR$="ZZZ" THEN 510
440 GOSUB 910
450 PR. ABBR$,DICE
460 GOTO 420

500 REM**PRESS 'RETURN' TO DO AGAIN
510 PR.
520 PR. "TO GET ANOTHER CHARACTER,"
530 PR. "PRESS THE 'RETURN' KEY.";
540 IN. KY$: GOTO 310

900 REM**DICE SUBROUTINE
910 D1 = INT(6*RND(0))+1
920 D2 = INT(6*RND(0))+1
930 D3 = INT(6*RND(0))+1
940 DICE = D1 + D2 + D3
950 RETURN
```

ZZZ is the "end of data" flag. See line 430.

This statement checks for the "end of data" flag.

Enter and run the program. It might go this way:



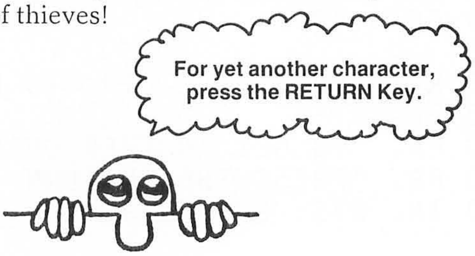
```
T&T

STR          6
CON          9
IQ           13
LK           14
DEX          14
CHR          16
TO GET ANOTHER CHARACTER,
PRESS THE 'RETURN' KEY.?■
```

Beware! This nimble hobbit can talk you out of your last pair of shoelaces!

Want another character? Press the RETURN key.

```
T&T

STR               15
CON               13
IQ                9
LK                11
DEX               13
CHR               7
TO GET ANOTHER CHARACTER,
PRESS THE 'RETURN' KEY.? ■
```

This character and the hobbit complement each other very well. Hmmm...they could be an incredible pair of thieves!



Now let's have a look at the program. Line 120 reserves space for three string variables: GAME$, ABBR$ and KY$.

```
120 DIM GAME$(20),ABBR$(3),K$(1)
```

Make room for 20 characters, 3 characters, 1 character.

Look through the program. You will see GAME$ use in line 330; ABBR$ in lines 420, 430 and 450; KY$ in line 540. Remember, before using a string variable, you must first DIMension it.

Line 210 is a DATA statement containing the value of GAME$, six values of ABBR$, and an "end of data" flag called ZZZ.

```
210 DATA T&T,STR,CON,IQ,LK,DEX,CHR,ZZZ
```

value of GAME$ values of ABBR$ end of data flag

We called the game T&T. However, line 120 reserves room for up to 20 characters for the value of GAME$, so you can use the full name, TUNNELS & TROLLS, if you prefer.

Line 310 clears the screen. You will learn more about the CHR$ *function* in this chapter.

Next, something new. The RESTORE statement in line 320 tells the computer to begin again at the first item of data, even if it has previously read all the data. Try this: press BREAK to stop the program; delete line 320 (type 320 and press RETURN); then RUN the program. The computer will roll the first character.

```
T&T

STR            13
CON            16
IQ             12
LK             5
DEX            11
CHR            9
TO GET ANOTHER CHARACTER,
PRESS THE 'RETURN' KEY.?■
```

Press the RETURN key. Oops! This is what you see.

```
ERROR   6 AT LINE 320
■
```

This happened because the computer has already READ all the DATA in rolling the first character. So, put RESTORE back into the program to tell the computer to begin at the first item in the first DATA statement (in case there are two or more DATA statements).

EXPERIMENT! Try both of these short programs:

```
10 GR. 0
20 DIM A$(40)
30 RESTORE
40 READ A$
50 PRINT A$
60 GOTO 30
90 DATA REINCARNATED!
```

RESTOREd to Life!

```
10 GR. 0
20 DIM A$(40)
30 READ A$
40 PRINT A$
50 GOTO 30
90 DATA REST IN PEACE
```

Embalmed Forever

Line 330 reads and prints the name of the game, the value of GAME$. Since it is preceded by RESTORE, line 330 will read the first value in the first DATA statement.

On to block 400! Line 420 reads a value of ABBR$, first STR, then CON and so on to ZZZ. The first six items (STR through CHR) are the basic characteristics of a T&T adventurer. The last item (ZZZ) is a "flag," which signals that all the interesting data have been read. This flag is sensed by line 430.

```
430 IF ABBR$ = "ZZZ" THEN 510
```

IF [the value of ABBR$ is the flag ZZZ] THEN go to line 510

For STR, CON, IQ, LK, DEX and CHR, the condition is false. The computer rolls and prints the information for the characteristic. Then the computer reads ZZZ. Now the condition is true — the computer goes to line 510.

Block 500 tells you how to get another character. Line 540 causes the computer to print a question mark, turn on the cursor and wait. You need only press RETURN to tell the computer to continue to the GOTO 310 statement in line 540 and thus go back for another character.

Questions

1. Is the program below like RESTOREd to Life or like Embalmed Forever?

```
10 GR. 0
20 DIM A$(40)
30 RESTORE
40 READ A$
50 PRINT A$
60 GOTO 40
90 DATA WHAT HAPPENED?
```

It's like ____________________

2. In the game of *RuneQuest* (RQ), the characteristics are STR, CON, SIZ, INT, POW, DEX and CHA. Modify the CREATE A CHARACTER program so the computer rolls a *RuneQuest* character.

Answers

1. Embalmed Forever. But if you change line 60 to GOTO 30, it will be like RESTOREd to Life!

2. Change only line 210! Here are three ways to write line 210.

```
210 DATA RuneQuest,STR,CON,SIZ,INT,POW,DEX,CHA,ZZZ
210 DATA RUNEQUEST,STR,CON,SIZ,INT,POW,DEX,CHA,ZZZ
210 DATA RQ,STR,CON,SIZ,INT,POW,DEX,CHA,ZZZ
```

You can just as easily change the program to roll a *Dungeons & Dragons*(D&D) character. The characteristics are STR,CON,INT,WIS,DEX and CHA.

The Characteristics for *Advanced Dungeons & Dragons* (AD&D) are the same as for D&D. Suppose you want the computer to print the full name of AD&D, all 27 characters, including spaces. How would you write line 120?

GUESS MY WORD

Next, a game of GUESS MY WORD. The computer chooses a three-letter word from a list of words in DATA statements. You guess the word. After an incorrect guess, the computer tells you to guess a "lower" word or a "higher" word. Here is the first part of the program and the list of words.

```
100 REM**GUESS MY WORD
110 GR. 0
120 DIM W$(3),G$(3),KY$(1)

300 REM**TELL HOW TO PLAY
310 PR. CHR$(125)
320 PR. "I'll think of a 3-letter word."
330 PR. "My word is between AAA and ZZZ."
340 PR.
350 PR. "My lowest word is AAA."
360 PR. "My highest word is ZZZ."
```

```
400 REM**PICK RANDOM WORD FROM DATA
410 RESTORE
420 READ NW
430 RW = INT(NW*RND(0))+1
440 FOR K=1 TO RW
450   READ W$
460 NEXT K
```

Don't RUN it yet.
We aren't finished.

```
1000 REM**VALUE OF NW AND NW WORDS
1010 DATA 44
1020 DATA AHA,ARK,BAH,CAT,DIG,DUO
1030 DATA EBB,ELF,FLY,FUN,GNU,HAW
1040 DATA HEX,JET,JOY,LAX,LEU,MEW
1050 DATA MUD,NIX,NUT,OAF,ODD,ORB
1060 DATA PAL,PLY,RAM,RED,ROC,SHH
1070 DATA SKY,SOL,TAX,TOO,UGH,VIA
1080 DATA WAG,WAX,WHO,WOW,YAK,YOU
1090 DATA ZIP,ZOO
```

Lines 1020-1090
contain 44 words.

In line 420, the computer reads the number of words (NW). Since line 420 is preceded by a RESTORE statement, the value of NW will be the first item in the first DATA statement. So, NW becomes 44.

Well, if NW is 44, the possible values of RW in line 430 are 1 to 44. Suppose RW is 13. The FOR-NEXT loop in lines 440 through 460 reads 13 values of W$, each replacing the previous one. The 13th value is HEX. Or, if RW is 23, W$ will end with ODD as its value. Any of the 44 words in lines 1020 through 1090 are possible values of W$.

You can put your list of words in lines 1020 on. If you do, count them and put the number of words in line 1010.

Having selected a random word, the computer moves on. What next?

```
500 REM**GET GUESS
510 PR.
520 PR. "YOUR GUESS";: IN. G$

600 REM**IF NOT CORRECT, GIVE HINT
610 IF G$<W$ THEN PR."TRY A HIGHER WORD": GOTO 510
620 IF G$>W$ THEN PR."TRY A LOWER WORD": GOTO 510
```

Aha! If the guess (G$) is "too low," then G$ < W$ is true and the computer prints: TRY A HIGHER WORD. But if the guess is too "high," then G$ < W$ is false and G$ > W$ is true. In this case, the computer prints: TRY A LOWER WORD.

A *lower* word is lower in the alphabet (towards AAA).

A *higher* word is higher in the alphabet (towards ZZZ).

Z
Y
X
W
V
U
T
S
R
Q
P
O
N
M
L
K
J
I
H
G
F
E
D
C
B
A

When you guess the computer's word, this happens:

```
700 REM**WINNER!
710 PR. "THAT'S IT! YOU GUESSED MY WORD."

800 REM**TELL HOW TO PLAY AGAIN
810 PR.
820 PR."TO PLAY AGAIN, PRESS THE 'RETURN' KEY.";
830 IN. KY$: GOTO 310
```

Put it all together, enter the program and play. You may wish to change the hints in lines 510 and 520. Make that easier to do by these changes to the program.

```
120 DIM M1$(40),M2$(40),W$(3),G$(3),KY$(1)
130 M1$ = "TRY A HIGHER WORD."
140 M2$ = "TRY A LOWER WORD."
610 IF G$<W$ THEN PR. M1$: GOTO 510
620 IF G$>W$ THEN PR. M2$: GOTO 510
```

IMPORTANT NOTICE! When you run this program, enter your guess as a three-letter word with all letters in *caps*.

Guess CAT instead of cat or Cat.
Guess JOY instead of joy or Joy.

Questions

1. (a) Which is lower, JAM or RED? ____
   (b) Which is lower, CAT or CAR? ____
   (c) Which is higher, ANT or ZOO? ____
   (d) Which is higher, SUN or SUM? ____
2. How would you change the original program to use words with up to eight letters?

   ____

Answers

1. (a) JAM is lower then RED.
   (b) CAR is lower than CAT.
   (c) ZOO is higher than ANT.
   (d) SUN is higher then SUM.
2. First, change line 120 as follows

```
120 DIM W$(8),G$(8),KY$(1)
```

Then, put your words in DATA statements, beginning at line 1020. Put the number of words in the DATA statement in line 1010.

Add some excitement in block 700. How about some sound? Or print the word all over the screen. Or switch to GRAPHICS 1 and print the winner's message in big letters with sound. Or go to GRAPHICS 3, 5 or 7. Print the winner's message in the text window and put on a colorful graphics show in the top part of the screen. Include sound, of course.

GUESS MY LETTER

For the very young, a game called GUESS MY LETTER. Here is the first piece.

```
100 REM**GUESS MY LETTER
110 GR. 0
120 DIM ABC$(26),L$(1),G$(1),KY$(1),MI$(6),M2$(5)
130 ABC$ = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
140 M1$ = "HIGHER"
150 M2$ = "LOWER"

300 REM**TELL HOW TO PLAY
310 PR. "I'LL THINK OF A LETTER, A TO Z."
320 PR.
330 PR. "MY HIGHEST LETTER IS Z."
340 PR.
350 PR. "MY LOWEST LETTER IS A."
```

Line 130 assigns to the string variable ABC$ a string value consisting of all 26 letters of the alphabet. Thus, ABC$ is a string in which the first character is A, the second character is B, the third character is C and so on. The 26th character is Z.

Perhaps you have guessed that we are going to tell the computer to select, at random, *one letter* from ABC$. Block 400 selects one letter from ABC$ and assigns it as the value of L$.

```
400 REM**PICK RANDOM LETTER FROM ABC$
410 RL = INT(26*RND(0))+1
420 L$ = ABC$(RL,RL)
```

The value of RL will be a random integer from 1 to 26. Line 420 uses this value to select one letter from ABC$.

- If RL is 1, then ABC$(RL,RL) is the letter A.
- If RL is 2, then ABC$(RL,RL) is the letter B.
- If RL is 3, then ABC$(RL,RL) is the letter C.
- and so on. If RL is 26, then ABC$(RL,RL) is the letter Z.

The rest of the game is much like the game called GUESS MY WORD:

```
500 REM**GET A GUESS
510 PR.
520 PR. "GUESS MY LETTER";: IN. G$

600 REM**IF NOT CORRECT, GIVE HINT
610 IF G$<L$ THEN PR. M1$: GOTO 510
620 IF G$>L$ THEN PR. M2$: GOTO 510

700 REM**WINNER!
710 PR. "THAT'S IT! YOU GUESSED MY LETTER."

800 REM**TELL HOW TO PLAY AGAIN
810 PR.
820 PR."TO PLAY AGAIN, PRESS THE 'RETURN' KEY.";
830 IN. KY$: GOTO 310
```

We leave it to you to add some pizzazz to block 700. Use sound, color, or anything else you can think of as the "reward" for a correct guess.

Boggled by ABC$(RL,RL)? This tells the computer to select a *substring* of ABC$. A substring is a string within a string. Try this ancient game of Words within Words:

The word PROVERB contains these shorter words:

PRO PROVE PROVER ROVE ROVER OVER VERB

Now let WORD$ = "PROVERB"

- WORD$(1,3) is "PRO" (PROVERB)
- WORD$(3,6) is "OVER" (PROVERB)
- WORD$(4,7) is "VERB" (PROVERB)

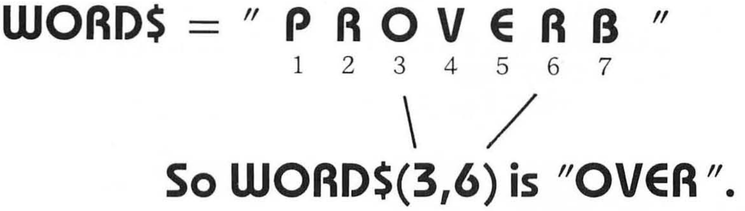
The word PROVERB has seven characters, numbered one through seven:

1	2	3	4	5	6	7
P	R	O	V	E	R	B

Assume the value of WORD$ is "PROVERB".

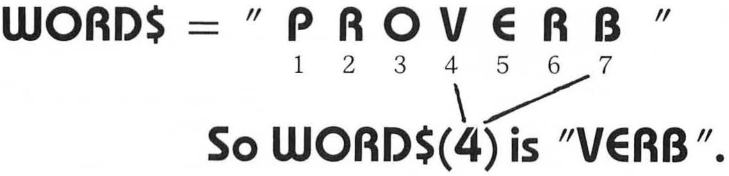
- WORD$(1,3) is the *substring* consisting of characters one through three of the value of WORD$.

- WORD$(3,6) is the *substring* consisting of characters three through six of the value of WORD$.
- WORD$(4,7) is the *substring* consisting of characters four through seven of the value of WORD$.



Here is a slightly shorter way to get a substring from a certain character to the right end of the word.

WORD$(4) is the substring consisting of all characters from character number 4 to the right end of the value of WORD$.



Questions

1. Suppose WORD$ = "PROVERB"

   (a) What is WORD$(1,6)? ____________

   (b) What is WORD$(2,5)? ____________

   (c) What is WORD$(1,7)? ____________

   (d) What is WORD$(1)? ____________

   (e) How do we tell the computer we want the substring "ROVER"? WORD$(____,____)

   (f) How do we tell the computer we want the substring "ER"?

   ____

   (g) How do we tell the computer we want the substring "ERB"?

   ____

2. Complete the following program to make and print 40 random 3-letter "words." Each "word" consists of a consonant, a vowel and a consonant. For example: TOK, CIL, MAR etc.

```
100 REM**RANDOM 3-LETTER "WORDS"
110 GR. 0
120 DIM V$(5),C$(21),L1$(1),L2$(1),L3$(1),KY$(1)
130 V$ = "AEIOU"
140 C$ = "BCDFGHJKLMNPQRSTVWXYZ"

200 REM**MAKE & PRINT 40 WORDS
210 PR. CHR$(125)
220 FOR WORD=1 TO 40

230   R = ______________________________
                                          } First random
235   L1$ = ____________________________    letter (consonant).

240   R = ______________________________
                                          } Second random
245   L2$ = ____________________________    letter (vowel).

250   R = ______________________________
                                          } Third random
255   L3$ = ____________________________    letter (consonant).

260   PR. L1$; L2$; L3$,————————— Print the word.

270 NEXT WORD

300 REM**TELL HOW TO DO AGAIN
310 PR.
320 PR. "FOR MORE WORDS, PRESS 'RETURN'.";
330 IN. KY$: GOTO 210
```

Answers

1. (a) PROVER (b) ROVE (c) PROVERB

   (d) PROVERB Start at the first character and take everything to the end of the word.

   (e) WORD$(2,6) (f) WORD$(5,6)

   (g) WORD$(5,7) or WORD$(5)

2.
```
230   R = INT(21*RND(0))+1
235   L1$ = C$(R,R)

240   R = INT(5*RND(0))+1
245   L2$ = V$(R,R)

250   R = INT(21*RND(0))+1
255   L3$ = C$(R,R)
```

The Land of ATASCII

Deep down inside the computer, each keyboard character has its very own numeric code, called an ATASCII code. ASCII means "American Standard Code for Information Interchange." ATASCII is ATari ASCII.

- The ATASCII code for A is 65.
- The ATASCII code for B is 66.
- The ATASCII code for C is 67.

You have probably guessed that the code for D is 68. The code for Z is 90. For the upper case (CAPS) letters A to Z, the ATASCII codes are 65 to 90.

ATARI BASIC has a built-in function, called ASC, which gives the ATASCII code for any character. Clear the screen and try these.

You type: PR. ASC("A")
It prints: 65

Remember to enclose A in quotation marks.

You type: PR. ASC("Z")
It prints: 90

You type: PR. ASC(" ")
It prints: 32

put a space here

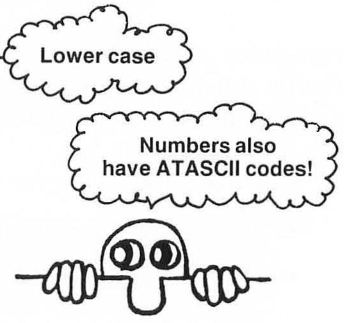
Yes, even a space has an ATASCII code!

EXPERIMENT! Use this program to find the ATASCII codes for keyboard characters.

```
100 REM**ATASCII CODES, USING ASC
110 GR. 0
120 DIM KEY$(1)

200 REM**GET CHARACTER, PRINT ATASCII
210 PR. "KEYBOARD CHARACTER";:IN. KEY$
220 CODE = ASC(KEY$)
230 PR. "THE ATASCII CODE IS "; CODE
240 PR.: GOTO 210
```

Here is what happened when we ran the program:

```
KEYBOARD CHARACTER?A
THE ATASCII CODE IS 65

KEYBOARD CHARACTER?Z
THE ATASCII CODE IS 90

KEYBOARD CHARACTER?a
THE ATASCII CODE IS 97

KEYBOARD CHARACTER?1
THE ATASCII CODE IS 49

KEYBOARD CHARACTER?*
THE ATASCII CODE IS 42

KEYBOARD CHARACTER? ●
THE ATASCII CODE IS 20

KEYBOARD CHARACTER? and so on. Take over!
```



A graphics character (CTRL T)

Look at line 220 in the program.

