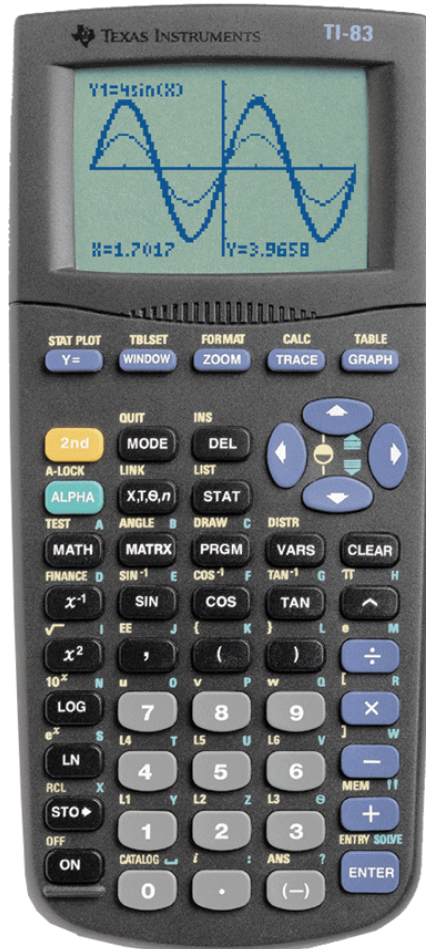




Introduction to Programming on the TI-83



Important notice regarding book materials

Texas Instruments makes no warranty, either expressed or implied, including but not limited to any implied warranties of merchantability and fitness for a particular purpose, regarding any programs or book materials and makes such materials available solely on an “as-is” basis. In no event shall Texas Instruments be liable to anyone for special, collateral, incidental, or consequential damages in connection with or arising out of the purchase or use of these materials, and the sole and exclusive liability of Texas Instruments, regardless of the form of action, shall not exceed the purchase price of this book. Moreover, Texas Instruments shall not be liable for any claim of any kind whatsoever against the use of these materials by any other party.

Permission is hereby granted to teachers to reprint or photocopy in classroom, workshop, or seminar quantities the pages or sheets in this work that carry a Texas Instruments copyright notice. These pages are designed to be reproduced by teachers for use in their classes, workshops, or seminars, provided each copy made shows the copyright notice. Such copies may not be sold, and further distribution is expressly prohibited. Except as authorized above, prior written permission must be obtained from Texas Instruments Incorporated to reproduce or transmit this work or portions thereof in any other form or by any other electronic or mechanical means, including any information storage or retrieval system, unless expressly permitted by federal copyright law. Send inquiries to this address:

Texas Instruments Incorporated
7800 Banner Drive, M/S 3918
Dallas, TX 75251

Attention: Manager, Business Services



Copyright © 1996, 2001 Texas Instruments Incorporated. Except for the specific rights granted herein, all rights are reserved.

Printed in the United States of America.

Introduction

The exercises in this guide demonstrate both general programming concepts and specific implementation on the TI-83. They are designed for those with little or no experience in programming.

You should work the exercises in order, as each exercise presumes that you understand the concepts covered in the previous one. You should also have read the appropriate sections of the *TI-83 Graphing Calculator Guidebook*, particularly Getting Started, Operating the TI-83 (Chapter 1), and Function Graphing (Chapter 3).

Words that appear in italics in the exercises are defined in the glossary at the back of this guide.

Exercise 1: Defining Environment	
Resetting Defaults.....	2
Exercise 2: Input, Output, and Program Flow	
Finding Prime Factors	9
Exercise 3: Defining the Viewing Window	
The Counting Window	18
Exercise 4: Graphs in Programs	
Guess the Slope and Intercept	21
Exercise 5: Using Programs as Building Blocks	
Exploring Triangles	27
Glossary.....	39

Exercise 1: Defining Environment

Resetting Defaults

Exercise 1 demonstrates these activities.

- Creating a new program.
- Executing a program.
- Testing a program.
- Editing and changing a program.
- Setting MODE and FORMAT.
- Setting the standard viewing WINDOW.
- Turning off Y= functions and STAT PLOT definitions.
- Using the PRGM I/O menu.

Introduction

The Getting Started example in the *TI-83 Graphing Calculator Guidebook* begins with resetting the calculator. This lets you start Exercise 1 with the calculator in a known state to assure that the keystroke-by-keystroke instructions will produce the illustrated results.

As you use the TI-83 for a variety of exercises, you may find that settings left over from previous exercises or programs must be changed before you can complete the current exercise.

In general, you should not use RESET as you did in Getting Started, because RESET also erases all variables and programs. Furthermore, RESET cannot be accessed from a program. You can reset all defaults by using the MEM RESET DEFAULTS menu; however, RESET DEFAULTS cannot be accessed from a program. In Exercise 1 you will write a program that allows you to reset the defaults with a single command.

Exercise 1: Defining Environment (Cont.)

Resetting Defaults

One of the first lessons a programmer learns is that it is important to define the environment and initialize the variables. **DEFAULTS**, the program in this exercise, defines the environment by returning MODE, WINDOW, and FORMAT settings to the defaults and by turning off, but not erasing, the Y= functions and STAT PLOT definitions. Initializing variables is discussed in Exercise 2.

You can use the program **DEFAULTS** in three ways.

- Execute it before a new exercise or program.
- Call it as a subroutine at the beginning of a program. You will do this in Exercise 4.
- Recall (RCL) it into another program as a template for defining the environment. You will do this in Exercise 5.

This exercise gives step-by-step instructions to create the program **DEFAULTS**. The program is shown here for your convenience. Begin the exercise on the next page.

```
PROGRAM:DEFAULTS
:Normal
:Float
:Radian
:Func
:Connected
:Sequential
:Real
:Full
:RectGC
:CoordOn
:GridOff
:AxesOn
:LabelOff
:ExprOn
:PlotsOff
:FnOff
:ZStandard
:ClrHome
```

Exercise 1: Defining Environment (Cont.)

Resetting Defaults

On the TI-83, programs are accessed by name; therefore, you create a new program by using the NEW menu, not the EDIT menu, to name the program.

Press **PRGM** **▶** **▶** to display the PRGM NEW menu.



EXEC EDIT **NEW**
Create New

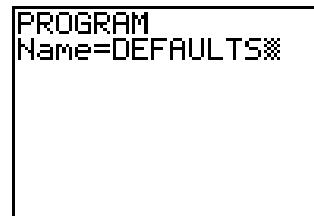
Press **1** to select **Create New**. The keyboard is set to ALPHA-LOCK.



PROGRAM
Name=

You may use numbers in a program name, except as the first character.

Press the letters **DEFAULTS** to name the program. Notice that after you enter the **S**, the cursor changes to a checkerboard pattern. This tells you that the name contains the maximum of eight characters.



PROGRAM
Name=DEFAULTS

Check the program name. You cannot change the name of a TI-83 program after you press **ENTER**.

Press **ENTER**. The *program edit screen* appears. On the TI-83 you enter program commands on the program edit screen.



PROGRAM: DEFAULTS
:

A *program* is a set of *commands* (steps, statements, or instructions) that are performed sequentially, as if they are entered directly from the keyboard. In a TI-83 program, a **:** (colon) indicates the beginning of each command. The colon at the beginning of each new *command line* on the program edit screen is created automatically by the calculator when you press **ENTER**.

You can enter more than one command on a command line by using the colon from the keyboard, just as on the home screen.

Exercise 1: Defining Environment (Cont.)

Resetting Defaults

Many of the commands in programs are familiar because you use them on the home screen. Some, however, are available as commands only in programs because they are *interactive* (the result of pressing the key is immediate) outside of programs. The commands in the **DEFAULTS** program are this type.