```
220 CODE = ASC(KEY$)
```

This line computes the ATASCII code of the value of KEY$ AND ASSIGNS this code as the value of CODE. Well, not quite. The value of KEY$ could be a string with more than one character. In this case, ASC computes the ATASCII code of the first character in the string.

For a complete list of ATASCII codes, see Appendix E.

ATASCII characters can be letters A to Z, or numerals 0 to 9, or punctuation marks (. , : and so on), or special characters (* , %, and so on), and *lots more*. To help you really explore the world of ATASCII, use the CHR$ function.

CHR$(CODE)

ATASCII Code, 0 to 255.

The CHR$ function gives the character that corresponds to the ATASCII code enclosed in parentheses.

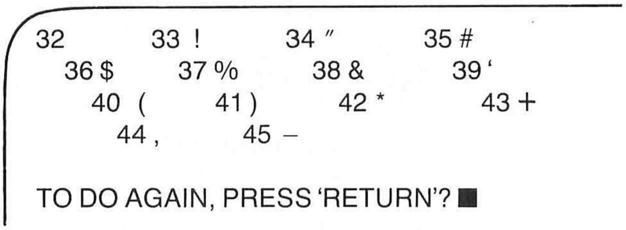
You type: PR. CHR$(65)
It prints: A
You type: PR. CHR$(20)
It prints: ●

Some ATASCII codes are not printed characters, but instead perform control functions — they cause the computer to do something. For example: PR. CHR$(125) tells the computer to clear the screen. For a complete list of ATASCII codes visit Appendix E.

EXPERIMENT! Use the following program to print characters that correspond to ATASCII codes:

```
100 REM**ATASCII CHARACTERS, USING CHR$
110 GR. 0
120 DIM KY$(1)

200 REM**ASK FOR ATASCII CODES
210 PR. CHR$(125)
220 PR. "FIRST ATASCII CODE";:IN. FIRST
230 PR. "LAST ATASCII CODE;:IN. LAST

300 REM**PRINT CODES & CHARACTERS
310 PR. CHR$(125)
320 FOR CODE=FIRST TO LAST
330   PR. CODE; CHR$(32); CHR$(CODE),
340 NEXT CODE

400 REM**TELL HOW TO DO AGAIN
410 PR.: PR.
420 PR. "TO DO AGAIN, PRESS 'RETURN'";
430 IN. KY$: GOTO 210
```

Here is what happened when we ran the program using 32 for FIRST ATASCII CODE and 45 for LAST ATASCII CODE:



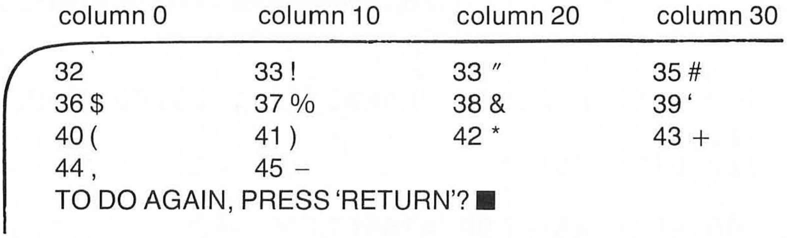
```
32          33 !         34 "         35 #
   36 $        37 %         38 &         39 '
      40 (        41 )         42 *         43 +
         44 ,        45 -

TO DO AGAIN, PRESS 'RETURN'? ■
```

Oops! We thought the codes and characters would appear in neat vertical columns. They don't because the left margin is set to column 2. The comma causes the computer to move 10 places from the previous printing position. If this causes a move from the right end of a line to the left end of the next line, the computer skips over columns 0 and 1. So everything is pushed an extra two places to the right.

OK, let's reset the left margin! We'll use one of those mysterious POKE commands to set the left margin to column 0. Add this line to the program.

```
130 POKE 82,0
```

Run the program and you will see this:



```
32          33 !        33 "        35 #
36 $        37 %        38 &        39 '
40 (        41 )        42 *        43 +
44 ,        45 -
TO DO AGAIN, PRESS 'RETURN'? ■
```

Now use the program to learn more about ATASCII codes. Remember, an ATASCII code is a number in the range 0 to 255.

- Try upper case letters, 65 to 90.
- Try lower case letters, 97 to 122.
- Try graphics characters, 0 to 26.
- Try inverse upper case letters, 193 to 218.
- Try inverse lower case letters, 225 to 250.
- Try inverse graphics characters, 128 to 154.
- Try codes we haven't mentioned.

When you are finished, you can leave the left margin set to column zero or change it back to column two like this: POKE 82,2

Pressing SYSTEM RESET also returns the left margin to column two.

Questions

1. Complete the following:

(a) You type: C$ = "A"
You type: PR.ASC(C$)
It prints: ____

(b) You type: C$ = "a"
You type: PR.AXC(C$)
It prints: ____

(c) You type: C$ = "CAT"
You type: PR.ASC(C$)
It prints: ____

(d) You type: C$ = "A "
You type: PR.ASC(C$)
It prints: ____

2. Suppose the computer obeys the following two lines:

```
410 RL = INT(26*RND(0))+65
420L$  = CHR$(RL)
```

Describe the value of L$. ____________

Answers

1. (a) 65 (b) 97 (c) 67 (d) 32 The code for space

2. Line 310 computes a random number from 65 to 90 and assigns it as the value of RL. Therefore, in line 420, the value of L$ will be a random letter from A to Z. Of course, L$ must appear in a DIM statement earlier in the program. Use these lines in the GUESS MY LETTER program to pick a random letter. If you do, you can delete line 130 which assigns the alphabet to ABC$.

WORD MAKER

Suppose you are creating a fantasy adventure such as *Dungeons and Dragons*, *RuneQuest*, or *Tunnels and Trolls*. You may wish to use unusual names for your heroes, wizards, monsters and other creatures. You could, of course, borrow names from fantasy adventure books such as *Lord of the Rings*. But perhaps you prefer to invent your own.

Why not use your computer to help you invent names? Sounds OK, but how do we get the computer to print names that are pronounceable (or almost so) and seem to be unusual, exotic or even fantastic?

That's the problem. Write a program to generate and print or display random names that might be used in a fantasy or science fiction game or story. Try this program to make five-letter words consisting of consonant, vowel, consonant, vowel, consonant. For example, here are some *possible* words: MALOK, BARAK, CONAN, DYMAX, KUMAN, LOSAS, MIKOS.

We begin by saving space for string variables and setting the left margin to zero. We want our words to line up in neat vertical columns:

```
100 REM**WORD MAKER
110 DIM C$(21),RC$(1),V$(6),RV$(1)
120 DIM WORD$(20),KY$(1)
130 GR. 0

200 REM**SET LEFT MARGIN TO ZERO
210 POKE 82,0
```

Arbitrarily and capriciously, we decide to make 60 words, 15 lines with four words per line. We call a subroutine to make one word:

```
400 REM**MAKE & PRINT WORDS
410 PR. CHR$(125)
420 FOR W=1 TO 60
430  GOSUB 710      :REM**MAKE A WORD
440  PR. WORD$,
450 NEXT W
```

After it prints 60 words, we want the computer to tell how to make another batch of words:

```
500 REM**TELL HOW TO DO AGAIN
510 PR.: PR.
520 PR. "FOR MORE WORDS, PRESS 'RETURN'";
530 IN. KY$: GOTO 410
```

Three subroutines complete the program. The first makes one word. In turn, it uses two other subroutines, one to add a consonant and another to add a vowel to the word being made.

```
700 REM**MAKE A WORD SUBROUTINE
710 WORD$ = ""
720 GOSUB 810  :REM**ADD CONSONANT
730 GOSUB 910  :REM**ADD VOWEL
740 GOSUB 810
750 GOSUB 910
760 GOSUB 810: RETURN
```

Finally, here are the subroutines to add a consonant and add a vowel:

```
800 REM**ADD CONSONANT SUBROUTINE
810 C$ = "BCDFGHJKLMNPQRSTVWXYZ"
820 RC = INT(21*RND(0))+1
830 RC$ = C$(RC,RC)
840 WORD$(LEN(WORD$)+1) = RC$
850 RETURN

900 REM*ADD VOWEL SUBROUTINE
910 V$ = "AEIOUY"
920 RV = INT(6*RND(0))+1
930 RV$ = V$(RV,RV)
940 WORDS(LEN(WORD$)+1) = RV$
950 RETURN
```

We use Y as both vowel and consonant.

WORD$	RC$ or RV$	LEN(WORD$)	WORD$(LEN(WORD$)+1)
empty	M	0	M
M	A	1	MA
MA	Z	2	MAZ
MAZ	I	3	MAZI
MAZI	N	4	MAZIN

Here is a sequence of examples to help you get the hang of how this works. Do them slowly and ponder each result. Clear the screen, then begin:

```
1)    You type:  DIM W$(20)
                 W$ = ″123456″
                 PR. W$
     It prints:  123456

2)    You type:  W$(1,1) = ″D″
                 PR. W$
     It prints:  D23456

3)    You type:  W$(2,3) = ″RA″
                 PR. W$
     It prints:  DRA456

4)    You type:  W$(4) = ″GON″
                 PR. W$
     It prints:  DRAGON

5)    You type:  W$(7) = ″S″
                 PR. W$
     It prints:  DRAGONS

6)    You type:  W$(LEN(W$)+1) = ″MOKE″
                 PR. W$
     It prints:  DRAGONSMOKE
```

Got it? Please say yes!

Questions

1. Our WORD MAKER program makes words of the form CVCVC, where ″C″ stands for ″consonant″ and ″V″ stands for ″vowel.″ For example,

```
MAZIN
↑↑↑↑↑
CVCVC
```

Change the program so it makes words in the form CCVCV (consonant, consonant, vowel, consonant, vowel). With this change, you might see words such as FRODO, THENA, STOKI, CHATO, XXOPU and so on.

2. Why not let the user decide on the structure of the words (sequence of consonants and vowels). Write a new block 200 so the computer asks:

```
WORD STRUCTURE? ■
```

   Now someone can enter CVCVC or CCVCV or VCCVCV or whatever sequence of consonants and vowels he or she wants. The only other thing you need change is the WORD MAKER SUBROUTINE.

3. Pretend you are the computer and complete each of the following:

```
        You type:  DIM NAME$(20),PREFIX$(10),SUFFIX$(10)
                   PREFIX$ = "BIL"
                   SUFFIX$ = "BO"
(a)     You type:  PR. PREFIX$;SUFFIX$
       It prints:  _______
(b)     You type:  NAME$ = PREFIX$
       It prints:  PR. NAME$

                   _______
(c)     You type:  NAME$(LEN(NAME$)+1) = SUFFIX$
       It prints:  PR. NAME$

                   _______
```

Answers

1. Change only block 700.

```
700 REM**MAKE A WORD SUBROUTINE
710 WORD$ = ""
720 GOSUB 810    :REM**ADD CONSONANT
730 GOSUB 810
740 GOSUB 910    :REM**ADD VOWEL
750 GOSUB 810
760 GOSUB 910: RETURN
```

2. We leave this as a challenge for you.

3. (a) BILBO

   (b) BIL

   (c) BILBO

CREATE A CHARACTER

ATARI BASIC provides a function called VAL to *convert strings to numbers*. Here are some examples:

VAL("1") is 1. VAL("2") is 2.
VAL("3") is 3. VAL("9") is 9.

Also:

VAL("12") is 12. VAL("123") is 123.
VAL("1234") is 1234. and so on

However,

VAL("ABC") gives an ERROR 18 message.

Letters are OK if they occur to the right of a number.

VAL("1ABC") is 1. VAL("123ABC") is 123.

Here is another way to get an error message: VAL(123)

VAL (____)

/

Must be a string or string variable or string expression.

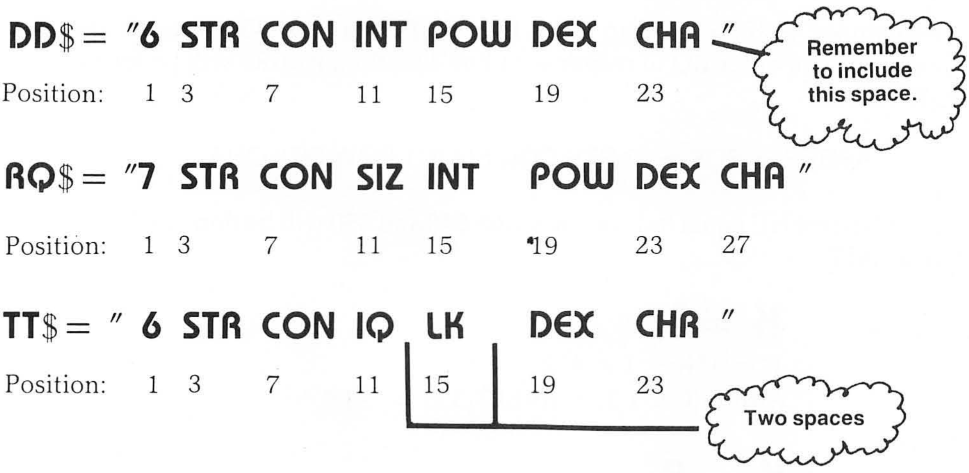
EXPERIMENT! Use this program to play with VAL.

```
10 REM**EXPERIMENT WITH VAL
20 GR. 0
30 DIM X$(20)
40 IN. X$
50 PR. VAL(X$)
60 PR.: GOTO 40
```

Below is the beginning of a program to create a character for any of the game systems *Dungeons & Dragons*, *RuneQuest*, or *Tunnels & Trolls*.

```
100 REM**CREATE A CHARACTER
110 DIM DD$(26),RQ$(30),TT$(26),KY$(1),
GAMES$(2),ABBR$(30)
120 GR. 0

200 REM**ABBREVIATIONS
210 DD$ = "6 STR CON INT POW DEX CHA "
220 RQ$ = "7 STR CON SIZ INT POW DEX CHA "
230 TT$ = "6 STR CON IZ  LK  DEX CHR "
```

Each string variable contains the abbreviations for one game and the number of abbreviations in the string. Each abbreviation is a substring consisting of the abbreviation followed by one or two spaces to make a total of exactly four positions:


DD$= "6 STR CON INT POW DEX CHA "
Position: 1 3 7 11 15 19 23
Remember to include this space.
RQ$= "7 STR CON SIZ INT POW DEX CHA "
Position: 1 3 7 11 15 19 23 27
TT$= " 6 STR CON IQ LK DEX CHR "
Position: 1 3 7 11 15 19 23
Two spaces

Let's move on. Here's more of the program:

```
300 REM**TELL WHAT TO DO
310 PR. CHR$(125)
320 PR. "I CAN CREATE A CHARACTER FOR"
330 PR.
340 PR. " DUNGEONS & DRAGONS (DD)"
350 PR. " RUNEQUEST          (RQ)"
360 PR. " TUNNELS & TROLLS   (TT)"
```

```
400 REM**FIND OUT WHICH GAME
410 PR.
420 PR. "WHICH GAME (DD, RQ, OR TT)";:IN. GAME$
430 IF GAME$="DD" THEN ABBR$=DD$: GOTO 510
440 IF GAME$="RQ" THEN ABBR$=RQ$: GOTO 510
450 IF GAME$="TT" THEN ABBR$=TT$: GOTO 510
460 GOTO 310
```

If you enter DD, RQ, or TT, the computer moves on to line 510 after first setting ABBR$ to the appropriate characteristic string. Time to roll a character:

```
500 REM**ROLL THE CHARACTERISTICS
510 NC = VAL(ABBR$)
520 FOR K=1 TO NC
530   C = 4*K - 1
540   GOSUB 910
550   PR. ABBR$(C,C+3), DICE
560 NEXT K
```

Remember, ABBR$ is set to DD$, RQ$ or TT$ in lines 430 through 450. For example, if you enter RQ in response to line 420, then ABBR$ will be set equal to RQ$.

ABBR$ = RQ$ = "7 STR CON SIZ INT POW DEX CHA "

Line 510 sets NC equal to 7. So lines 530, 540 and 550 will be done for K = 1, 2, 3, 4, 5, 6 and 7.

K = 1

C = 4*K − 1 = 4*2 − 1 = 3
ABBR$(C,C+3) = ABBR$(3,6) = "STR "

K = 2

C = 4*K − 1 = 4*2 − 1 = 7
ABBR$(C,C+3) = ABBR$(7,10) = "CON "

K = 7

C = 4*K − 1 = 4*7 − 1 = 27

ABBR$(C,C+3) = ABBR$(27,30) = "CHA "

Finally, the last two pieces of the program:

```
600 REM**TELL HOW TO DO AGAIN
610 PR.
620 PR. "FOR ANOTHER, PRESS 'RETURN'";
630 IN. KY$: GOTO 310

900 REM**DICE SUBROUTINE
910 D1 = INT(6*RND(0))+1
920 D2 = INT(6*RND(0))+1
930 D3 = INT(6*RND(0))+1
940 DICE = D1 + D2 + D3
950 RETURN
```

Questions

1. When the computer asks WHICH GAME (DD, RQ, OR TT)?■, what happens if you enter:

   (a) DD ____________________

   (b) MONOPOLY ____________________

2. In lines 210, 220 and 230, the string values for DD$, RQ$ and TT$ each end with a space. Why is this space required? ____________________

   ____

Answers

1. (a) The computer creates a character using the six abbreviations in DD$.

   (b) The conditions in lines 430, 440 and 450 are all false. So the computer goes back to line 310 and asks again.

2. Each abbreviation occupies exactly four spaces. Line 550 prints a substring from ABBR$ exactly four characters long, Beginning with the first letter of the abbreviation and ending with a space. Therefore, the last abbreviation must be four characters long. Try it without the final space and see what happens. Of course, you could pack the information as follows:

```
210 DD$ = "6STRCONINTWISDEXCHA"
220 RQ$ = "7STRCONSIZINTPOWDEXCHA"
230 RQ$ = "6STRCONIQ LK DEXCHR"
```

If you pack the information as shown above, how must you change lines 530 and 550?

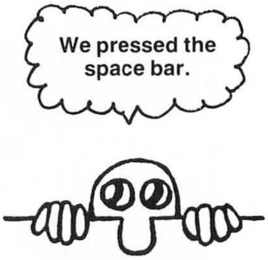
OPEN A CHANNEL & GET A CHARACTER

Inside the computer, there are many pathways for information to move from place to place. They are called *channels*. We will describe how to OPEN a channel and GET a character from the keyboard. Experiment with this little program.

```
100 REM**EXPERIMENT WITH OPEN & GET
110 GR. 0

200 REM**OPEN CHANNEL TO KEYBOARD
210 OPEN #1,4,0,"K:"

300 REM**TELL WHAT TO DO
310 PR. "PRESS A KEY."

400 REM**GET & PRINT KEY & GO FOR MORE
410 GET #1, KEY
420 PR. "YOU PRESSED "; CHR$(KEY)
430 PR. "ITS ATASCII CODE IS "; KEY
440 PR.: GOTO 310
```

Here is what happened when we ran the program:

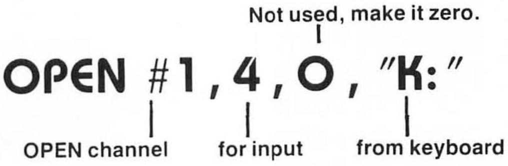
```
PRESS A KEY
YOU PRESSED A
ITS ATASCII CODE IS 65

PRESS A KEY
YOU PRESSED 7
ITS ATASCII CODE IS 55

PRESS A KEY
YOU PRESSED
ITS ATASCII CODE IS 32

...and so on. Press a key!
```



Spend some time with this program.

- Hold CTRL down and press a key.
- Try some LOWR CASE letters.
- Try some inverse video characters.

Line 210 tells the computer to OPEN channel number 1 for input from the keyboard.

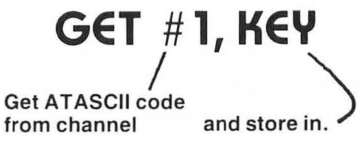


Not used, make it zero.

OPEN #1,4,0,"K:"

OPEN channel for input from keyboard

Line 410 gets the ATASCII code of a key that is pressed and stores this number as the value of KEY.



GET #1, KEY

Get ATASCII code from channel and store in.

This is done as soon as you press the key. You don't have to press RETURN. Please note that channel #1 is used in both lines 210 and 410.

Use this program to GET and play some music:

```
100 REM**KEYBOARD MUSIC USING GET
110 GR. 0

200 REM**OPEN CHANNEL TO KEYBOARD
210 OPEN #1,4,0,"K:"

300 REM**TELL WHAT TO DO
310 PR. "PRESS ANY KEY, 1 TO 8, OR"
320 PR. "PRESS ZERO (0) FOR SILENCE."

400 REM**GET & VERIFY A KEY
410 GET #1, KEY
420 IF KEY<48 OR KEY>55 THEN 410

500 REM**PLAY THE NOTE
510 ON KEY-47 GOSUB 610,620,630,640,650,660,
670,680,690
520 SO. 0,TN,10,10
530 GOTO 410

600 REM**SELECT TONE NUMBER SUBROUTINE
610 TN =   0: RETURN
620 TN = 121: RETURN
630 TN = 108: RETURN
640 TN =  96: RETURN
650 TN =  91: RETURN
660 TN =  81: RETURN
670 TN =  72: RETURN
680 TN =  64: RETURN
690 TN =  60: RETURN
```

Enter this program and play some music. Use the number keys 1 to 8 for tones. Use 0 to shut off the sound. You will quickly find some limitations. For example, you can't play the same tone twice in succession and hear two distinct sounds.

Questions

1. In the KEYBOARD MUSIC USING GET program:

   (a) What is the purpose of line 420?

   ____

   (b) In line 510, what are the possible values of KEY-47?______________

2. If you pluck a string on a quitar or strike a key on a piano, the sound "decays," dies out naturally. Change block 500 so the sound fades away to silence, unless you press another key.

Answers

1. (a) If the value of KEY is not an ATASCII code for a number key 0 to 8, the computer goes back to line 410.

   (b) The possible values of KEY, thanks to line 420, are 48 (ATASCII code for "0") to 56 (ATASCII code for "8"). So the possible values of KEY-47 are 1 to 9, just what we need in the ON—GOSUB statement.

2. Here is one way. Change block 500 from line 520 on, as follows:

```
500 REM**PLAY THE NOTE
510 ON KEY-47 GOSUB 610,620,630,640,650,660,
670,680,690
520 FOR L=10 TO 0 STEP -1
530   SO. 0,TN,10,L
540   TD = 10
550   FOR Z=1 TO TD: NEXT Z
560 NEXT L
570 GOTO 410
```

The loudness (L) fades from 10 to 0. Try other values of TD

SELF-TEST

Use your string magic to wrap-up this Self-Test.

1. Start with easy stuff by answering these questions or completing the sentences.

   (a) What does DIM ABC$(26),WORD$(20),KY$(1) tell the computer to do?

   ____

   (b) What does RESTORE tell the computer to do?

   ____

   (c) Which is true: "STAR" < "STARS" or "STARS" < "STAR"

   ____

   (d) Which is false: "ZZZY" < "ZZZ" or "ZZZ" < "ZZZY"

   ____

   (e) What is LEN("ABCDEFG")?

   ____

   (f) What is LEN("12345679")?

   ____

   (g) What is LEN("")?________

   (h) Since ASC("* ") is 42, then CHR$(42) must be ________.

2. If W$ = "DRAGONSMOKE", what is:

   (a) W$(1,7)? ________

   (b) W$(7,11)? ________

   (c) W$(3,3)? ________

   (d) W$(7)? ________

   And, for each substring of W$, write the corresponding expression:

   (e) W$(____,____) is "OK".

   (f) ____ is "RAG".

   (g) ____ is "E".

   (h) ____ is "RAGONSMOKE".

3. What is the printed result of the following program? *Don't* RUN it. Look at it, puzzle over it, figure out what it does.

```
100 REM**MYSTERY PROGRAM
110 DIM W$(3)
120 GR. 0

200 REM**HERE IS THE MYSTERY
210 NW = 0
220 READ W$
230 IF W$<>"***" THEN NW=NW+1: GOTO 220

300 REM**PRINT THE RESULT AND STOP
310 PR. "THERE ARE ";NW;" WORDS."
320 END

1000 REM**WORDS
1010 DATA ARK, CAT, ELF, FUN, GNU
1020 DATA HEX, JET, JOY, MEW, MUD
1030 DATA NUT, OAF, PAL, RAM, RED
1040 DATA SOL, TOO, UGH, VIA, WAG
1050 DATA YAK, ZIP, ZOO, ***
```

What does the computer print?______________________________

4. Now, using what you learned in question 3, rewrite the GUESS MY WORD game in pages 273 to 276. Follow the outline of REM statements.

```
100 REM**GUESS MY WORD
200 REM**COMPUTE NW=NUMBER OF WORDS
300 REM**TELL HOW TO PLAY
400 REM**PICK RANDOM WORD FROM DATA
500 REM**GET GUESS
600 REM**IF NOT CORRECT, GIVE CLUE
700 REM**WINNER!
800 REM**TELL HOW TO PLAY AGAIN
1000 REM**LIST OF WORDS + ***
```

Put as many three-letter words here as you want with * after last word.**

5. Complete the following program to play MUSIC FROM ATASCII CHARACTER. The program begins this way:

```
100 REM**MUSIC FROM ATASCII CHARACTERS
110 DIM STRNG$(100),C$(1),KY$(1)
120 GR. 0

200 REM**TELL WHAT TO DO
210 PR. CHR$(125)
220 PR. "ENTER ANY STRING AND I'LL PLAY"
230 PR. "MUSIC MADE FROM YOUR STRING."

300 REM**ASK FOR A STRING
310 PR.
320 PR. "YOUR STRING";: IN. STRNG$
```

You write the rest of the program. For each character in STRNG$, play the tone that corresponds to the ATASCII code of the character. After all characters have been "played," ask if the user wants to play again.

6. Complete this program to play music in the key of C:

```
100 REM**STRING MUSIC IN KEY OF C
110 DIM STRNG$(100)
120 GR. 0

200 REM**TELL WHAT TO DO
210 PR. CHR$(125)
220 PR. "ENTER A STRING WITH ONLY LETTERS"
230 PR. "A TO G. I'LL PLAY MUSIC IN THE"
240 PR. "KEY OF C MADE FROM YOUR STRING."
```

You write the rest. Play tones as follows for letters A to G.

C	D	E	F	G	A	B
121	108	96	91	81	72	64

For any character not a letter A to G, play silence.

7. Write a program to make sentences that begin with "The Dragon." A run might go like this.

```
Please give me:
           a verb ? ate
          a noun ? knight
      an article ? the
    an adjective ? lazy

The Dragon ate the lazy knight.

FOR ANOTHER, PRESS RETURN
```

Answers to Self-Test

1. (a) Reserve room in memory, as follows: 26 character positions for ABC$, 20 positions for WORD$, and one position for KY$.

   (b) Start at the first data item in the first DATA statement.

   (c) "STAR" < "STARS" is true.

   (d) "ZZZY" < "ZZZ" is false.

   (e) 7

   (f) 8 Did we fool you? Look again. There is no 8 in the string.

   (g) 0

   (h) *

2. (a) DRAGONS (b) SMOKE
   (c) A (d) SMOKE
   (e) W$(9,10) (f) W$(2,4)
   (g) W$(11,11) or W$(11) (h) W$(2,11) or W$(2)

3. There are 44 words. This program counts the words up to, but not including ***.

4. We leave this to you.

5.
```
400 REM**NUMBER OF CHARACTERS IN STRING
410 NC = LEN(STRNG$)

500 REM**PLAY, MAESTRO, PLAY!
510 FOR K=1 TO NC
520   C$ = STRNG$(K,K)
530   TN = ASC(C$)
540   TD = 10
550   FOR L=10 TO 0 STEP -1
560     SO. 0,TN,10,L
570     FOR Z=1 TO TD: NEXT Z
580   NEXT L
590 NEXT K

600 REM**TELL HOW TO DO AGAIN
610 PR.
620 PR. "TO DO AGAIN, PRESS 'RETURN'";
630 IN. KY$: GOT 210
```

Tone fades away.

6 and 7. We leave these to you. In problem 6, we suggest you use ON...GOTO or ON...GOSUB to select the tone numbers for letters A to G.

Chapter
Eleven

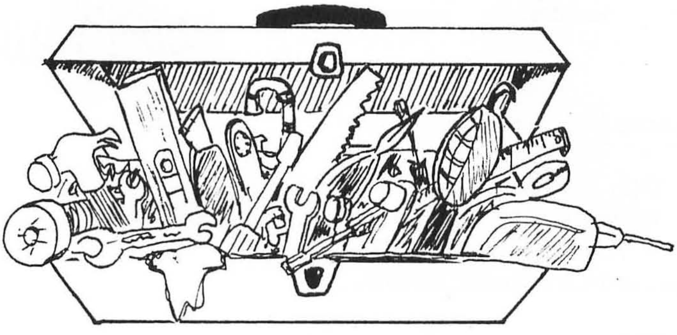
Subscripted Variables

In this chapter you will learn to use subscripted variables and bunches of subscripted variables, called arrays. You will use arrays of subscripted variables in programs to twinkle stars, play music, simulate coin flipping and dice rolling, count votes and compute information from quiz scores. Subscripted variables add a new dimension to your ability to make the computer do what you want it to do.

When you finish this chapter, you will be able to:

- Recognize and use subscripted numeric variables
- Use the DIM(for DIMension) statement to reserve memory space for arrays(lists) of subscripted variables
- Enter numbers into numeric arrays in several ways

Arrays of subscripted variables are powerful and useful tools.



A NEW KIND OF VARIABLE

You have used simple numeric variables and string variables.

- Numeric variables: A N TD KOLOR TEMPO ROW1
- String variables: A$ N$ NAME$ WORD$

Now you will meet and learn how to use a new type of variable called a *subscripted numeric variable*.

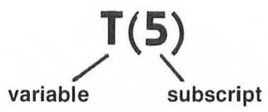
- Subscripted numeric variables: N(3) SP(7) COL(K)

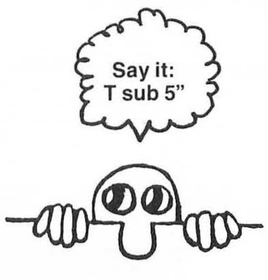
A subscripted variable consists of a variable name followed by a number enclosed in parentheses:

- A(3) is a subscripted variable
- A3 is not a subscripted variable. It is a simple numeric variable.
- But A3(2) *is* a subscripted variable.

A subscripted variable consists of:

a numeric variable name	T
left parenthesis	T(
a subscript	T(5
right parenthesis	T(5)





A subscript can be a number.

D(1)	"D sub 1"
ROW(6)	"ROW sub 6"

A subscript can be a numeric variable.

T(R)	"T sub R"
KOLOR(K)	"KOLOR sub K"
COL1(X)	"COL1 sub X"

A subscript can be a numeric expression.

T(N + 1)	"T sub N plus 1"
W(2*K − 1)	"W sub 2 times K minus 1"

A subscript can be a quite complicated numeric expression.

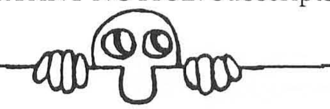
$$\text{TN}(\underbrace{\text{ASC(L\$)} - 64}_{\text{ATASCII code of L\$ minus 64}})$$

Now you know what subscripted variables look like. But what are they good for? Patience—we are slowly getting to it.

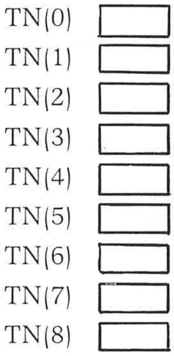
An *array* is a list of subscripted variables. Below are some arrays of subscripted variables.

- An array with three subscripted variables: T(0) T(1) T(2)
- An array with four subscripted variables: D(0) D(1) D(2) D(3)

IMPORTANT NOTICE! Subscripts are whole numbers 0, 1, 2, 3 and so on.



Once more, think of the computer's memory as a bunch of boxes, called number boxes (Chapter 4) and string boxes (Chapter 5). Now you also have arrays of *subscripted* number boxes. For example, here is an array of nine number boxes:



This is the array TN consisting of nine subscripted variables, TN(0) through TN(8).

Each number box (subscripted numeric variable) can hold one number. So lets put numbers in these boxes:

TN(0)	$\boxed{0}$
TN(1)	$\boxed{121}$
TN(2)	$\boxed{108}$
TN(3)	$\boxed{96}$
TN(4)	$\boxed{91}$
TN(5)	$\boxed{81}$
TN(6)	$\boxed{72}$
TN(7)	$\boxed{64}$
TN(8)	$\boxed{60}$

Do these numbers look familiar? Boxes TN(1) through TN(8) have values that are the tone numbers for Middle C, D, E, F, G, A, B, and C above Middle C. In the next section, we will use the array TN in a music program.

Questions

1. Circle each subscripted numeric variable.

   TN(0)   R2D2(C3P0)   N$(10)   K9   X(3*N − 2)
   Z(− 1)   A(L$)   TN(KEY − 48)

2. A small array: D(0) $\boxed{7}$ D(1) $\boxed{2}$ D(2) $\boxed{0}$

   (a) How many subscripted numeric variables does the array have? ____
   (b) What is the value of D(0)? ____
   (c) What variable has the value of 0? ____
   (d) What is the value of D(0) + D(1)? ____

3. An array called ROW has six subscripted variables, beginning with ROW(0). Write the names of all subscripted variables in the array called ROW.

Answers

1. These are subscripted numeric variables: TN(0), R2D2(C3P0), X(3*N − 2), and TN(KEY − 48). What are the others? N$(10) is a *string* variable, K9 is a *simple* numeric variable, Z(− 1) and A(L$) are illegal.

2. (a) 3 (b) 7 (c) D(2) (d) Since the value of D(0) is 7 and the value of D(1) is 2, the value of D(0) + D(1) is 7 + 2 = 9.

3. ROW(0), ROW(1), ROW(2), ROW(3), ROW(4) and ROW(5).

KEYBOARD MUSIC

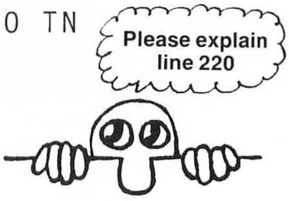
We will use the array TN to store tone numbers in a program to play music using the number keys 0 through 8. Before using an array, we must first DIMension it. So we begin this way:

```
100 REM** KEYBOARD MUSIC WITH ARRAY TN
110 DIM TN(8)
120 GR.0
```

Line 110 tells the computer to save space for nine subscripted variables, TN(0) through (TN8). This is similar to the way string variables are dimensioned. Remember, though, that for numeric subscripted variables, the first subscript is zero (0) instead of one (1).

Next, we want to put tone numbers in TN(0) through TN(8). We'll use READ and DATA in a FOR-NEXT loop.

```
200 REM** READ TONE NUMBERS INTO TN
210 FOR K=0 TO 8
220   READ TN: TN(K)=TN
230 NEXT K
240 DATA 0, 121, 108, 96
250 DATA 91, 81, 72, 64, 60
```



OK. Line 220 tells the computer to read a value for the *simple* numeric variable TN, then assign the value to the *subscripted* variable TN(K). It seems that, in ATARI BASIC, you can't read a value directly into a subscripted numeric variable. Try the following to see what happens:

```
220 READ TN(K)
```

The tone numbers are now stored, with TN(0)=0, TN(1)=121 and so on. The next three blocks are similar to the KEYBOARD MUSIC USING GET program in chapter 10.

```
300 REM**OPEN CHANNEL TO KEYBOARD
310 OPEN #1,4,0,"K:"

400 REM**TELL WHAT TO DO
410 PR. "PRESS ANY KEY, 1 TO 8, OR"
420 PR. "PRESS ZERO (0) FOR SILENCE."
```

```
500 REM**GET & VERIFY A KEY
510 GET #1, KEY
520 IF KEY<48 OR KEY>55 THEN 510
```

To get past block 500, someone must press a number key from zero (0) to eight (8). These keys have ATASCII codes from 48(for zero) to 55(for eight). Aha! We now use the ATASCII code to pick up the tone number in the array TN.

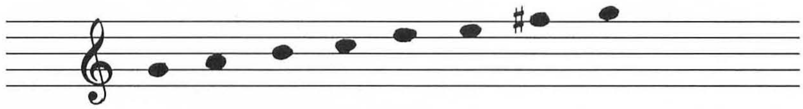
```
600 REM**PLAY THE NOTE
610 FOR L=10 TO 0 STEP -1
620   SO. 0,TN(KEY-48), 10, L
630   TD = 10
640   FOR Z=1 TO TD: NEXT Z
650 NEXT L
660 GOTO 510
```

The sound fades away since loudness (L) goes from 10 to 0. Try other values of TD

For example, if someone presses the 4 key, the value of KEY in line 510 will be 52. So, in line 620, TN(KEY – 48) will be TN(52 – 48) = TN(4) = 91. Remember, since the values of KEY can be 48 to 55, the values of KEY – 48 will be 0 to 8, the subscripts in the array TN!

Questions

1. How would you change the program to play music in the scale of G? Here are the letter notes, tone numbers, and number keys.



Letter note	G	A	B	C	D	E	F#	G
Tone number	81	72	64	60	53	47	42	40
Number key	1	2	3	4	5	6	7	8

2. Modify our program to add one more note, played by pressing the nine (9) key. In our scale of C, add the tone number 53 as the value of TN(9).

Answers

1. Change only the DATA statements, as follows:

```
240 DATA 0, 81, 72, 64
250 DATA 60, 53, 47, 42, 40
```

2. Make these changes:

```
110 DIM TN(90)
210 FOR K=0 TO 9
250 DATA 91, 81, 72, 64, 60, 53
```

IT'S 1984 AND WE CAN STILL VOTE!

Election day is coming and the pollsters are everywhere, sampling public opinion. Jason decided to run his own poll. So he asked his friends their opinions.

> Who will you vote for in the coming election? Circle the number to the left of your choice.
>
> 1. Sam Smoothe
> 2. Gabby Gruff

Let's write a program to count the votes each candidate received in the poll. You have 32 responses to your questionnaire, each response being either a "1" or a "2". First, record the votes in DATA statements.

```
900 REM**VOTES & END-OF-DATA FLAG(-1)
910 DATA 1, 1, 2, 2, 2, 1, 1, 2, 2, 2, 1, 1, 1, 2
920 DATA 1, 2, 1, 1, 2, 2, 1, 1, 1, 2, 1, 2, 2, 2
930 DATA 1, 1, 2, 1, -1
```

End-of-data flag (not a vote)

How many votes did Sam Smoothe receive? How many votes did Gabby Gruff get? To answer those last two questions, you probably counted the 1's in the DATA statements to find out how many votes Sam Smoothe received. Then you counted the 2's to find out how many votes Gabby Gruff received.

Your computer can count votes by using subscripted variables to keep a running total of the 1's and 2's read from the DATA statements. When it comes to the end-of-data flag (−1) it stops counting and prints the results. Let's write a program to the count the votes. We'll use C(1) to count Sam's votes and C(2) to count Gabby's votes.

Sam's votes:	C(1)	□
Gabby's votes:	C(2)	□

We begin by DIMensioning the C array and starting the vote count at zero.

```
100 REM**VOTE COUNTING PROGRAM
110 DIM C(2)
120 GR. 0

200 REM**SET COUNTS TO ZERO
210 C(1) = 0
220 C(2) = 0
```

Next, read one vote (V). If V is the end-of-data flag (−1), quit counting and go print the results. Otherwise, count the vote.

```
300 REM**READ A VOTE & CHECK FOR FLAG
310 READ V
320 IF V=-1 THEN 510

400 REM**COUNT VOTE & GO BACK
410 C(V) = C(V) + 1
420 GOTO 310
```

Crucial vote-counting statement

Look again at the crucial vote-counting statement:

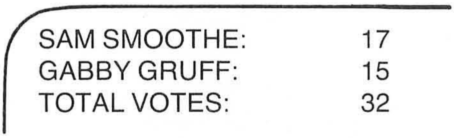
410 C(V) = C(V)+1

Remember, a vote (V) can be only one (1) or two (2). If V is 1, line 410 tells the computer to add one to the value of C(1). If V is 2, line 410 tells the computer to add one to the value of C(2).

Eventually, the computer will read the end-of-data flag (−1) and go to line 510.

```
500 REM**PRINT THE RESULTS
510 PR. "SAM SMOOTHE: "; C(1)
520 PR. "GABBY GRUFF: "; C(2)
530 PR. "TOTAL VOTES: "; C(1) + C(2)
540 END
```

We entered the entire program, including the DATA statements, and ran it. This is what we saw.



```
SAM SMOOTHE:          17
GABBY GRUFF:          15
TOTAL VOTES:          32
```

Questions

1. The first five votes in line 910 are 1, 1, 2, 2, and 2. Show the values of C(1) and C(2) after each of these votes has been read and added:

   1st vote C(1) □ C(2) □
   2nd vote C(1) □ C(2) □
   3rd vote C(1) □ C(2) □
   4th vote C(1) □ C(2) □
   5th vote C(1) □ C(2) □

2. Suppose the following poll is conducted:

> Which candidate will you vote for in the coming election? Circle the number to the left of your choice.
>
> 1. Sam Smoothe
> 2. Gabby Gruff
> 3. No opinion

The results of this poll are shown below:

```
2, 2, 2, 1, 2, 1, 1, 2, 1, 1, 3, 2, 1, 3
2, 1, 1, 3, 1, 3, 2, 2, 1, 1, 3, 2, 1, 3
1, 1, 2, 1, 2, 1, 1
```

Modify the VOTE COUNTING program to count these answers. A run should look like this:

```
SAM SMOOTHE:          17
GABBY GRUFF:          12
NO OPINION:           6
TOTAL VOTES:          35
```

Answers

1. 1st vote C(1) $\boxed{1}$ C(2) $\boxed{0}$

   2nd vote C(1) $\boxed{2}$ C(2) $\boxed{0}$

   3rd vote C(1) $\boxed{2}$ C(2) $\boxed{1}$

   4th vote C(1) $\boxed{2}$ C(2) $\boxed{2}$

   5th vote C(1) $\boxed{2}$ C(2) $\boxed{3}$

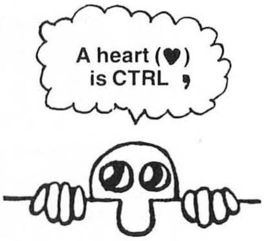
2. Make these additions and changes:

```
110 DIM C(3)
230 C(3)=0
530 PR."NO OPINION:";C(3)
540 PR."TOTAL VOTES:";C(1)+C(2)+C(3)
550 END
910 DATA2,2,2,1,2,1,1,2,1,1,3,2,1,3
920 DATA2,1,1,3,1,3,2,2,1,1,3,2,1,3
930 DATA1,1,2,1,2,1,1,-1
```

TWINKLING STARS

In a galaxy far away, where peace and love prevail, the stars are shaped like hearts. On a dark night, you can see a myriad of heart-shaped stars gently twinkling.