Press **MODE**. The screen that is displayed looks like the interactive MODE screen, but notice that no settings are highlighted. The flashing cursor is on **Normal**.

```
Normal Sci Eng
Float 0123456789
Radian Degree
Func Par Pol Seq
Connected Dot
Sequential Simul
Real a+bi re^θi
Full Horiz G-T
```

Press **ENTER**. **Normal** is copied to cursor position on the program edit screen.

```
PROGRAM:DEFAULTS
:Normal
:█
```

Press **ENTER** to start a new command line. The colon at the beginning of the command line is created automatically.

Press **MODE** to display the MODE screen again. Press **↓** to position the cursor on **Float**.

```
PROGRAM:DEFAULTS
:Radian
:Func
:Connected
:Sequential
:Real
:Full
:█
```

Press **ENTER** to copy **Float** to the program edit screen.

Press **ENTER** to begin a new command line.

Continue this process to enter the commands for all of the default MODE settings, the leftmost setting on each line.

From a new command line, press **2nd** **FORMAT** to display the FORMAT screen. Notice that it is the same type as the MODE screen.

```
RectGC PolarGC
CoordOn CoordOff
GridOff GridOn
AxesOn AxesOff
LabelOff LabelOn
ExprOn ExprOff
```

Press **ENTER** **ENTER** to copy **RectGC** to the program edit screen and begin a new command line.

```
PROGRAM:DEFAULTS
:RectGC
:CoordOn
:GridOff
:AxesOn
:LabelOff
:ExprOn
:█
```

Continue as before, selecting all FORMAT default settings.

Exercise 1: Defining Environment (Cont.)

Resetting Defaults

When you try to graph a function, it can be annoying to find a stat plot turned on from a recent exercise. It can be equally annoying to display a stat plot and discover a graphing function on the display. Although you usually turn Y= functions and stat plots on or off interactively on the Y= or STAT PLOTS screens, **PlotsOff** and **FnOff** are available for use on the home screen and from the program editor.

Press **2nd** [STAT PLOT].

Press **4** to select **PlotsOff**. **PlotsOff** is copied to the program edit screen. If you do not enter any plot number(s) after **PlotsOff**, all plots will be turned off.

Press **ENTER** to start a new line.

```
Plots TYPE MARK
1:Plot1(
2:Plot2(
3:Plot3(
4:PlotsOff
5:PlotsOn
```

Now press **VAR** **▸** **4** to display the function ON/OFF menu.

Press **2** to select **FnOff**. **FnOff** is copied to the program edit screen. **FnOff** works just like **PlotsOff**. To turn off all functions, do not enter any numbers.

Press **ENTER** to start a new line.

```
ON/OFF
1:FnOn
2:FnOff
```

The quickest way to return the viewing WINDOW to its default values is to use **ZStandard**. Press **ZOOM** to display the ZOOM menu, which looks just as it does outside of the program edit screen.

Press **6** to select **Zstandard**, which is copied to the command line. Press **ENTER**. The graph is not displayed at this time. When the program is executed, **ZStandard** automatically regraphs, which clears any drawn elements from the graph display.

```
ZOOM MEMORY
1:ZBox
2:Zoom In
3:Zoom Out
4:ZDecimal
5:ZSquare
6:ZStandard
7↓ZTrig
```

```
PROGRAM: DEFAULTS
:AxesOn
:LabelOff
:ExprOn
:PlotsOff
:FnOff
:ZStandard
:█
```

In programs where you might set the WINDOW by storing to the WINDOW variables, you may want to add **ClrDraw** to the initialization portion of the program.

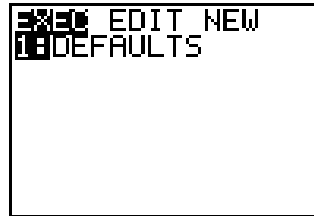
Exercise 1: Defining Environment (Cont.)