```
100 REM**TWINKLING STARS
110 DIM COL(100),ROW(100)
120 GR. 0
130 POKE 752,1

200 REM**RANDOM STAR LOCATIONS
210 NSTARS = 13
220 FOR STAR=1 TO NSTARS
230   COL(STAR) = INT(40*RND(0))
240   ROW(STAR) = INT(23*RND(0))
250 NEXT STAR

300 REM**DARK SKY, BRIGHT STARS
310 SE. 2,0,0
320 SE. 1,0,14

400 REM**PUT STARS IN THE SKY
410 FOR STAR=1 TO NSTARS
420   POS. COL(STAR),ROW(STAR)
430   PR. "♥";
440 NEXT STAR

500 REM**TWINKLE A RANDOM STAR
510 STAR = INT(NSTARS*RND(0))+1
520 POS. COL(STAR),ROW(STAR): PR. " ";
530 GOSUB 910
540 POS. COL(STAR),ROW(STAR): PR. " ";

600 REM**GO TWINKLE ANOTHER
610 GOTO 510

900 REM**TIME DELAY SUBROUTINE
910 TD = 20
920 FOR Z=1 TO TD: NEXT Z
930 RETURN
```



Enter and run the program. You should see 13 stars appear on a black screen. Each star is heart-shaped. To type a heart, hold down the CTRL key and press the comma (,) key.

Each star has a screen position consisting of a column and row. Star #1 is a COL(1), ROW(1); star #2 is at COL(2), ROW(2); star #3 is at COL(3), ROW(3); and so on. We don't use COL(0) and ROW(0) in this program.

Block 200 tells the computer to assign random screen positions to NSTARS stars. We set the value of NSTARS to 13 (line 210). You can change this to get more stars or fewer stars.

Block 400 puts the stars in the sky. Remember to include the semicolon (;) at the end of line 430.

Block 500 selects a random star (1 to NSTARS) and "twinkles" it. In line 510, the value of STAR will be an integer from 1 to NSTARS. Suppose STAR is 7. Then line 520 tells the computer to put a space at COL(7), ROW(7), thus "turning off" star #7. After a time delay, line 540 turns the star back on again.

Change the time delay in line 910 to slow down or speed up the action.

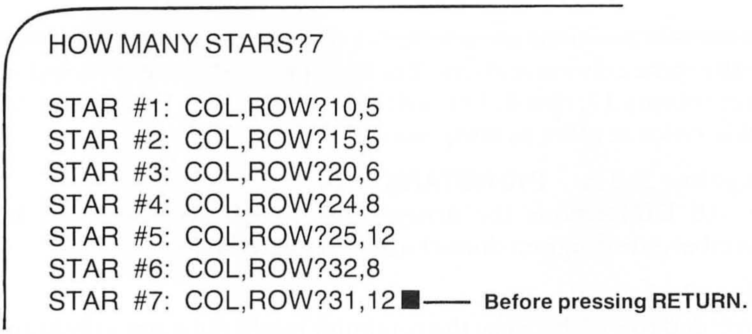
Instead of stars in random screen locations, perhaps you would like to put your favorite constellation on the screen: the Big Dipper? Draco? Orion? Or perhaps you want to invent your own constellation. You can do so with the following program:

```
100 REM**TWINKLING STARS WITH INPUT
110 DIM COL(100) ,ROW(100)
120 GR. 0

200 REM**GET STAR LOCATIONS
210 PR. "HOW MANY STARS";: IN. NSTARS
220 PR.
230 FOR STAR=1 TO NSTARS
240   PR. "STAR #";STAR;": COL,ROW";: IN. C,R
250   COL(STAR) = C: ROW(STAR) = R
260 NEXT STAR

300 REM**DARK SKY, BRIGHT STARS
310 SE. 2,0,0
320 SE. 1,0,14
330 POKE 752,1

400 REM**PUT STARS IN THE SKY
410 PR. CHR$(125)
420 FOR STAR=1 TO NSTARS
430   POS. C(STAR),ROW(STAR)
440   PR. "*";
450 NEXT STAR
```

(continued)

```
500 REM**TWINKLE THE STARS
510 STAR = INT(INSTARS*RND(0))+1
520 POS. COL(STAR),ROW(STAR): PR. " ";
530 GOSUB 910
540 POS. COL(STAR),ROW(STAR): PR. "*";
550 GOTO 510

900 REM**TIME DELAY SUBROUTINE
910 TD = 20
920 FOR Z=1 TO TD: NEXT Z
930 RETURN
```

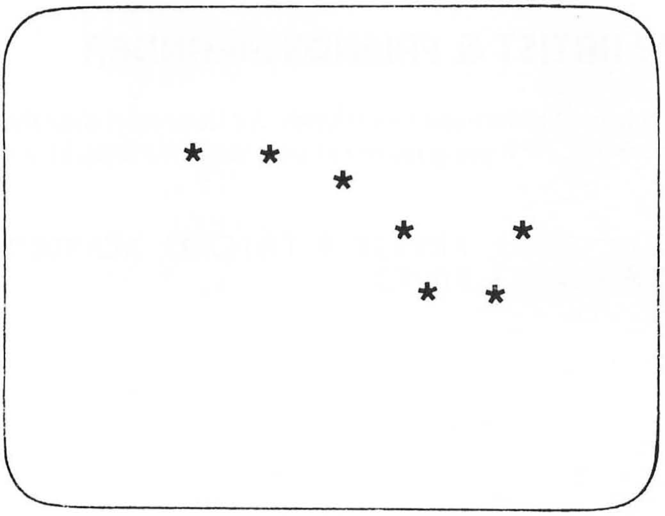
OK, let's try it.



```
HOW MANY STARS?7

STAR #1: COL,ROW?10,5
STAR #2: COL,ROW?15,5
STAR #3: COL,ROW?20,6
STAR #4: COL,ROW?24,8
STAR #5: COL,ROW?25,12
STAR #6: COL,ROW?32,8
STAR #7: COL,ROW?31,12■
```

Before pressing RETURN.

We entered the number of stars and the column and row for each star. When we pressed RETURN the last time, here is what we saw:



Questions

1. One time we ran the TWINKLING STARS program and counted only 12 stars. How could this happen?__________
2. How would you change the program to put 24 stars in the sky?__________
3. In lines 230 and 240, what are the possible values of

   (a) COL(STAR)__________

   (b) ROW(STAR)__________

Answers

1. Since the star positions are selected at random, it is possible for two stars to have the same column and row. For example, perhaps star #4 and star #9 are both at column 17, row 8. You will see only one star. However, it is likely to twinkle twice as often as other stars.
2. Change line 210 to: `210 NSTARS = 24`
   Line 110 DIMensions the arrays COL and ROW for up to 100 stars. Remember, the program doesn't use COL(0), ROW(0).
3. (a) 0 to 39 (b) 0 to 22.
   We avoided row 23 because the computer might put a star at column 39, row 23, thus causing the screen display to scroll up one line.

ZAPPY ARTIST & FRIENDS MEANDER

Zappy Artist is back and has brought two friends. All three start near the center of the screen and meander, each going his or her own way. We introduce Zappy and two friends.

```
100 REM**ZAPPY ARTIST & FRIENDS MEANDER
110 DIM COL(3),ROW(3)
120 GR. 19
```

```
300 REM**ZAPPY & FRIENDS APPEAR
310 FOR K=1 TO 3
320   COL(K) = 17+K
330   ROW(K) = 12
340   COLOR K
350   PLOT COL(K),ROW(K)
360 NEXT K

400 REM**THEY ALL MEANDER
410 FOR K=1 TO 3
420   GOSUB 610
430   COLOR K
440   PLOT COL(K),ROW(K)
450 NEXT K

500 REM**TIME DELAY, THEN MEANDER MORE
510 TD = 100
520 FOR Z=1 TO TD: NEXT Z
530 GOTO 410

600 REM**CHOOSE A DIRECTION SUBROUTINE
610 WAY = INT(4*RND(0))+1
620 IF WAY=1 THEN COL(K)=COL(K)+1
630 IF WAY=2 THEN ROW(K)=ROW(K)+1
640 IF WAY=3 THEN COL(K)=COL(K)-1
650 IF WAY=4 THEN ROW(K)=ROW(K)-1
660 IF COL(K)>39 THEN COL(K)=39
670 IF ROW(K)>23 THEN ROW(K)=23
680 IF COL(K)<0  THEN COL(K)=0
690 IF ROW(K)<0  THEN ROW(K)=0

700 RETURN
```

Questions

1. Look at block 300. Artists 1, 2 and 3 appear. Where will each artist appear?

   (a) Artist #1 (K=1) appears in column ____, row ____.

   (b) Artist #2 (K=2) appears in column ____, row ____.

   (c) Artist #3 (K=3) appears in column ____, row ____.

2. In the CHOOSE A DIRECTION SUBROUTINE, what is the function of lines 660 through 690?

   ____

3. Change the program so it works in GRAPHICS 21.

Answers

1. (a) COL(1) is 18 and ROW(1) is 12.

   (b) COL(2) is 19 and ROW(2) is 12.

   (c) COL(3) is 20 and ROW(3) is 12.

2. These lines prevent any artist from trying to move "off the screen." Without these lines, COL(K) might become less than zero or greater than 39 or ROW(K) might become less than zero or greater than 23. Either case would cause the computer to stop with an error message.

3. We'll leave this for you. Change lines 120, 320, 330, 660, 670, 680, and 690.

DOUBLE SUBSCRIPTS

So far you have used subscripted numeric variables with single subscripts. Each variable has *one* subscript.

```
COL(K)          TN(KEY-47)          C(V)
```

You can also use subscripted numeric variables with *two* subscripts.

C(2,3)

Two subscripts, separated by a comma.

Suppose it is election time again. Jason, having learned more about poll-taking, is going door to door with the following questionnaire:

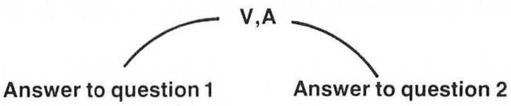
Q1. Who will you vote for in the coming election?
Circle the number to the left of your choice.

1. Sam Smoothe
2. Gabby Gruff
3. No opinion

Q2. What age group are you in? Circle the number to the left of your age group.

1. Under 30
2. 30 or over

Since there are two questions, each reply consists of two numbers: the answer to question 1 and the answer to question 2. We will use V (for Vote) to represent the answer to question 1 and A (for Age) to represent the answer to question 2.



The possible values of V are 1, 2 or 3. The possible values of A are 1 or 2.
Here are some possible responses:

Reply	Meaning
1,1	one vote for Sam Smoothe, voter is under 30
1,2	one vote for Sam Smoothe, voter is 30 or over
3,1	no opinion, voter is under 30

We want to write a program to summarize data for a two-question questionnaire. We will use subscripted variables to count votes as shown:

		Under 30		30 or over
Sam Smoothe	C(1,1)	□	C(1,2)	□
Gabby Gruff	C(2,1)	□	C(1,2)	□
No Opinion	C(3,1)	□	C(3,2)	□

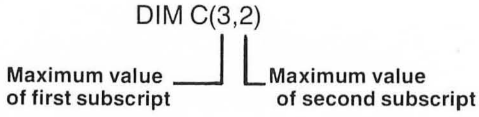
Here are 29 replies to Jason's questionnaire. Remember, each reply is a *pair* of numbers and represents *one* vote. The first number of each pair is the answer to question 1. The second number of each pair is the answer to question 2.

3,1	2,2	3,2	1,2	1,2	2,1
2,2	1,1	1,2	3,1	3,2	2,2
3,1	2,1	2,2	1,1	1,1	1,2
1,1	2,1	2,1	1,2	2,1	3,1
2,1	3,1	2,1	3,1	2,2	

Naturally, we want the computer to do the counting. Below is the beginning of our program:

```
100 REM**VOTE COUNTING, 2 DIMENSIONAL
110 DIM C(3,2)
120 GR. 0
```

Line 110 tells the computer to reserve memory space for the array C which has doubly-subscripted variables. The first subscript can be 0,1,2, or 3; the second subscript can be 0,1, or 2.


DIM C(3,2)
Maximum value of first subscript
Maximum value of second subscript

Next, let's set all the counts to zero. We won't be using the zero subscripts, so here are *three* ways to write block 200.

```
(1) 200 REM**SET COUNTS TO ZERO
    210 C(1,1) = 0
    220 C(1,2) = 0
    230 C(2,1) = 0
    240 C(2,2) = 0
    250 C(3,1) = 0
    260 C(3,2) = 0
```

```
(2) 200 REM**SET COUNTS TO ZERO
    210 FOR I=1 TO 3
    220   C(I,1) = 0
    230   C(I,2) = 0
    240 NEXT I
```

```
(3) 200 REM**SET COUNTS TO ZERO
    210 FOR I=1 TO 3
    220   FOR J=1 TO 2
    230     C(I,J) = 0
    240   NEXT J
    250 NEXT I
```

We prefer method (3) because it can easily be modified for different sizes of arrays (number of subscripts).

The array is now set up. Let's READ and count the votes:

```
300 REM**READ & COUNT VOTES
310 READ V,A: IF V=-1 THEN 510
320 C(V,A) = C(V,A) + 1: GOTO 310
```

Well, that was easy! Line 310 tells the computer to read two numbers and check for an end-of-data flag. If the flag is detected (V is −1), the computer goes to line 510, which you will soon see. Otherwise, the computer uses the values of V and A to select the count and increase by one, then goes back and reads more data.

Since line 310 is a READ statement, some DATA statements must be given somewhere. Here they are, including two end-of-data flags:

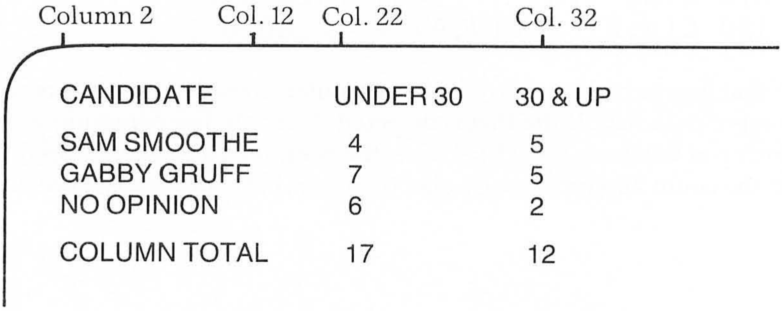
```
900 REM**DATA IN PAIRS:
901 REM**VOTE FOLLOWED BY AGE GROUP
910 DATA 3,1, 2,2, 3,2, 1,2, 1,2, 2,1
920 DATA 2,2, 1,1, 1,2, 3,1, 3,2, 2,2
930 DATA 3,1, 2,1, 2,2, 1,1, 1,1, 1,2
940 DATA 1,1, 2,1, 2,1, 1,2, 2,1, 3,1
950 DATA 2,1, 3,1, 2,1, 3,1, 2,2, -1,-1
```

Remember, each reply is a *pair* of numbers representing *one* vote. To emphasize this, we have typed a space after each reply (pair of values) in the DATA statements above. We need two end-of-data flags (– 1, – 1) because line 310 insists on reading two numbers (V,A).

One thing left to do — block 500 to print the results.

```
500 REM**SHOW THE RESULTS
510 PR. CHR$(125)
520 PR. "CANDIDATE","UNDER 30","30 & UP"
530 PR.
540 PR. "SAM SMOOTHE",C(1,1),C(1,2)
550 PR. "GABBY GRUFF",C(2,1),C(2,2)
560 PR. "NO OPINION", C(3,1),C(3,2)
570 PR.
580 PR. "TOTAL VOTES",
590 PR. C(1,1)+C(2,1)+C(3,1),C(1,2)+C(2,2)+C(3,2)
999 END
```

When we ran the program, this is what we saw:

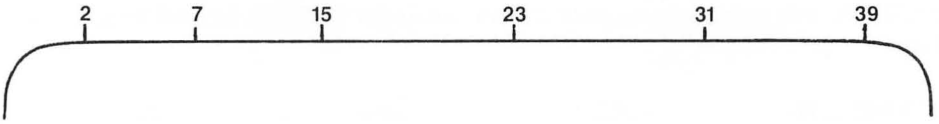


```
 CANDIDATE           UNDER 30  30 & UP

 SAM SMOOTHE          4         5
 GABBY GRUFF          7         5
 NO OPINION           6         2

 COLUMN TOTAL         17        12
```

"OK," we thought, "but we would also like the line total for each candidate to appear." However, we ran out of space. So let's use another nifty feature of ATARI BASIC: the TAB feature:

- Clear the screen. As usual, the cursor appears in column 2.
- Press the TAB key: CLR SET TAB The cursor moves to column 7.
- Press the TAB key. The cursor moves to column 15.
- Press the TAB key a few more times to see what happens.

Here are the standard TAB positions:



You can use these TAB positions in a PRINT statement to space information on the screen. To include a TAB command:

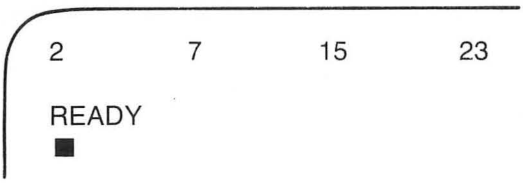
1) Use CHR$(127) to tell the computer to move the cursor to the *next* TAB position, or

2) Use "►" as a string in the PRINT statement. To type ►, first press ESC, then TAB.

Try this:

Type: PR. CHR$(125);2;"► ";7;"► ";15;"► ";23

This is what you will see:



```
2    7       15      23

READY
■
```

Before we apply the TAB feature to printing the results of the poll, let's write block 400 to compute some totals. But first, rewrite block 200 so *all* the members of array C are set to zero.

```
200 REM**SET COUNTS TO ZERO
210 FOR I=0 TO 3
220   FOR J=0 TO 2
230     C(I,J) = 0
240   NEXT J
250 NEXT I
```

I and J both begin at zero.

In block 400, we are going to compute *line totals* for Sam and Gabby and NO OPINION. We will also compute *column* totals for UNDER 30 and 30 & UP, like this.

CANDIDATE	<30	30+	TOTAL
SAM SMOOTHE	C(1,1)	C(1,2)	C(1,0)
GABBY GRUFF	C(2,1)	C(2,2)	C(2,0)
NO OPINION	C(3,1)	C(3,2)	C(3,0)
COLUMN TOTALS	C(0,1)	C(0,2)	C(0,0)

OK, here we go!

```
400 REM**COMPUTE TOTALS
410 FOR I=1 TO 3
420   C(I,0) = C(I,1)+C(I,2)
430 NEXT I
440 FOR J=0 TO 2
450   C(0,J) = C(1,J)+C(2,J)+C(3,J)
460 NEXT J
```

Line totals

Column totals

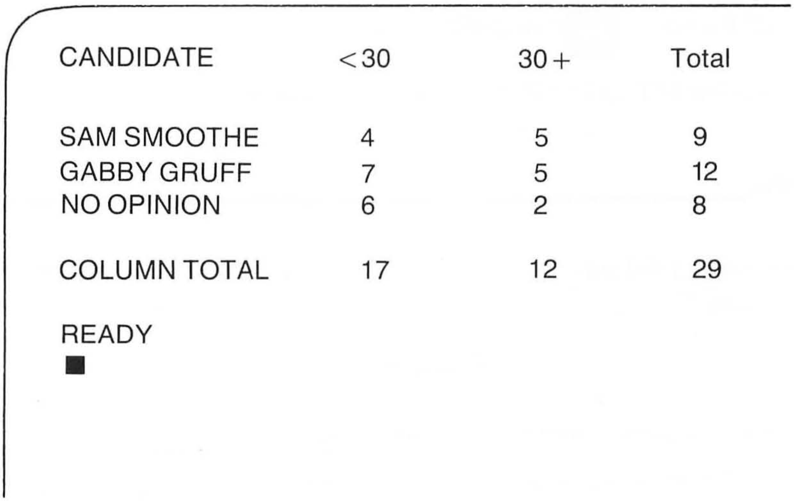
Now we print all the information. Note how we use the TAB feature ("►") to position information on the screen.

```
500 REM**PRINT THE RESULTS
510 PR. CHR$(125)
520 PR. "CANDIDATE";"►";"<30";"►";"30+";"►";"TOTAL"
530 PR.
540 PR. "SAM SMOOTHE";"►";C(1,1);"►";C(1,2);"►";C(1,0)
550 PR. "GABBY GRUFF:;"►";C(2,1);"►";C(2,2);"►";C(2,0)
560 PR. "NO OPINION ";"►";C(3,1);"►";C(3,2);"►";C(3,0)
570 PR.
580 PR. "COLUMN TOTAL";"►";C(0,1);"►";C(0,2);"►";C(0,0)

999 END
```

One more thing. Rewrite line 310 as follows:

```
310 READ V,A: IF V=-1 THEN 410
```

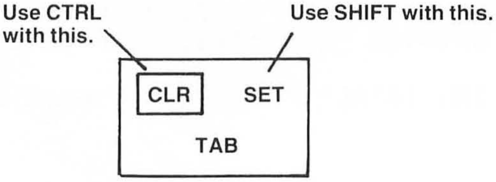
Having made all the above changes, we ran the program.



```
CANDIDATE          <30        30+        Total

SAM SMOOTHE         4          5           9
GABBY GRUFF         7          5          12
NO OPINION          6          2           8

COLUMN TOTAL       17         12          29

READY
■
```

Remember, the TAB ("►") causes the cursor to move to the *next* tab position. In printing CANDIDATE, the cursor moves past the tab stop at column 7. So the first "►" in line 520 causes the cursor to move to column 15. This also happens in lines 540, 550, 560 and 580.

Yes, you can clear the tab stops at columns 2, 7, 15, 23, 31 and 39; and you can put tab stops where *you* want them. Look at the TAB key.



- To set a tab stop, position the cursor to the column where you want to set a tab stop, then hold down SHIFT and press SET.
- To clear a tab stop, position the cursor to the column where you want to clear a tab stop, then hold down CTRL and press CLR.

In a PRINT statement, you can TAB, SET a tab stop or CLR a tab stop. First type quotation marks, then press ESC, then press TAB or CTRL/CLR or SHIFT/SET, then type quotation marks. Or, you can use the CHR$ function.

- To TAB: "►" or CHR$(127)
- To SET a tab: "➡" or CHR$(159)
- To CLR a tab: "⬅" or CHR$(158)

The symbols for SET and CLR are inverse video arrows.

Questions

1. The following DIM statement tells the computer to reserve room in memory for the array C.

DIM C(3,2)

(a) How many subscripted variables are in this array?________

(b) Write the names of all variables in the array.________________________

2. What does the statement: C(V,A) = C(V,A) + 1 tell the computer to do?_____

____

3. Explain how block 400 (COMPUTE TOTALS) works.

____

____

Answers

1. (a) 12 (b) C(0,0), C(0,1), C(0,2), C(1,0), C(1,1), C(1,2), C(2,0), C(2,1), C(2,2), C(3,0), C(3,1), C(3,2).
2. Add one to the value of C(V,A) and put the new result in C(V,A).
3. Lines 410 through 430 compute C(I,0) = C(I,1) + C(I,2) for I equal to 1, 2 and 3. These are the line totals for Sam Smoothe (I = 1), Gabby Gruff (I = 2) and NO OPINION (I = 3). Lines 440 through 460 compute C(0,J) = C(1,J) + C(2,J) + C(3,J) for J = 0,1 and 2. These are the column totals for <30 (J = 1), 30+ (J = 2) and TOTAL (J = 0).

QUIZ SCORES

In a small class of eight students, each student has taken four quizzes. Here are the scores:

	QUIZ 1	QUIZ 2	QUIZ 3	QUIZ 4
Student 1	65	57	71	75
Student 2	80	90	91	88
Student 3	78	82	77	86
Student 4	45	38	44	46
Student 5	83	82	79	85
Student 6	70	68	83	59
Student 7	98	92	100	97
Student 8	85	73	80	77

Use S to mean student and Q for Quiz. Then SCORE(S,Q) is the score of student S on Quiz Q. For example, SCORE(1,1) is 65; SCORE(4,2) is 38; SCORE (7,4) is 97.

Another class might have 30 students and 5 quizzes per student. Still another class might have 23 students and 7 quizzes per student, and so on. Let's begin a program to read scores for NS students and NQ quizzes per student.

```
100 100 REM**QUIZ SCORES
110 DIM SCORE(40,20)
120 GR. 0
```

Now let's read the array of scores.

```
200 REM**READ NS STUDENTS, NQ QUIZZES
210 READ NS,NQ
220 FOR S=1 TO NS
230   FOR Q=1 TO NQ
240     READ SCORE: SCORE(S,Q) = SCORE
250   NEXT Q
260 NEXT S

900 REM**NS,NQ,AND SCORES
910 DATA 8,4
920 DATA 65,57,71,75
930 DATA 80,90,91,88
940 DATA 78,82,77,86
950 DATA 45,38,44,46
960 DATA 83,82,79,85
970 DATA 70,68,83,59
980 DATA 98,92,100,97
990 DATA 85,73,80,77
```

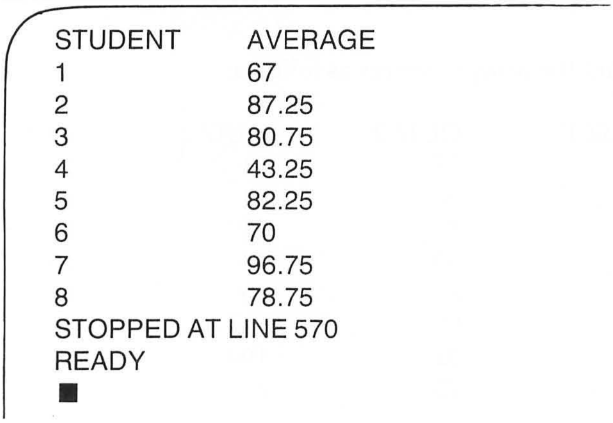
Now that we have the array in the computer, what shall we do with it? One thing someone might want is the average score for each student. Let's do it, beginning at line 400. We will put the average score in SCORE(S,0) for each student.

```
400 REM**COMPUTE STUDENT AVERAGES
410 FOR S=1 TO NS
420   TS = 0
430   FOR Q=1 TO NQ
440     TS = TS + SCORE(S,Q)
450   NEXT Q
460   SCORE(S,0) = TS/NQ
470 NEXT S
```

(continued)

```
500 REM**PRINT AVERAGES
510 PR. CHR$(125)
520 PR. "STUDENT","AVERAGE"
530 PR.
540 FOR S=1 TO NS
550   PR. S,SCORE(S,0)
560 NEXT S
570 STOP
```

Enter the program and run it. This is what you should see:



```
STUDENT      AVERAGE
1            67
2            87.25
3            80.75
4            43.25
5            82.25
6            70
7            96.75
8            78.75
STOPPED AT LINE 570
READY
■
```

If there were three or six quizzes, there might be several decimal places in the average. For example: 67.33333333. Here is a way to round a number to an integer or to one or two or three decimal places.

- Round the value of X to the nearest integer.

  INT(X+.5)

- Round the value of X to the nearest tenth.

  INT(10*X+.5)/10

- Round X to the nearest hundreth.

  INT(100*X+.5)/100

The above works whenever the value of X is zero or positive.

EXPERIMENT! Try this program:

```
10 PR. CHR$(125)
20 PR. "NUMBER PLEASE";:IN. X
30 PR. "NEAREST INTEGER:", INT(X+.5)
40 PR. "NEAREST TENTH:",INT(10*X+.5)/10
50 PR. "NEAREST HUNDREDTH:",INT(100*X+.5)/100
60 PR.: GOTO 20
```

Questions

1. Write block 300 to print the array of scores as follows:

```
STUDENT #   QUIZ 1      QUIZ 2      QUIZ 3      QUIZ 4
1           65          57          71          75
2           80          90          91          88
3           78          82          77          86
4           45          38          44          46
5           83          82          79          85
6           70          68          83          59
7           98          92          100         97
8           85          73          80          77
```

2. Write blocks 600 and 700 to compute and print the average score for each quiz. For the data used in the program, the results should look like these:

```
QUIZ #        AVERAGE
1             75.5
2             72.75
3             78.125
4             76.625
```

Answers

1.

```
200 REM**PRINT SCORES
210 PR. CHR$(125)
220 PR. "STUDENT #";              ]
230 FOR Q=1 TO NQ                 ]— Print headings.
235   PR. "►";"QUIZ ";Q;          ]
240 NEXT Q
245 PR.: PR.
250 FOR S=1 TO NS
260   PR. S;"►"; ———— Print student number
                      and TAB.
265   FOR Q=1 TO NQ
270     PR. "►";SCORE(S,Q);——— TAB and print score.
275   NEXT Q
280 NEXT S
285 PR.
290 STOP
```

When you run the program, it will print the scores and stop at line 290. To move on to line 300, type CONT and press RETURN.

2. We put the average for quiz Q in SCORE(0,Q).

```
600 REM**COMPUTE QUIZ AVERAGES
610 FOR Q=1 TO NQ
620   TQ = 0
630   FOR S=1 TO NS
640     TQ = TQ + SCORE(S,Q)
650   NEXT S
660   SCORE(0,Q) = TQ/NS
670 NEXT Q
680 STOP

700 REM**PRINT QUIZ AVERAGES
710 PR. CHR$(125)
720 PR. "QUIZ #","AVERAGE"
730 PR.
740 FOR Q=1 TO NQ
750   PR. Q,SCORE(0,Q)
760 NEXT Q
770 STOP
```

Now run the complete program, blocks 100, 200, 300, 400, 500, 600, 700 and 900. First, the computer reads the scores, prints them on the screen and stops. Type CONT and press RETURN. The computer computes and prints the student averages, then stops. Type CONT and press RETURN. The computer computes and prints the quiz averages, then stops.

SELF-TEST

You are nearing the end of this adventure. You have acquired skill, knowledge, and experience points. So plunge into another Self-Test with confidence — nothing can go array.

1. Classify each variable as a numeric variable (NV), string variable (SV), subscripted numeric variable (SNV) or not a legal variable (OOPS).

   (a) X ____ (b) X$ ____
   (c) $X ____ (d) X(3) ____
   (e) UP7 ____ (f) 7UP ____
   (g) SEVENUP ____ (h) C(I,J) ____

2. Write a single DIM statement to reserve memory space for the numeric array N with variables N(0) up to N(100), the array T with variables T(0,0) to T(7,5), and the string variable S$ with up to 50 characters.

   DIM ______________________

3. Write the names of all subscripted variables that belong to the arrays defined by the following DIM statements.

   (a) DIM ARRAY(3)

   The variables are ______________________

   (b) DIM B(1,2)

   The variables are ______________________

4. For you people with lazy fingers or no coins, we ask you to complete the following program to flip up to 100 "coins" and count the number of heads (H) and tails (T). Show each flip as H and space or T and space. After the coin has been flipped the desired number of times, show the total number of heads and tails.

```
100 REM**COIN FLIPPER & COUNTER
110 GR. 0
120 DIM COIN$(4),C(2),KY$(1)
130 COIN$ = "H T "

200 REM**TELL WHAT TO DO
210 PR. CHR$(125)
220 PR. "HOW MANY FLIPS (1 TO 100)";:IN. NF
230 IF NF<1 THEN 210
240 IF NF>100 THEN 210

300 REM**START COUNTS AT ZERO

310 C(1) = ____                 1 for heads (H)

320 ___________                 2 for tails (T)

400 REM**FLIP & COUNT
410 PR.
420 FOR FLIP=1 TO NF

430    _________________________

440    _________________________

450    __________________________

460 NEXT FLIP

500 REM**SHOW COUNTS
510 PR.: PR.
520 PR. C(1);"HEADS & ";C(2);" TAILS"

600 REM**TELL HOW TO DO AGAIN
610 PR.
620 PR. "TO DO AGAIN, PRESS 'RETURN'";
630 IN. KY$: GOTO 210
```

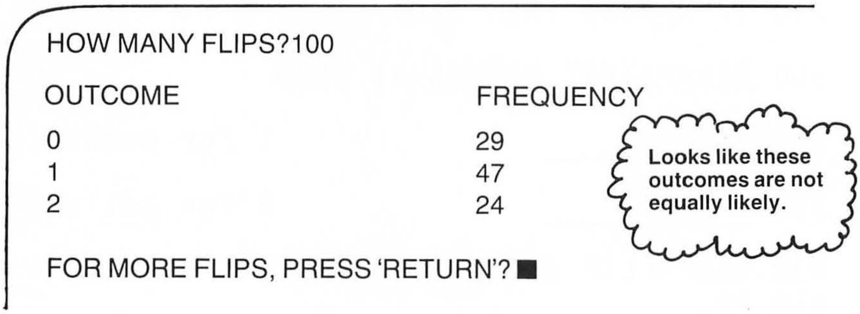
Print the flip
(H or T) and increase
the appropriate count.

5. Flip two coins. Let heads equal 0 and Tails equal 1. The outcome of a flip is the sum of these two numbers. Here are the possible outcomes:

FIRST COIN	SECOND COIN	OUTCOME
H	H	0
H	T	1
T	H	1
T	T	2

The possible outcomes are 0,1, or 2.

Write a program to flip two coins and count the outcomes. A RUN might look like this.



```
HOW MANY FLIPS?100

OUTCOME                         FREQUENCY

0                               29
1                               47
2                               24

FOR MORE FLIPS, PRESS 'RETURN'? ■
```

Looks like these outcomes are not equally likely.

The FREQUENCY is the number of times the OUTCOME occured.

6. Modify your program for question 5 so the computer rolls two dice and counts how many times each possible outcome occurred. The results might look as follows:

```
HOW MANY ROLLS?1000

OUTCOME                    FREQUENCY

2                          26
3                          59
4                          87
5                          106
6                          139
7                          170
8                          132
9                          112
10                         78
11                         60
12                         31

FOR MORE ROLLS, PRESS 'RETURN'? ■
```

7. How about three dice? Aha! We use three dice to roll characteristics for fantasy adventure characters. The results of running your program for a large number of rolls (try 1000) should tell you something about the probability, or likelihood, of rolling the various values from 3 to 18.

8. Write a GRAPHICS 19 program to put random tiny rectangles of light in random places on the screen. Make them all orange. Then select one at random and make it orange or light green or dark blue, at random. Keep doing this — use our TWINKLING STARS program as the model for this program.

9. In the game of Scrabble, each letter has a letter score as follows:

A=1	B=3	C=3	D=2	E=1	F=4
G=2	H=4	I=1	J=8	K=5	L=1
M=3	N=1	O=1	P=3	Q=10	R=1
S=1	T=1	U=1	V=4	W=4	X=8
Y=4	Z=10				

The Scrabble score of a word is the sum of the letter scores of the letters in the word. Write a program to compute the Scrabble score of a word, or any string of letters. Ignore anything that is not a letter. For example:

```
YOUR WORD?CAT
THE SCRABBLE SCORE IS 5

YOUR WORD?ISN'T
THE SCRABBLE SCORE IS 4

and so on.
```

Ignore the apostrophe.

10. The array CHORD consists of 10 subscripted variables, each with two subscripts. Each variable holds the tone numbers for a two-note chord to be played using voices 0 and 1.

CHORD # 1	VOICE 0	VOICE 1
0	CHORD(0,0)	CHORD(0,1)
1	CHORD(1,0)	CHORD(1,1)
2	CHORD(2,0)	CHORD(2,1)
:	:	:
9	CHORD(9,0)	CHORD(9,1)

We want a program to do two things. First, use a FOR-NEXT loop with INPUT to get 20 tone numbers and store them in the array CHORD. Then let someone play chords by pressing the number keys 0 through 9. Press zero (0) to play chord # 0, press one (1) to play chord # 1, and so on.

Answers to Self-Test

1. (a) X NV (b) X$ SV
(c) $X OOPS (d) X(3) SNV
(e) UP7 NV (f) 7UP OOPS
(g) SEVENUP NV (h) C(I,J) SNV

2. DIM N(100), T(7,5), S$(50)

3. (a) ARRAY(0), ARRAY(1), ARRAY(2), and ARRAY(3)

(b) B(0,0), B(0,1), B(0,2), B(1,0), B(1,1), and B(1,2)

4.

```
310 C(1) = 0
320 C(2) = 0

430    R = INT(2*RND(0))
440    PR. COIN$(2*R+1,2*R+2);
450    C(R+1) = C(R+1)+1
```

Lines 430-450 can also be written as follows.

```
430    R = INT(2*RND(0))+1
440    PR. COIN$(2*R-1,2*R);
450    C(R) = C(R) + 1
```

Perhaps you have yet a different way!

5.

```
100 REM**FLIP TWO COINS & COUNT
110 GR. 0
120 DIM C(2),KY$(1)

200 REM**TELL WHAT TO DO
210 PR. CHR$(125)
220 PR. "HOW MANY FLIPS";:IN. NF
230 IF NF<1 THEN 210

300 REM**START COUNTS AT ZERO
310 C(0) = 0
320 C(1) = 0
330 C(2) = 0
```

(continued)

```
400 REM**FLIP & COUNT
410 FOR K=1 TO NF
420   FC = INT(2*RND(0))      First coin
430   SC = INT(2*RND(0))      Second coin
440   SUM = FC + SC
450   C(SUM) = C(SUM) + 1
460 NEXT K

500 REM**SHOW COUNTS
510 PR.
520 PR. "OUTCOME","FREQUENCY"
530 PR.
540 FOR SUM=0 TO 2
550   PR. SUM,C(SUM)
560 NEXT SUM

600 REM**TELL HOW TO DO AGAIN
610 PR.
620 PR. "TO DO AGAIN, PRESS 'RETURN'";
630 IN. KY$: GOTO 210
```

6,7. We leave these for you to ponder.

8.

```
100 REM**STARS, CHANGING COLOR
110 GR. 19
120 DIM COL(100),ROW(100)

200 REM**RANDOM STAR LOCATIONS
210 NSTARS = 25
220 FOR STAR=1 TO NSTARS
230   COL(STAR) = INT(40*RND(0))
240   ROW(STAR) = INT(24*RND(0))
250 NEXT STAR

400 REM**PUT NSTARS IN SKY
400 FOR STAR=1 TO NSTARS
420   COLOR 1
430   PLOT COL(STAR),ROW(STAR)
440 NEXT STAR
```

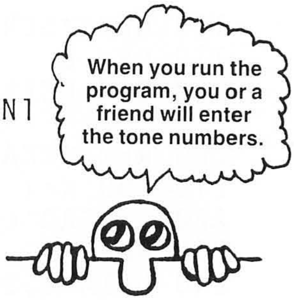
(continued)

```
500 REM**RANDOM STAR, RANDOM COLOR
510 STAR = INT(NSTARS*RND(0))+1
520 COLOR INT(3*RND(0))+1
530 PLOT COL(STAR),ROW(STAR)

600 REM**GO CHANGE ANOTHER
610 GOTO 510
```

9.

```
100 REM**SCRABBLE SCORES
110 GR. 0
120 DIM LS(26),WORD$(50),L$(1)

200 REM**READ LETTER SCORES INTO LS
210 FOR K=1 TO 26
220   READ LS: LS(K) = LS
230 NEXT K
240 DATA 1,3,3,2,1,4,2,4,1,8,5,1,3
250 DATA 1,1,3,10,1,1,1,1,4,4,8,4,10

300 REM**ASK FOR A WORD
310 PR "YOUR WORD";: IN. WORD$

400 REM**START WORD SCORE AT ZERO
410 WS = 0

500 REM**COMPUTE WORD SCORE
510 LW = LEN(WORD$)
520 FOR L=1 TO LW
530   L$ = WORD$(L,L)
540   IF L$<"A" THEN 580
550   IF L$>"Z" THEN 580
560   K = ASC(L$) - 64
570   WS = WS + LS(K)
580 NEXT L

600 REM**PRINT SCORE & GO AROUND
610 PR. "THE SCRABBLE SCORE IS ";WS
620 PR.: GOTO 310
```

10.
```
100 REM**KEYBOARD CHORDS
110 GR. 0
120 DIM CHORD(10,1)

200 REM**TELL WHAT TO DO
210 PR. "FOR EACH CHORD, TYPE TWO TONE"
220 PR. "NUMBERS, SEPARATED BY A COMMA,"
230 PR. "THEN PRESS 'RETURN'."

300 REM**GET THE TONE NUMBERS
310 PR.
320 FOR C=0 TO 9
330   PR. "CHORD #";C;:IN. TN0,TN1
340   CHORD(C,0) = TN0
350   CHORD(C,1) = TN1
360 NEXT C

400 REM**TELL HOW TO PLAY
410 PR.
420 PR. "TO PLAY, PRESS A NUMBER KEY."

500 REM**OPEN CHANNEL TO KEYBOARD
510 OPEN #1,4,0,"K:"

600 REM**GET & VERIFY A KEY
610 GET #1, KEY
620 IF KEY<48 OR KEY>56 THEN 610

700 REM**PLAY A CHORD & GO FOR MORE
710 C = KEY - 47
720 FOR L=10 TO 0  STEP -1
730   SO. 0,CHORD(C,0),10,L
740   SO. 1,CHORD(C,1),10,L
750   TD = 10
760   FOR Z=1 TO TD: NEXT Z
770 NEXT L
780 GOTO 610
```


When you run the program, you or a friend will enter the tone numbers.

You can easily modify this program to play chords with *three* voices.

Chapter Twelve

The End...of the Beginning

Look backwards — you have learned how to read and understand simple programs in ATARI BASIC. In the first 11 chapters of this beginner's book, we have shown you programs using the following ATARI BASIC *keywords*.

ABS
AND
ASC
CHR$
COLOR
CONT
DATA
DIM
DRAWTO
END
FOR
GET
GOSUB
GOTO
GRAPHICS
IF
INPUT
INT
LEN
LET
LIST
NEW
NEXT
ON
OPEN
OR
PLOT
POKE
POSITION
PRINT
READ
REM
RESTORE
RETURN
RND
RUN
SETCOLOR
SGN
SOUND
STEP
STOP
THEN
VAL

Use the Index to find places where the above keywords are used.

We used an *implied* LET to assign values to variables. For example:

```
ROW = 0
P$ = "POSITIVE"
```

These can also be written using an *explicit* LET.