Resetting Defaults

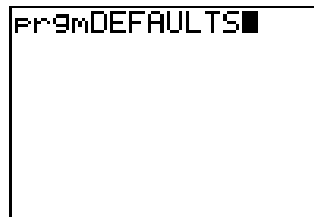
Now it's time to *execute* (run) the program. You cannot execute a program from the program editor. You must leave program editor and return to the home screen.

Press **2nd** [QUIT] to return to the home screen.

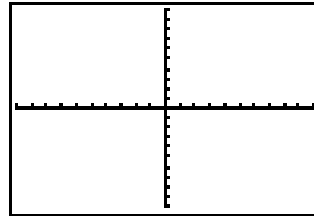
Now press **PRGM**. The PRGM EXEC menu is displayed with **DEFAULTS** as the name of one of the programs.



Select **DEFAULTS**. **prgmDEFAULTS** is copied to the home screen.



Press **ENTER** to execute the program.



When you execute the program, **ZStandard** displays the graph as the last step.

Exercise 1: Defining Environment (Cont.)

Resetting Defaults

You probably don't want the program to end with the graph displayed, so add an instruction at the end of the program to clear and display the home screen.

Press **[PRGM]** **[▶]** to display the PRGM EDIT menu.

```
EXEC [000] NEW
1:DEFAULTS
```

Select **DEFAULTS** to display the program so that you can begin *editing* (changing) it.

```
PROGRAM: DEFAULTS
: AxesOn
: LabelOff
: ExprOn
: PlotsOff
: FnOff
: ZStandard
: █
```

Press and hold **[▼]** until you reach the end of the program.

Press **[PRGM]**. This allows you to access a special set of menus that is available only from the program edit screen because they contain commands that apply only to programming.

```
CTL [ ] EXEC
2:Prompt
3:Disp
4:DispGraph
5:DispTable
6:Output(
7:getKey
8:ClrHome
```

Press **[▶]** to display the PRGM I/O menu, and then press **[▼]** until the cursor is on **ClrHome**.

Press **[ENTER]** to select **ClrHome**. This instruction clears all text from the home screen (like **[CLEAR]** **[CLEAR]**) and displays the home screen.

```
PROGRAM: DEFAULTS
: LabelOff
: ExprOn
: PlotsOff
: FnOff
: ZStandard
: ClrHome
: █
```

Press **[ENTER]**.

Press **[2nd]** **[QUIT]**. Note that the last command on the home screen is **prgmDEFAULTS**. To execute the program again, simply press **[ENTER]**. What happens? Do you like this version better?

Programmers often go through the process that you just went through.

- Enter program commands.
- Execute the program.
- Return to the program edit screen to correct errors.
- Make changes, or add improvements; and then execute the program again.

Exercise 2: Input, Output, and Program Flow

Finding Prime Factors

Exercise 2 demonstrates the following activities.

- Using **Input** to enter values in a program.
- Using **Disp** to display results.
- Initializing and updating variables.
- Inserting commands.
- Controlling program flow with **Lbl** and **Goto**.
- Controlling program flow with **If/Then/Else**.
- Controlling program flow with **If**.

Introduction

To make a program as flexible as possible, you may want it to *prompt* (display a message asking for a value) the person running the program to enter information—the value for a variable, for example. To make it as friendly as possible, you may want it to display results automatically. On the TI-83, the commands to help a program communicate with the user are grouped together on the PRGM I/O (input/output) menu. You will use two of these commands, **Input** and **Disp**, in this program. The PRGM I/O commands are described in the *TI-83 Graphing Calculator Guidebook* on pages 16-16 through 16-21.

Programs are frequently used for *iterative procedures* (groups of commands that are performed over and over until the desired result is achieved). On the TI-83, the instructions to help you control the flow of a program, including iterations, are grouped together under the PRGM CTL (control) menu.