```
LET ROW = 0
LET P$ = "POSITIVE"
```

Look forward — there is more to learn. There is more to learn about the keywords we covered in this book. There are also more keywords for you to learn. Here is a list of ATARI BASIC keywords we did not discuss in Chapters one through 11.

ADR
ATN
BYE
CLOAD
CLOG
CLOSE
CLR
COM
COS
CSAVE
DEG
DOS
ENTER
EXP
FRE
LOAD
LOCATE
LOG
LPRINT
NOT
NOTE
PEEK
POINT
POP
PTRIG
PUT
RAD
SAVE
SIN
SQR
STATUS
STICK
STRIG
STR$
TRAP
USR
XIO

Browse through the Appendices. You will find information on some, but not all, of the above keywords. Also consult the reference manual that came with your ATARI computer:

ATARI 400/900 BASIC REFERENCE MANUAL.
ATARI part number C015307 Rev. 1

ATARI BASIC Reference Guide for Experienced Programmers.
ATARI part number C061570 Rev. A

As perennial beginners, we find the ATARI reference manuals difficult to understand. Fortunately, however, we found a book that has told us almost everything we want to know about the ATARI computers. Here is a book we wish we had written:

Poole, Lon with Martin McNiff & Steven Cook. *Your ATARI Computer.* Published by Osborne/McGraw-Hill, 630 Bancroft Way, Berkeley, CA 94710.

Also look in Appendix H for a list of books, magazines, and other things to help you move onward. Good luck. Enjoy your journey.

The End...

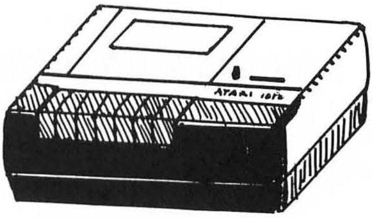
of the beginning.

Appendix A

Using the ATARI Program Recorder

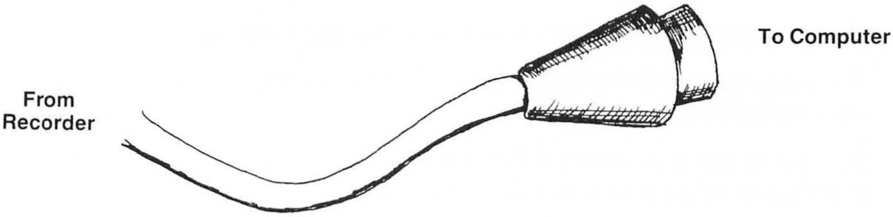
ATARI computer programs can be permanently stored on tape cassettes. You can buy professionally written programs or you can record your own programs on cassettes. You can also find program listings in magazines, type them into your computer, and save them on cassettes. You can use the ATARI model 410 or ATARI Model 1010 Program recorder to load cassette programs into the computer's memory or to save a program already in memory on a blank cassette.

ATARI 410 Program Recorder



ATARI 1010 Program Recorder

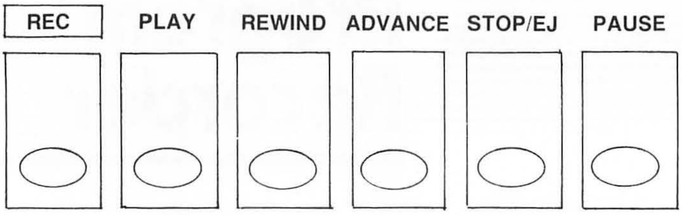
An ATARI Program recorder has two cables attached to it: a power cord and a cable that plugs into an ATARI computer.



Plug this cable into the computer outlet labelled PERIPHERAL. This outlet is on the right side of an ATARI 400 or 800. It is on the back edge of an ATARI 600XL or 800XL, near the right side as you face the computer.

Of course, also plug the power cord into a 110-volt electrical outlet.

The program recorder has six keys you will use to load or save programs.



All keys lock down except the STOP/EJ. (STOP/EJECT) key. To release a key, press the STOP/EJ. key. Oops! That doesn't work for PAUSE. If the PAUSE key is down, press it and it will pop up. You can also release most other keys by pressing another key. Try it.

The program recorder has a tape counter to help you position the tape where you want it. Press the little button to set it to 000.

SAVING A PROGRAM ON TAPE (CSAVE)

Suppose there is a program in the computer's memory and you want to record it on a cassette. Here's how:

1. Put a blank cassette in the program recorder.
2. REWIND the tape.
3. Set the tape counter to zero.
4. Use the ADVANCE key to position the tape to the number of your choice in the tape counter. Later you will use this number to help you find the program on the tape.
5. On the keyboard type CSAVE and press the RETURN key.
6. The computer will beep twice. Press the REC and PLAY keys together so that they both stay down.
7. Now press the computer's RETURN key. The tape starts moving. The computer is recording a program from its memory.

You can listen in. Turn up the volume on the TV. First you hear a steady high-pitched tone, then a burst of different sound. Each time this happens, you know the computer has recorded a piece of the program. When it is finished, the sound stops and the tape no longer moves.

8. Press the STOP key, REWIND the tape and remove it from the program recorder.

You may wish to make two or more copies of the program on the same tape. In this case, *don't* press STOP and remove the program. Instead, advance the tape a little and note the number in the tape counter — write it down. Then record another copy of the program on the tape.

LOADING A PROGRAM FROM TAPE (CLOAD)

Use CLOAD to load a program that was recorded using the CSAVE command described above. Here's how:

1. Put the program cassette in the program recorder.
2. REWIND the tape.
3. Set the tape counter to zero.
4. Use the ADVANCE key to position the tape to the beginning of the program on the tape. If you don't know the tape counter number, try starting from 000 and see if that works.
5. On the keyboard, type CLOAD and press the RETURN key.
6. The computer beeps. Press only the PLAY key so that it stays down.
7. On the keyboard, press the RETURN key. The tape starts moving. The computer is loading a program from tape into its memory.

You can listen in. Turn up the volume on the TV. You hear several seconds of silence, then a burst of sound. Each burst of sound tells you the computer has loaded a piece of the program. When the entire program has been loaded into memory, the sound stops and the tape stops moving.

8. Press the STOP key on the program recorder, REWIND the tape and remove it from the recorder.

The program is in the computer's memory. You can type RUN to run it, LIST to list it on the screen, CSAVE it on another tape, or do whatever you want to do.

> NOTE: The CLOAD command first erases any program already in memory, then loads the program from tape.

SAVE AND LOAD

CSAVE and CLOAD work only with the program recorder. Two other commands, SAVE and LOAD, work with the program recorder and also with other peripheral devices. To save a program on a tape cassette,

Type: SAVE "C:" and press RETURN.

To load a program saved with SAVE, you can use either CLOAD of LOAD.

Type: CLOAD and press RETURN,

or type: LOAD "C:" and press RETURN.

The LOAD command erases any old program in memory before reading in the new program.

LIST AND ENTER

Sometimes you may want to store only part of a program or load only a part of a program. You can do this with the LIST and ENTER commands.

To record the entire program in the computer's memory,

Type: LIST "C:" and press RETURN

You can record part of a program by specifying the first and last line numbers. For example, suppose you want to record the part of a program from line number 300 to line number 599.

Type LIST "C:",300,599 and press RETURN.

If you record a program using LIST, you must use ENTER to get it back into the computer's memory.

Type ENTER "C:" and press RETURN.

IMPORTANT NOTICE: When you use the ENTER command, the computer does not erase the program already in memory. Instead, it *merges* the program on tape with the program in memory.

You can use LIST to record subroutines, DATA statements, and other things you might use in more than one program. Then use ENTER to add them to the program already in memory.

Appendix B

Arithmetic

Use the following symbols to tell the ATARI computer to do arithmetic:

Operation	Symbol	Example	Result
Addition	+	3+4	7
Subtraction	−	3−4	−1
Multiplication	*	3*4	12
Division	/	3/4	.75

If an expression has two or more operations, they are done in the same order used in pencil-and-paper arithmetic. Here are some examples:

Additions and subtractions are done in left to right order.

$$2\overset{\textcircled{1}}{+}3\overset{\textcircled{2}}{+}4 = 9 \qquad 2\overset{\textcircled{1}}{+}3\overset{\textcircled{2}}{-}4 = 1$$

Multiplications and divisions are done left to right.

$$2\overset{\textcircled{1}}{*}3\overset{\textcircled{2}}{*}4 = 24 \qquad 2\overset{\textcircled{1}}{*}3\overset{\textcircled{2}}{/}4 = 1.5$$

$$2\overset{\textcircled{1}}{/}3\overset{\textcircled{2}}{*}4 = 2.66666666$$

Multiplications and divisions are done before additions or subtractions.

$$2\overset{\textcircled{1}}{*}3\overset{\textcircled{2}}{+}4 = 10 \qquad 2\overset{\textcircled{2}}{+}3\overset{\textcircled{1}}{*}4 = 14$$

$$2\overset{\textcircled{2}}{+}3\overset{\textcircled{1}}{/}4 = 2.75 \qquad 2\overset{\textcircled{2}}{-}3\overset{\textcircled{1}}{/}4 = 1.25$$

$$2\overset{\textcircled{1}}{*}3\overset{\textcircled{3}}{+}4\overset{\textcircled{2}}{*}5 = 26 \qquad 2\overset{\textcircled{2}}{+}3\overset{\textcircled{1}}{/}4\overset{\textcircled{3}}{+}5 = 7.75$$

Operations within parentheses are performed first. If parentheses are nested (one pair within another pair), operations within the innermost pair are performed first, then evaluation proceeds to the next pair out.

$$2\overset{\textcircled{2}}{*}(3\overset{\textcircled{1}}{+}4) = 2*7 = 14$$

$$2\overset{\textcircled{2}}{/}(3\overset{\textcircled{1}}{*}4) = 2/12 = 0.1666666666$$

$$(2\overset{\textcircled{1}}{+}3)\overset{\textcircled{3}}{*}(4\overset{\textcircled{2}}{+}5) = 5*9 = 45$$

$$\begin{aligned} 2\overset{\textcircled{4}}{*}(3\overset{\textcircled{3}}{+}4\overset{\textcircled{2}}{*}(5\overset{\textcircled{1}}{+}6)) &= 2*(3+4*11) \\ &= 2*(3+44) \\ &= 2*47 \\ &= 94 \end{aligned}$$

Numbers larger than 9999999999 or smaller than .01 are printed in *floating point*. Floating point notation is similar to *scientific notation* used in math, science, and engineering books.

Number	Scientific Notation	Floating Point Notation
10000000000	1×10^{10}	1E+10
20000000000	2×10^{10}	2E+10
.000000001	1×10^{-9}	1E−09
.0000000002	2×10^{-10}	2E−10

Floating point notation is a compact way of expressing very large or quite small numbers. In floating point, a number is represented by a *mantissa* and an *exponent*, separated by the letter E.

The number of miles in a light year:

$$\underbrace{5.865696}_{\text{mantissa}}\underbrace{E+12}_{\text{exponent}}$$

Time in seconds for light to travel one inch:

$$\underbrace{8.485391549}_{\text{mantissa}}\underbrace{E-11}_{\text{exponent}}$$

EXPERIMENT! Use this program:

```
10 GR.0
20 PR."NUMBER,PLEASE";:IN.N
30 PR.N
40 PR.
50 GOTO20
```

Try these numbers:

Volume of the Earth in bushels: 31708000000000000000000

Speed of a snail in kilometers per second: .000015

Mass of the hydrogen atom in kilograms: .00000000000000000000167

Mass of the Earth in kilograms: 598000000000000000000000

Your Turn. Call the Reference desk at your local library and get these numbers.

Diameter of a gnat's eyelash in miles.

Volume of Jupiter in thimblefuls (or is that "thimbles full?").

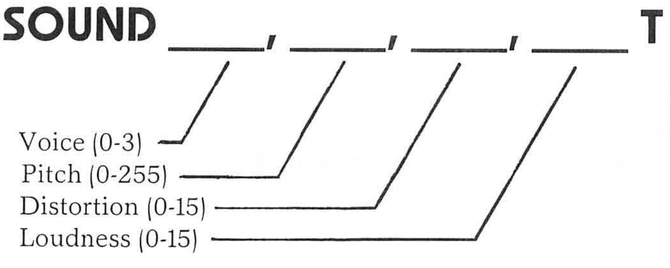
Appendix C

Sound and Music

Using the SOUND statement, you can tell the ATARI computer to make sounds and play music. The computer has four independent voices, so it can make up to four different sounds at the same time. You can use these four voices to create harmony, dissonance, sound effects, and just plain old noise.

SOUND

Use the SOUND statement to tell the computer to make a sound.



For example: SOUND 0,121,10,10

PITCH

The second number in a SOUND statement sets the pitch, or frequency, of the sound. Small numbers produce high tones and large numbers produce low tones. Zero (0) produces no tone.

DISTORTION

The third number determines the type of distortion in the sound. Use 10 or 14 to get undistorted sound, "pure tones." Other even numbers produce different kinds of distortion (noise) in the sound. In most cases, odd numbers (1,3,5,7,9,11,13,15) turn off the sound.

Values of distortion other than 10 or 14 may change the pitch of the sound. Try these.

SO. 0,121,10,10	Pure tone, Middle C.
SO. 0,121,12,10	Lower, distorted tone.

LOUDNESS

The fourth number controls the loudness for a given setting of the volume control on the TV. Loudness ranges from 0 (silence) to 15 (loudest sound).

TURNING OFF THE SOUND

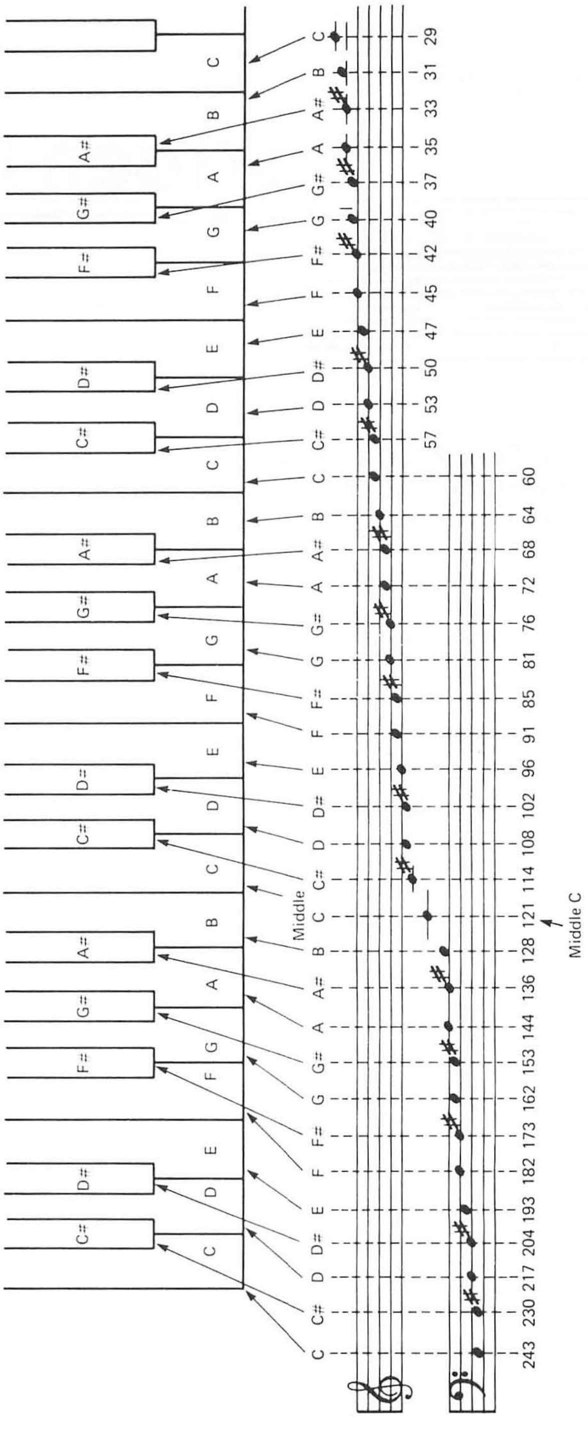
To turn off a single voice, use a SOUND statement with zero loudness. We usually also set pitch and distortion to zero.

Turn off voice 0:	SOUND 0,0,0,0
Turn off voice 1:	SOUND 1,0,0,0
Turn off voice 2:	SOUND 2,0,0,0
Turn off voice 3:	SOUND 3,0,0,0

An END statement turns off all four voices. Pressing SYSTEM RESET also turns off all four voices.

MUSICAL NOTES

The computer can produce pure tones (distortion 10 or 14) as shown by the table on page 353. This table relates selected pitch values to notes on the musical staff and keys on the piano. These relationships may not be true for distortion other than 10 or 14.

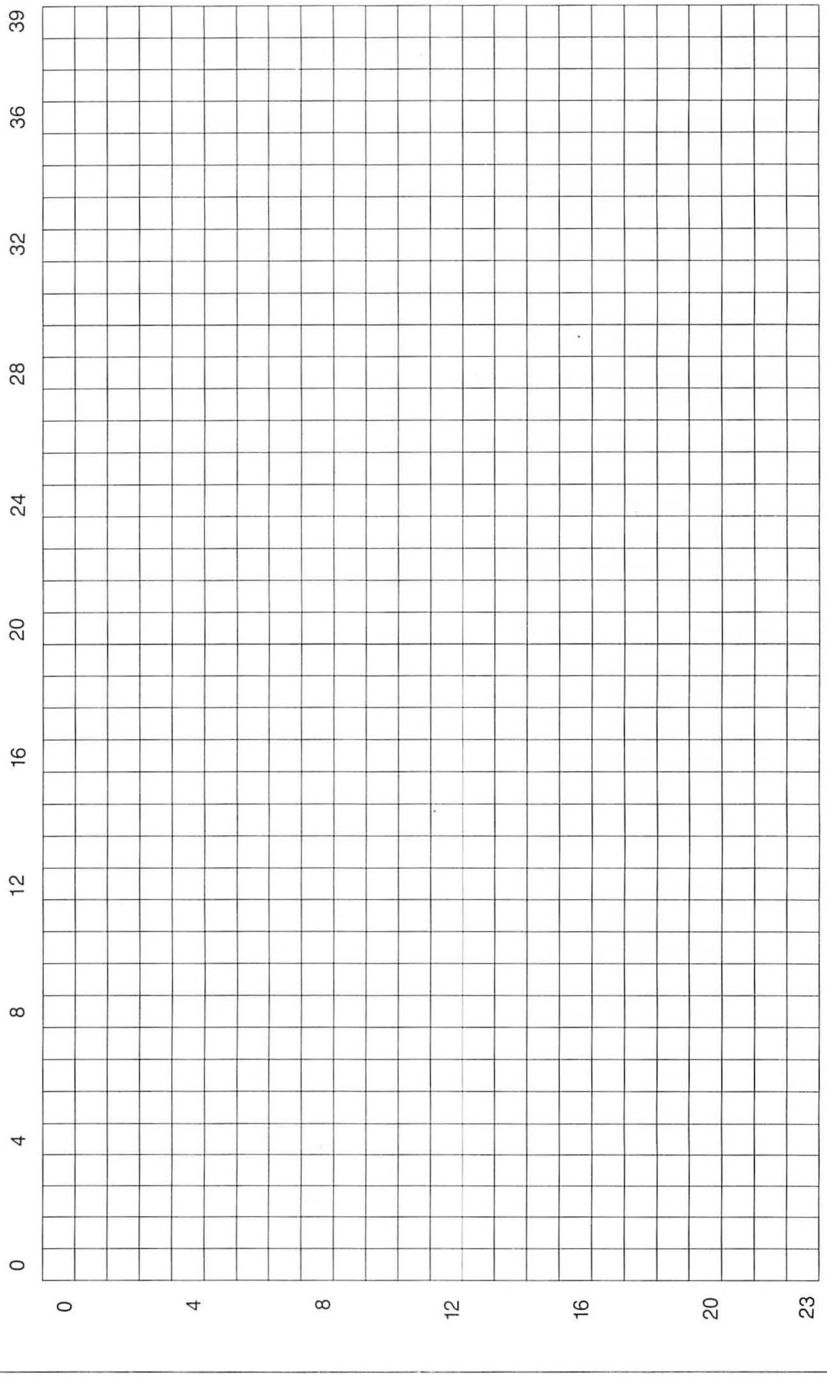

C 29
B 31
A# 33
A 35
G# 37
G 40
F# 42
F 45
E 47
D# 50
D 53
C# 57
C 60
B 64
A# 68
A 72
G# 76
G 81
F# 85
F 91
E 96
D# 102
D 108
C# 114
Middle C 121
B 128
A# 136
A 144
G# 153
G 162
F# 173
F 182
E 193
D# 204
D 217
C# 230
C 243
Middle C

Appendix D

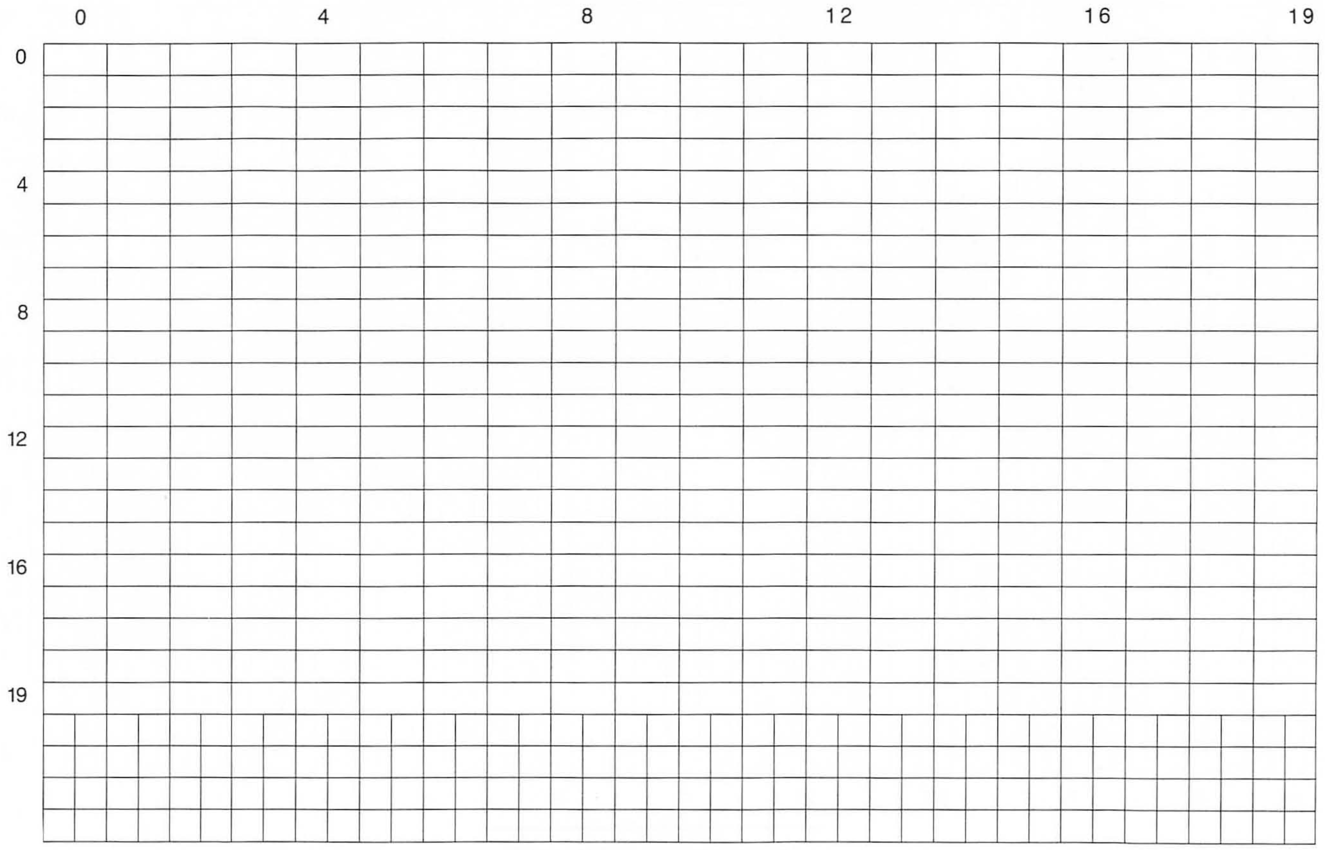
Screen Maps

You may copy the screen maps in this appendix for your own personal use. The screen is your palette; the screen is your page. Here are worksheets for designing stuff to appear on the screen:

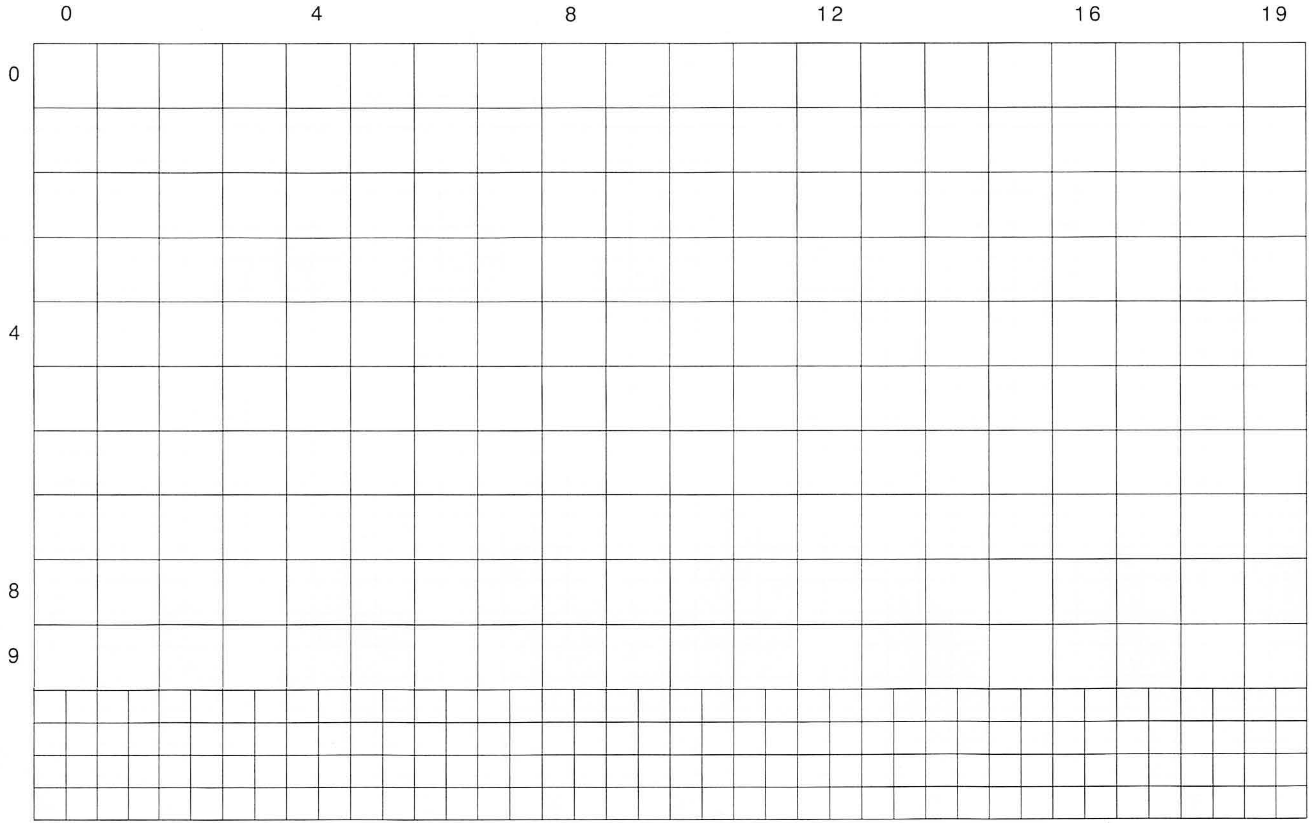
Graphics Mode 0



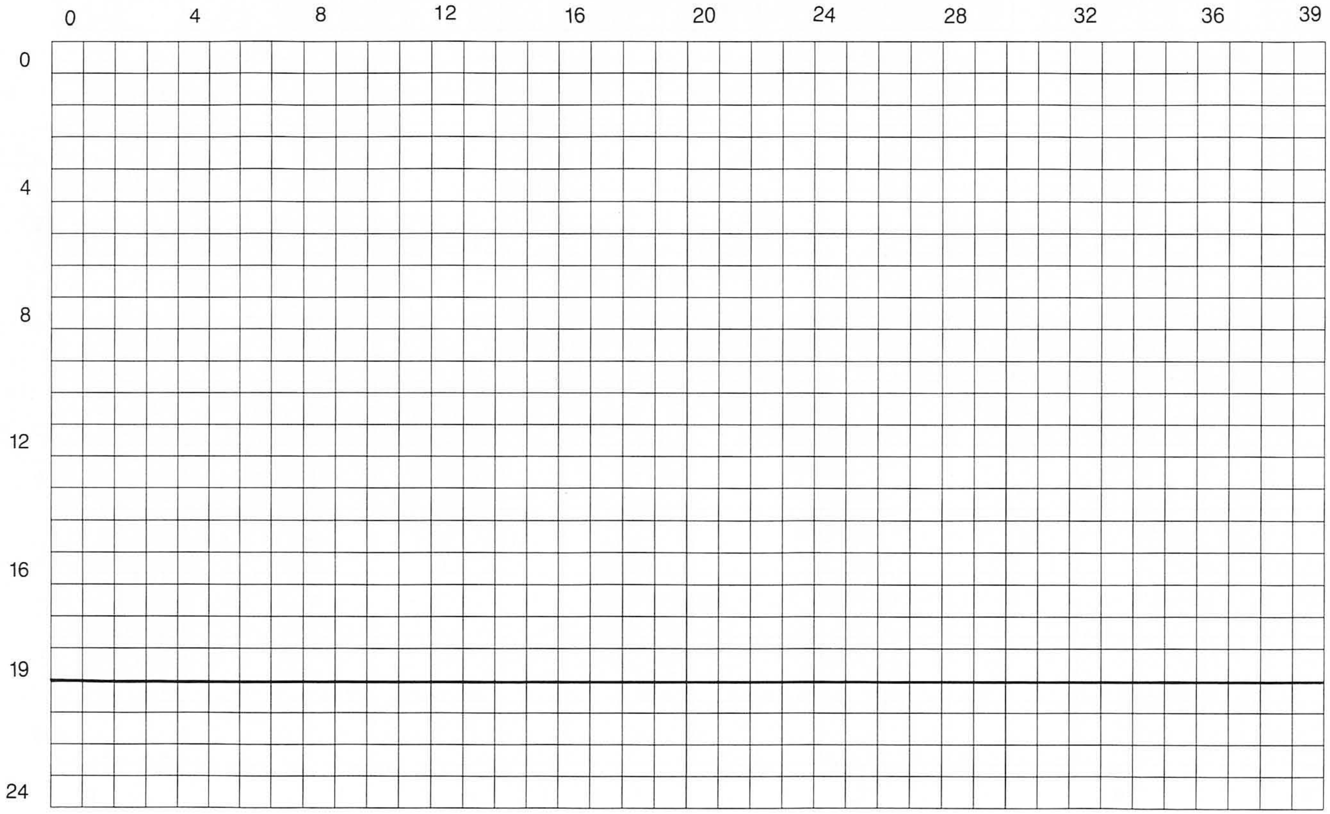
Graphics Mode 1 with Text Window



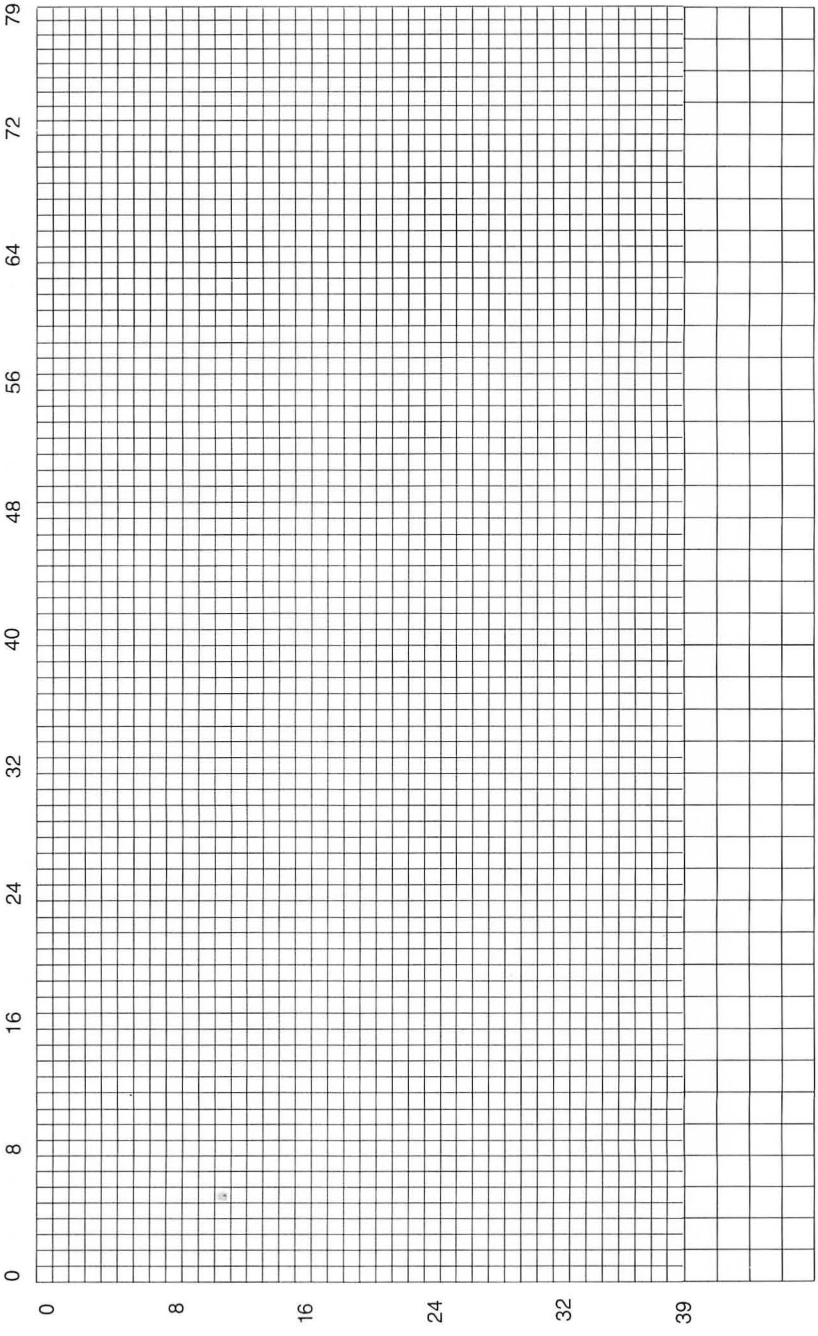
Graphics Mode 2 with Text Window



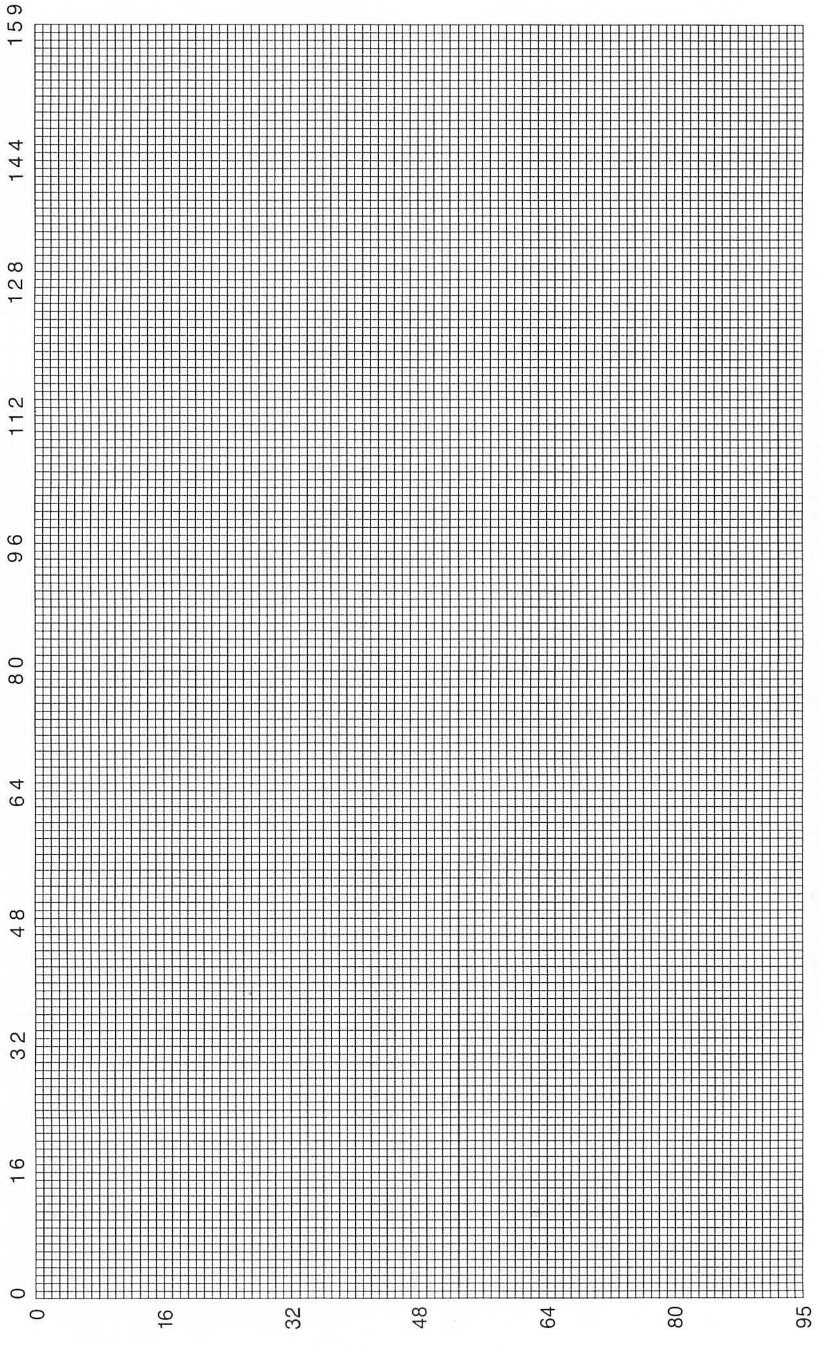
Graphics Mode 3 with Text Window



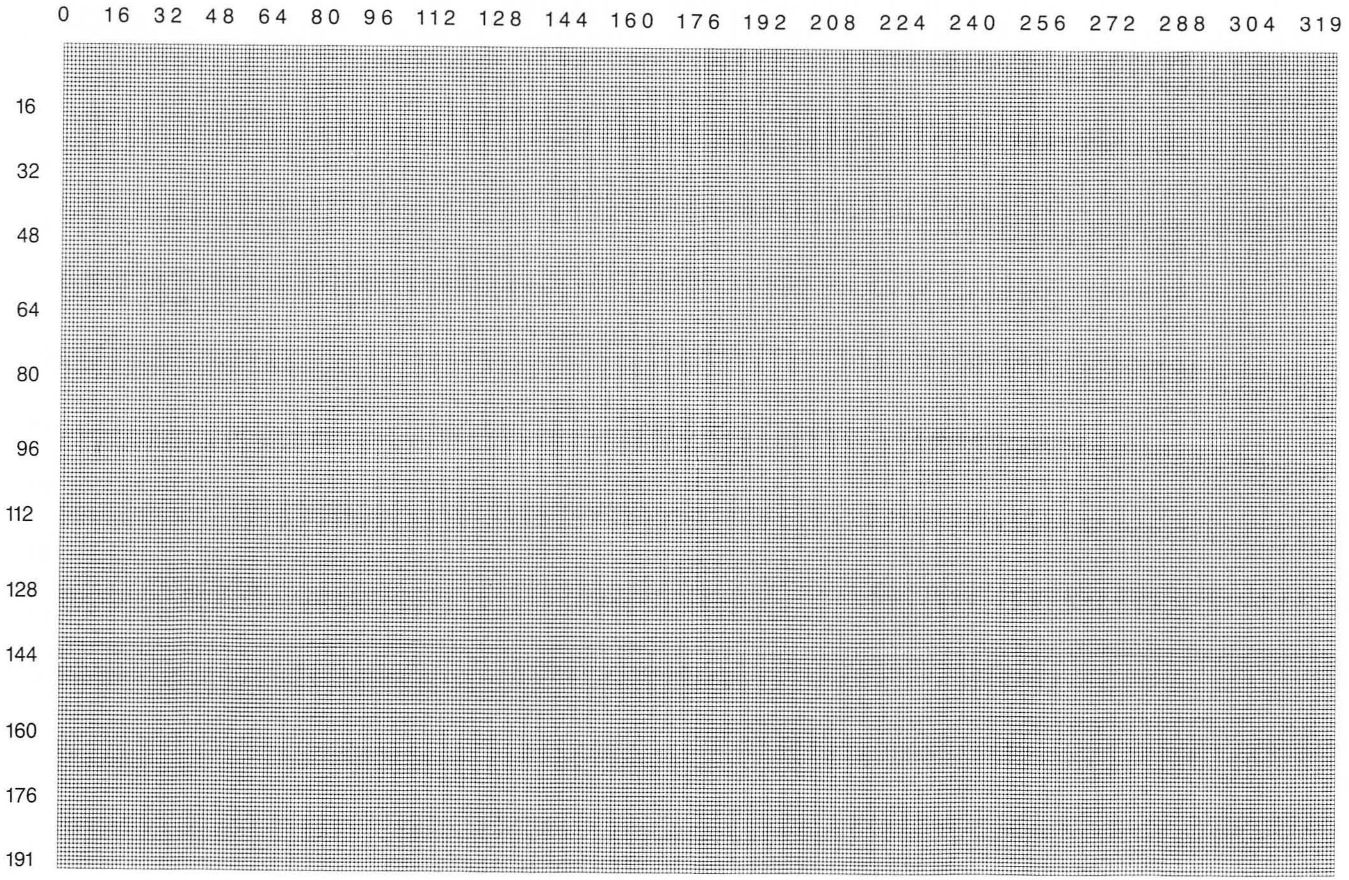
Graphics Mode 4 or 5 with Text Window



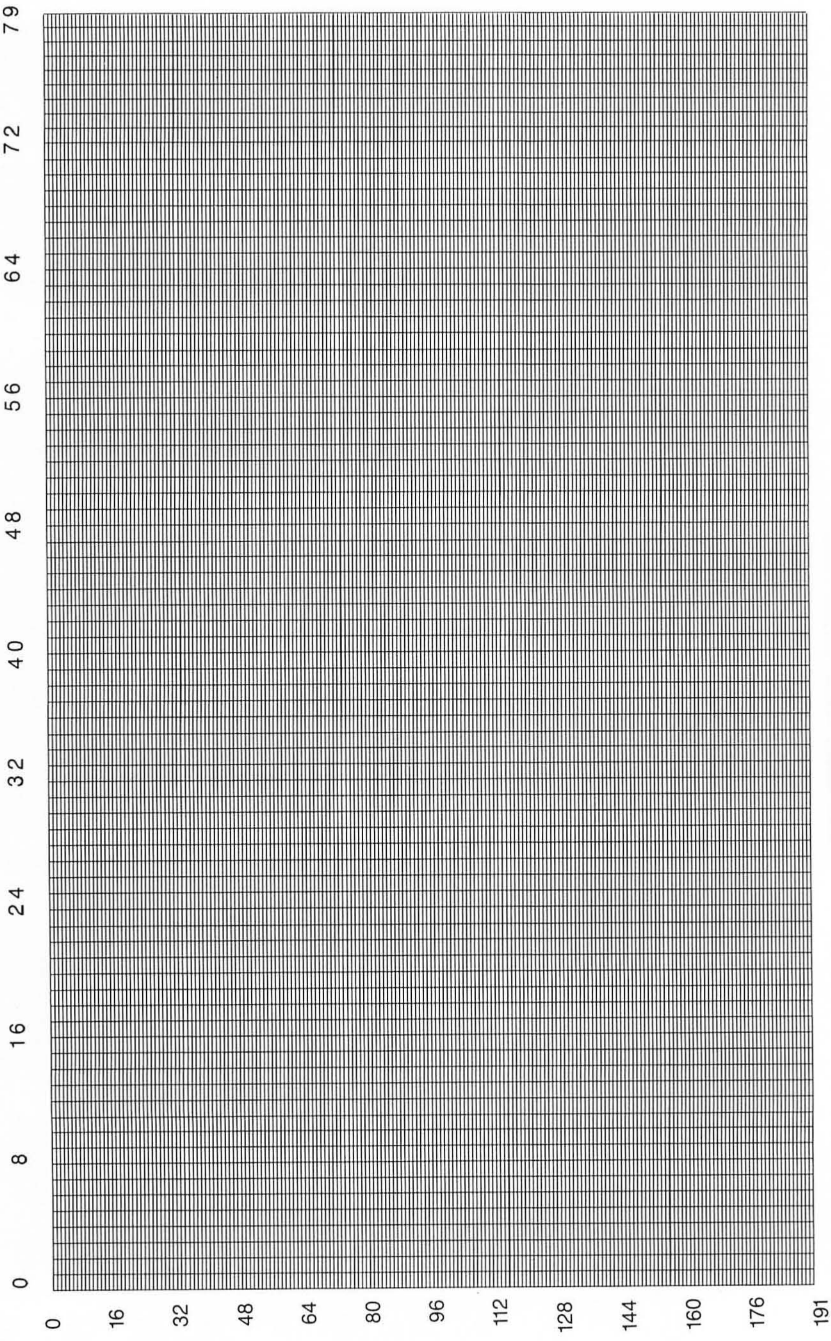
Graphics Mode 6 or 7



Graphics Mode 8



Graphics Mode 9, 10, or 11 (GTIA)



Appendix
E

ATASCII Codes

Most computers use a standard code called ASCII, which means "American Standard Code for Information Interchange." ATARI computers use a similar code called ATASCII, which means "ATari ASCII."

An ATASCII code is a number in the range 0 to 255. An ATASCII code represents a keyboard character or control function. Some ATASCII codes are generated by pressing two or more keys simultaneously or in a certain order. Conversely, you can produce the ATASCII character or control feature by using the ATASCII code in a CHR$ function.

For eachATASCII code, the following table shows the printed character (if any), and the key or keys you press to send the code into the computer. In the table:

CTRL C	means hold down CTRL and press C.
ESC \ CTRL +	means press ESC, then hold down CTRL and press +.
SHIFT 4	means hold down SHIFT and press 4.
ATARI Key CTRL ,	means press Λ to get inverse video*, then hold down CTRL and press the comma key.

If we show letters in upper case, make sure the computer is in CAPS mode; if we show letters in lower case, make sure the computer is in LOWR mode.

* On the 600XL or 800XL, press ◩

ATASCII Code	ATASCII Character	Key(s)
0	♥	CTRL ,
1	├	CTRL A
2	▕	CTRL B
3	┘	CTRL C

ATASCII Code	ATASCII Character	Key(s)
4	┤	CTRL D
5	┐	CTRL E
6	╱	CTRL F
7	╲	CTRL G
8	◢	CTRL H
9	▗	CTRL I
10	◣	CTRL J
11	▝	CTRL K
12	▘	CTRL L
13	▔	CTRL M
14	▁	CTRL N
15	▖	CTRL O
16	♣	CTRL P
17	┌	CTRL Q
18	─	CTRL R
19	┼	CTRL S
20	●	CTRL T
21	▄	CTRL U
22	▎	CTRL V
23	┬	CTRL W
24	┴	CTRL X
25	▌	CTRL Y

ATASCII Code	ATASCII Character	Key(s)
26	└	CTRL Z
27	␛	ESC \ ESC
28	↑	ESC \ CTRL –
29	↓	ESC \ CTRL =
30	←	ESC \ CTRL +
31	→	ESC \ CTRL *
32	□	Space key
33	!	SHIFT 1
34	"	SHIFT 2
35	#	SHIFT 3
36	$	SHIFT 4
37	%	SHIFT 5
38	&	SHIFT 6
39	:	SHIFT 7
40	<	SHIFT 8
41	>	SHIFT 9
42	*	*
43	+	+
44	,	,
45	−	–
46	.	•
47	/	/

Reprinted courtesy of Atari.

ATASCII Code	ATASCII Character	Key(s)
48	0	0
49	1	1
50	2	2
51	3	3
52	4	4
53	5	5
54	6	6
55	7	7
56	8	8
57	9	9
58	:	SHIFT ;
59	;	;
60	<	<
61	=	=
62	>	>
63	?	SHIFT /
64	@	SHIFT 8
65	A	A
66	B	B
67	C	C
68	D	D
69	E	E

ATASCII Code	ATASCII Character	Key(s)
70	F	F
71	G	G
72	H	H
73	I	I
74	J	J
75	K	K
76	L	L
77	M	M
78	N	N
79	O	O
80	P	P
81	Q	Q
82	R	R
83	S	S
84	T	T
85	U	U
86	V	V
87	W	W
88	X	X
89	Y	Y
90	Z	Z
91	[	SHIFT ;

Reprinted courtesy of Atari.

ATASCII Code	ATASCII Character	Key(s)
92	\	SHIFT ,
93	]	SHIFT +
94	^	SHIFT *
95	_	SHIFT –
96	♦	CTRL .
97	a	a
98	b	b
99	c	c
100	d	d
101	e	e
102	f	f
103	g	g
104	h	h
105	i	i
106	j	j
107	k	k
108	l	l
109	m	m
110	n	n
111	o	o
112	p	p
113	q	q

ATASCII Code	ATASCII Character	Key(s)
114	r	r
115	s	s
116	t	t
117	u	u
118	v	v
119	w	w
120	x	x
121	y	y
122	z	z
123	♣	CTRL ;
124	|	SHIFT =
125	↖	ESC \ CTRL <
126	◀	ESC \ BACK S
127	▶	ESC \ TAB
128	♥	* ATARI Key CTRL ,
129	┣	ATARI Key CTRL A
130	▐	ATARI Key CTRL B
131	┘	ATARI Key CTRL C
132	┤	ATARI Key CTRL D
133	┐	ATARI Key CTRL E
134	╱	ATARI Key CTRL F
135	╲	ATARI Key CTRL G

* ◩ on 600XL or 800XL

Reprinted courtesy of Atari.

ATASCII Code	ATASCII Character	Key(s)
136	◢	ATARI Key CTRL H
137	▗	ATARI Key CTRL I
138	◣	ATARI Key CTRL J
139	▝	ATARI Key CTRL K
140	▘	ATARI Key CTRL L
141	▔	ATARI Key CTRL M
142	▁	ATARI Key CTRL N
143	▖	ATARI Key CTRL O
144	♣	ATARI Key CTRL P
145	┌	ATARI Key CTRL Q
146	─	ATARI Key CTRL R
147	┼	ATARI Key CTRL S
148	●	ATARI Key CTRL T
149	▄	ATARI Key CTRL U
150	▎	ATARI Key CTRL V
151	┬	ATARI Key CTRL W
152	┴	ATARI Key CTRL X
153	▌	ATARI Key CTRL Y
154	└	ATARI Key CTRL Z
155	EOL	RETURN
156	↑	ESC \ SHIFT BACK S
157	↓	ESC \ SHIFT >

ATASCII Code	ATASCII Character	Key(s)
158	←	ESC \ CTRL TAB
159	→	ESC \ SHIFT TAB
160	■	ATARI Key SPACE BAR
161	!	ATARI Key SHIFT 1
162	"	ATARI Key SHIFT 2
163	#	ATARI Key SHIFT 3
164	$	ATARI Key SHIFT 4
165	%	ATARI Key SHIFT 5
166	&	ATARI Key SHIFT 6
167	*	ATARI Key SHIFT 7
168	(	ATARI Key SHIFT 9
169	)	ATARI Key SHIFT 0
170	*	ATARI Key *
171	+	ATARI Key +
172	,	ATARI Key ,
173	—	ATARI Key –
174	.	ATARI Key .
175	/	ATARI Key /
176	0	ATARI Key 0
177	1	ATARI Key 1
178	2	ATARI Key 2
179	3	ATARI Key 3

Reprinted courtesy of Atari.

ATASCII Code	ATASCII Character	Key(s)
180	4	ATARI Key 4
181	5	ATARI Key 5
182	6	ATARI Key 6
183	7	ATARI Key 7
184	8	ATARI Key 8
185	9	ATARI Key 9
186	:	ATARI Key SHIFT ;
187	;	ATARI Key ;
188	<	ATARI Key <
189	=	ATARI Key =
190	>	ATARI Key >
191	?	ATARI Key SHIFT /
192	@	ATARI Key SHIFT 8
193	A	ATARI Key A
194	B	ATARI Key B
195	C	ATARI Key C
196	D	ATARI Key D
197	E	ATARI Key E
198	F	ATARI Key F
199	G	ATARI Key G
200	H	ATARI Key H
201	I	ATARI Key I

ATASCII Code	ATASCII Character	Key(s)
202	J	ATARI Key J
203	K	ATARI Key K
204	L	ATARI Key L
205	M	ATARI Key M
206	N	ATARI Key N
207	O	ATARI Key O
208	P	ATARI Key P
209	Q	ATARI Key Q
210	R	ATARI Key R
211	S	ATARI Key S
212	T	ATARI Key T
213	U	ATARI Key U
214	V	ATARI Key V
215	W	ATARI Key W
216	X	ATARI Key X
217	Y	ATARI Key Y
218	Z	ATARI Key Z
219	[	ATARI Key SHIFT ,
220	\	ATARI Key SHIFT +
221	]	ATARI Key SHIFT .
222	^	ATARI Key SHIFT *
223	_	ATARI Key SHIFT –

Reprinted courtesy of Atari.

ATASCII Code	ATASCII Character	Key(s)
224	◆	ATARI Key CTRL .
225	a	ATARI Key a
226	b	ATARI Key b
227	c	ATARI Key c
228	d	ATARI Key d
229	e	ATARI Key e
230	f	ATARI Key f
231	g	ATARI Key g
232	h	ATARI Key h
233	i	ATARI Key i
234	j	ATARI Key j
235	k	ATARI Key k
236	l	ATARI Key l
237	m	ATARI Key m
238	n	ATARI Key n
239	o	ATARI Key o
240	p	ATARI Key p
241	q	ATARI Key q
242	r	ATARI Key r
243	s	ATARI Key s
244	t	ATARI Key t
245	u	ATARI Key u

ATASCII Code	ATASCII Character	Key(s)
246	v	ATARI Key v
247	w	ATARI Key w
248	x	ATARI Key x
249	y	ATARI Key y
250	z	ATARI Key z
251	♠	ATARI Key CTRL ;
252	|	ATARI Key CTRL =
253	↰	ESC \ CTRL 2
254	◀	ATARI Key ESC \ CTRL BACK S
255	▶	ATARI Key ESC \ CTRL >

Reprinted courtesy of Atari.

Appendix F

ATARI BASIC Reserved Words

The following words are reserved words in ATARI BASIC. A reserved word may not be used as a variable or as the beginning of a variable. For example, you may not use VAL or VALUE as a variable because VAL is a reserved word.

ABS
ADR
AND
ASC
ATN
BYE
CLOAD
CHR$
CLOG
CLOSE
CLR
COLOR
COM
CONT
COS
CSAVE
DATA
DEG
DIM
DOS
DRAWTO
END
ENTER
EXP
FOR
FRE
GET
GOSUB
GOTO
GRAPHICS
IF
INPUT
INT
LEN
LET
LIST
LOAD
LOCATE
LOG
LPRINT
NEW
NEXT
NOT
NOTE
ON
OPEN
OR
PADDLE
PEEK
PLOT
POINT
POKE
POP
POSITION
PRINT
PTRIG
PUT
RAD
READ
REM
RESTORE
RETURN
RND
RUN
SAVE
SETCOLOR
SGN
SIN
SOUND
SQR
STATUS
STEP
STICK
STRIG
STOP
STR$
THEN
TO
TRAP
USR
VAL
XIO

These words are the *keywords* used in writing programs and telling the computer things to do. Some of the keywords have abbreviations you can use to more quickly type them into the computer. The following keywords have abbreviations.

KEYWORD	ABBREVIATION	KEYWORD	ABBREVIATION
BYE	B.	NEXT	N.
CLOAD	CLOA.	NOTE	NO.
CLOSE	CL.	OPEN	O.
COLOR	C.	PLOT	PL.
CONT	CON.	POINT	P.
DATA	D.	POKE	POK.
DEG	DE.	POSITION	POS.
DIM	DI.	PRINT	PR. OR ?
DOS	DO.	PUT	PU.
DRAWTO	DR.	READ	REA.
ENTER	E.	REM	R. or . SPACE
FOR	F.	RESTORE	RES.
GET	GE.	RETURN	RET.
GOSUB	GOS.	RUN	RU.
GOTO	G.	SAVE	S.
GRAPHICS	GR.	SETCOLOR	SE.
INPUT	I. or IN.	SOUND	SO.
LET	LE.	STATUS	ST.
LIST	L.	STOP	STO.
LOAD	LO.	TRAP	T.
LOCATE	LOC.	XIO	X.
LPRINT	LP.		

If you LIST a program, all keywords are spelled out in full, even though you typed the abbreviations.

Appendix
G

Error Messages

ERROR CODE NO.	WHAT IT MEANS
2	You ran out of memory. This may happen when you enter a program line or DIM a string variable or an array.
3	Value error. A value is outside its legal range.
4	Too many variables. A maximum of 128 different variable names is allowed.
5	String length error. This happens if you try to store a string longer than specified in a DIM statement.
6	Out of data error. A READ statement is executed after all items in DATA statements have already been read.
7	Number greater then 32767 or less than 0.
8	INPUT statement error. Attempt to enter a non-numeric value for a numeric variable.
9	DIM error. DIM size is greater than 32767 or a subscript is outside the range specified in a DIM statement or a string or subscripted variable has not been DIMensioned or a string or array is DIMensioned a second time.
10	Argument stack overflow. Too many GOSUBs or too large an expression.
11	Floating point overflow or underflow. The result of an operation is greater than 1E 98 (overflow) or less than 1E-99 (underflow). This happens when you try to divide by zero.
12	Line not found. A GOSUB, GOTO, or IF-THEN statement references a non-existent line number.

13 NEXT without FOR. The computer encountered a NEXT statement without a matching FOR statement.

14 Line too long. The statement is too long or too complex.

15 GOSUB or FOR line missing. A NEXT or RETURN statement was encountered, but the corresponding FOR or GOSUB had been deleted since the last RUN. (Nevermind — we don't understand this one either).

16 RETURN without GOSUB.

17 Garbage error. Might be a bad memory chip, but can also occur from unwise use of POKE. Type NEW and re-enter the program or turn the computer off and on, then re-enter the program.

18 Invalid string character. String does not start with a valid character or string used in a VAL function is not a numeric string.

The following are INPUT or OUTPUT errors that might happen during the use of printers, disk drives, or other accessory devices. For more information, consult the reference manuals that come with such devices.

19 LOAD program too long.

20 Device number larger than 7 or equal to 0.

21 LOAD file error.

128 BREAK abort. User hit BREAK key during input or output operation.

129 IOCB already open. (Input/Output Control Block)

130 Nonexistent device specified.

131 IOCB write only.

132 Invalid command.

133 Device or file not open.

134 Bad IOCB number.

135 IOCB read only error.

136 EOF (End of File) has been reached.

137 Truncated record.

138 Device timeout.

139 Device NAK

140	Serial bus input framing error.
141	Cursor out of range for this mode.
142	Serial bus data frame overrun.
143	Serial bus data frame checksum error.
144	Device done error.
145	Read after write error.
146	Function not implemented.
147	Not enough RAM memory for selected graphics mode.
160	Drive number error.
161	Too many open files.
162	Disk full.
163	Unrecoverable system data I/O error.
164	File number mismatch.
165	File name error.
166	POINT data length error.
167	File locked.
168	Command invalid.
169	Directory full.
170	File not found.
171	POINT invalid.

Appendix
H

Look Here First

We hope you looked here soon after you first cracked this book. You can teach yourself how to use, program, and enjoy computers better, and have more fun, if you use more than one source of information and inspiration. As you progress from beginner to intermediate to advanced and perhaps beyond, try some of these.

BOOKS

- Moore, Herb, Judy Lower, and Bob Albrecht, *ATARI SOUND AND GRAPHICS.* (John Wiley and Sons, Inc., 655 Third Avenue, New York, NY 10158.) A beginner's book devoted entirely to sound and graphics. Herb Moore is a creative musician with a flair for writing.
- Kohl, Herb, Ted Kahn, and Len Lindsay, *ATARI GAMES AND RECREATIONS.* (Reston Publishing Co., 11480 Sunset Hills Road, Reston, VA 22090) Beginners can use this book to learn how to design and program computer games in ATARI BASIC.
- Poole, Lou, Martin McNiff, and Steven Cook, *YOUR ATARI COMPUTER: A GUIDE TO ATARI 400/800 PERSONAL COMPUTERS.* (Osborne/McGraw-Hill, 630 Bancroft Way, Berkeley, CA 94710.) This is the most complete source of information about ATARI computers. Use it as your reference guide after you finish working and playing your way through ATARI BASIC.

MAGAZINES

Here are two monthly magazines devoted entirely to ATARI computers:

- *A.N.A.L.O.G.,* 565 Main Street, Cherry Valley, MA 01611. Phone: (617) 892-3488.
- *ANTIC,* 524 Second Street, San Francisco, CA 94107. Phone: (415) 957-0886.

The following monthly magazines include information on several computers, including the ATARIs.

- *COMPUTE!*. P.O. Box 5406, Greensboro, NC 27403. Phone: (800) 334-0868.
- *HOME COMPUTER,* P.O. Box 5537, Eugene, OR 97405. Phone: (503) 485-8796.

REFERENCE CARDS

These accordion-fold reference cards are mighty useful!

- *ACE POCKET REFERENCE CARD (ATARI),* Advanced Computing Enterprises, 5516 Rosehill, Shawnee KS 66216. Phone: (913) 262-2875 or (913) 631-4180.
- *REFERENCE CARD FOR THE ATARI 400/800 MICROCOMPUTERS,* Nanos Systems Corp., P.O. Box 24344, Speedway, IN 46224. Phone: (317) 244-4078.

The above information may change. For an up-to-date list, send a self-addressed, stamped envelope to DragonSmoke, P.O. Box 310, Menlo Park, CA 94026.

Index

See page 341 for a list of ATARI BASIC keywords covered in this book and page 342 for a list of keywords *not* covered in this book.

Index of Programs

The names of these programs appear in REM statements in the first line of the program.

Computers $14.95

A hands-on guide to ATARI BASIC on the XL series, 400, 800, and 1200 machines.

ATARI® BASIC XL™ EDITION

A SELF-TEACHING GUIDE

BOB ALBRECHT, LEROY FINKEL AND JERALD R. BROWN

Praise for the first edition of *Atari BASIC* which sold over 450,000 copies!

"May be the finest introduction to BASIC programming I have ever seen."
—*Microcomputing*

"Should turn a novice into a pro, if not overnight, then over the course of a week."
—*Science & Electronics*

"Really top notch!"—*Interface Age*

"An excellent place to start. I highly recommend it."—*Compute!*

The XL edition of this classic shows how to adapt BASIC to Atari's brand-new, powerful XL series of microcomputers and its 400, 800, and 1200 machines.

You need no math, science, or computer background to learn to read and write Atari BASIC. Emphasizing good programming style, this self-paced, easy-to-understand guide takes you step-by-step through simple techniques for creating programs for home, school, or business applications. Lots of educational and recreational activities and exercises—featuring sound, color, graphics games and simulations—make learning easy and fun.

BOB ALBRECHT, JERALD R. BROWN and LEROY FINKEL are founders of the People's Computer Company. They are authors of eleven other best-selling computer Self-Teaching Guides. Together, they are responsible for sales of over a million Wiley books. Bob Albrecht, with Ramon Zamora, also founded ComputerTown U.S.A.

Wiley Self-Teaching Guides have taught more than three million people to use, program, and enjoy microcomputers. Look for them all at your favorite bookshop or computer store.

WILEY PRESS
a division of JOHN WILEY & SONS, Inc.
605 Third Avenue, New York, N.Y. 10158
New York • Chichester • Brisbane •
Toronto • Singapore

ISBN 0 471-80726-5

Cover Design: DENISE LE BRUN