The program **PRIMES** uses three types of control commands: **Lbl/Goto**, **If/Then/Else**, and **If** by itself. The PRGM CTL commands are described on pages 16-8 through 16-15 in the *TI-83 Graphing Calculator Guidebook*.

```
PROGRAM:PRIMES
:Input "FIND PRIMES OF: ",N
:2→Q
:Lbl A
:If fPart(N/Q)=0
:Then
:Disp Q
:Pause
:N/Q→N
:Else
:Q+1→Q
:End
:If Q≤√(N)
:Goto A
:If N≠1
:Disp N
```

Exercise 2: Input, Output, and Program Flow (Cont.)

Finding Prime Factors

To determine the *program logic* (how to structure a program), first walk through the problem manually, using pen, paper, and brain power. Notice each small step in the process; and then translate each step into one or more programming commands. For example, to calculate prime factors, you would list these steps.

1. Pick a number to find the prime factors of; call it N .
2. Choose 2 as a first guess; call the guess Q .
3. Divide the number by the guess— N/Q .
4. Is the result an integer—no remainder?
 - a. If the result is an integer, write down the correct guess Q , and then divide the number N by Q . Now try again to divide N by Q .
 - b. If the result is not an integer, try again, dividing N by the next bigger integer— $Q+1$.

At this point, you might notice that you are about to do the same things over and over. This is called a *loop*.

How do you know when you have found all of the prime factors? The largest prime factor for a number is its square root, so when Q gets bigger than the square root of N , you stop looking at larger values of Q .

5. The last step is to quit looking.

An important thing to remember about loops is to get out of them. If you are executing a simple program on the TI-83 and notice that the busy signal is moving for one full minute, but nothing else is happening, it is likely that the program contains a loop with no way out. Press **[ON]** to interrupt execution, and then select **Goto** from the **ERR:BREAK** menu. If the cursor is positioned on a command within a loop, you should re-examine the program logic.

Some functions on the TI-83 are calculated using *algorithms* (internal programs) that themselves use iterative procedures. For some of these, **fnInt(** and **solve(**, for example, the algorithms may be very complex and take a long time to compute for certain expressions. If the cursor is on such a function, there may not be a problem with the program—it's just trying to solve a difficult problem.

Exercise 2: Input, Output, and Program Flow (Cont.)

Finding Prime Factors

Now that you have worked the problem manually, translate it into a program.

Press **PRGM** **▶** **▶** to display the PRGM NEW menu.

Press **1** to select **Create New**.

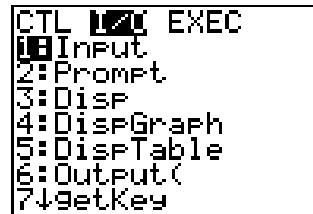
Press the letters **P R I M E S**, and then press **ENTER** to name the new program and display the program edit screen.



```
PROGRAM: PRIMES
: █
```

The first pen-and-paper step in the problem is to pick a number. Use the programming command **Input** because it allows you to prompt and store that value to a variable, all in one command.

Press **PRGM** **▶** to display the PRGM I/O (input/output) menu. Press **1** to copy **Input**, including a trailing blank space, to the first command line in the program.



```
CTL █ EXEC
1: Input
2: Prompt
3: Disp
4: DispGraph
5: DispTable
6: Output(
7: getKey
```

Instructions and functions on the TI-83 often have *arguments* (parameters such as values, names, or text). Sometimes arguments occur before the instruction or function, like 2 or **▶Frac**. Sometimes arguments occur after the instruction or function, like $\sqrt{\quad}$ or **sin**. And sometimes arguments occur before and after, like **+** or *****. When instructions require more than one argument after the name, or when functions require at least one argument, an open parenthesis is part of the name as a reminder, like **nDeriv(** or **round(**.

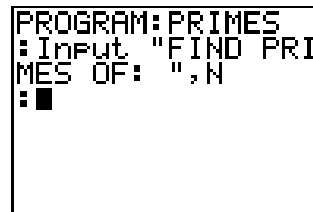
Input gives three choices: no arguments, one argument, or two arguments. You will use **Input** with no arguments in Exercise 5. Here we'll use two arguments—the first is the message, and the second is the name of the variable. Arguments are separated from each other by commas.

Press **2nd** **[A-LOCK]** to set ALPHA-LOCK and enter **["] FIND [] PRIMES [] OF [:]**.

Colon **[:]** is the ALPHA character above **[]**;
quotation mark **["]** is the ALPHA character above **[+]**; space **[]** is the ALPHA character above **[0]**.

The TI-83 has 16 characters per line. This phrase has 15 characters. Press **[]** to add an extra space at the end so the user can begin his or her response on a blank line. This isn't necessary, it's just nicer for the user. Press **["]** to complete the prompt.

Press **[ALPHA]** to turn off ALPHA-LOCK, and then press **[] [ALPHA] N (the variable name) [ENTER]**.



```
PROGRAM: PRIMES
: Input "FIND PRI
MES OF: ",N
: █
```

Exercise 2: Input, Output, and Program Flow (Cont.)

Finding Prime Factors

The second pen-and-paper step begins with a guess of 2.

Press **2** **[STO]** **[ALPHA]** **Q** **[ENTER]** to *initialize* (give a value to) the first guess.

You usually initialize the value of a variable before the beginning of a loop; otherwise, you'd keep starting over every time and never finish.

```
PROGRAM:PRIMES
:Input "FIND PRI
MES OF: ",N
:2→Q
:█
```

Now you're ready to begin the loop. Identify it with a label. Press **[PRGM]** to display the PRGM CTL menu. Press **[↓]** until the cursor is on item **Lbl**.

```
█ I/O EXEC
3:Else
4:For(
5:While
6:Repeat
7:End
8:Pause
9:Lbl
```

Press **[ENTER]** to select **Lbl**. Press **[ALPHA]** **A** **[ENTER]** to "name" the label.

```
PROGRAM:PRIMES
:Input "FIND PRI
MES OF: ",N
:2→Q
:Lbl A
:█
```

Exercise 2: Input, Output, and Program Flow (Cont.)

Finding Prime Factors

The third pen-and-paper step tests whether or not the current guess **Q** is a prime factor of the current number **N**.

If, **For**(, **While**, **Repeat**, **IS**>(, and **DS**<(each has a *condition* (test), which is defined in the command. Whether the condition is true or false determines which command will be executed next by the program. You may sometimes hear this referred to as *branching*.

For example, a simple **If** command skips the next statement if the condition is false. The commands **If X=1:3→A:4→B** store 4 in **B** if $X \neq 1$; **A** remains unchanged. On the other hand, if $X=1$, these commands first store 3 in **A** and then store 4 in **B**. Notice that this is not an “either/or” situation—it simply tells the program where to go next, so that when the test is true, it executes both commands.

The TI-83 has commands that allow you to do “either/or” logic—the **If/Then/Else** group. It is easy to understand. Just say to yourself, “**If** the condition is true, **Then** do this, **Else** do that.” Add an **End** command (like a period at the end of a sentence), and that’s all there is to it.

For example, the commands **If X=1:Then:3→A:Else:4→B:End** store 3 in **A** if $X=1$ and store 4 in **B** if $X \neq 1$. Furthermore, **If/Then/Else** lets you do several things in each option; for example, **If X=1:Then:3→A:X+1→X:Else:5→A:4→B:End**.

Press **PRGM** to display the PRGM CTL menu.

Select **If**, which is the first command on this menu because it is used so frequently.

```
PRGM I/O EXEC
1:If
2:Then
3:Else
4:For(
5:While
6:Repeat
7↓End
```

Since you know that **Q** is a prime factor if N/Q does not have a remainder, enter the condition: “If N/Q is an integer.” On the TI-83 you can test for remainder by checking if the result of the division has a fractional (decimal) part, using the function **fPart**.

Press **MATH** **▸** **4** to select **fPart**(. Press **ALPHA** **N** **⊘** **ALPHA** **Q** **⊘** **2nd** **[TEST]** **1** (to select **=**) **0** (zero) **ENTER**.

Parentheses group N/Q so that the command calculates the fractional part of N/Q , rather than finding the fractional part of **N** and then dividing that result by **Q**.

```
PROGRAM:PRIMES
:Input "FIND PRI
MES OF: ",N
:2→Q
:Lb1 A
:If fPart(N/Q)=0
:■
```

Exercise 2: Input, Output, and Program Flow (Cont.)

Finding Prime Factors

In the pen-and-paper logic, step 4a describes what to do if the result is an integer, in other words, when the condition is true. “If it is true, then display Q and divide N by Q for a new guess.”

Press **PRGM** **2** to select **Then**, which is immediately after **If** on the PRGM CTL menu because they are used together.

Press **ENTER** to begin a new command. In English we consider **Then** a mere adverb, but on the TI-83 it is important enough to be a command by itself.

```
PROGRAM: PRIMES
: 2→Q
: Lbl A
: IF fPart(N/Q)=0

: Then
: Disp Q
: ■
```

Press **PRGM** **▾** to display the PRGM I/O menu.

Press **3** to select **Disp**, and the press **ALPHA** **Q** **ENTER**.

When **Disp** has an expression, in this case simply the name of a variable, as an argument, it evaluates the expression and displays the value on a new line on the home screen.

Programs are sometimes very literal. **Disp** says you want to display the value, but it doesn't mention *looking at it*. The TI-83 does indeed display the requested value, but it doesn't pause and wait for you to look at it. In a program that displays just one value, this may not be a problem, because the value is usually there on the home screen when the program is done. However, in this instance the program may find and display many prime factors, each on a new line, so some may scroll off the display before you can note them. The solution to this problem is **Pause**.

Press **PRGM** to display the PRGM CTL menu. Although you generally use **Pause** in association with input or output, it really controls the flow of the program by pausing the execution of the program, so you'll find **Pause** as item **8** on the PRGM CTL menu.

```
PRGM I/O EXEC
2:Then
3:Else
4:For(
5:While
6:Repeat
7:End
8:Pause
```

Press **8** **ENTER** to select **Pause**.

Press **ALPHA** **N** **÷** **ALPHA** **Q** **STO▶** **ALPHA** **N** **ENTER** to add the second part of the “Then if true” group.

```
PROGRAM: PRIMES
: If fPart(N/Q)=0

: Then
: Disp Q
: Pause
: N/Q→N
: ■
```

Exercise 2: Input, Output, and Program Flow (Cont.)

Finding Prime Factors

Step 4b in the pen-and-paper logic describes what to do if the result is not an integer, in other words when the condition is false. “If it is false, *increment* (increase by a fixed value) Q by 1.”

Press **PRGM** **3** to select **Else**, which is grouped right below **Then** on the PRGM CTL menu.

Press **ENTER**. **Else** is important enough to rate a command line by itself also.

Press **ALPHA** **Q** **+** **1** **STO** **ALPHA** **Q** **ENTER**.

```
PROGRAM: PRIMES
:Then
:Disp Q
:Pause
:N/Q→N
:Else
:Q+1→Q
:
:█
```

Lastly, you must somehow indicate that the “Else if false” group is complete. **Else** indicated the end of the **Then** group, so you didn’t have that problem there.

Once again press **PRGM** to display the PRGM CTL menu. Item **7** is **End**. **End** is used to indicate the end of **Else**, **For(**, **While**, and **Repeat** groups.

```
PRGM I/O EXEC
1:If
2:Then
3:Else
4:For(
5:While
6:Repeat
7:End
```

Press **7** to select **End**, and then press **ENTER**.

```
PROGRAM: PRIMES
:Disp Q
:Pause
:N/Q→N
:Else
:Q+1→Q
:End
:
:█
```

Exercise 2: Input, Output, and Program Flow (Cont.)

Finding Prime Factors

You have finally made it through testing the first guess. Where do you go from here? Back to the beginning of the loop, using **Goto**.

Press **PRGM** to display the PRGM CTL menu.
Press **▼** until the cursor is on item **0:Goto**.

```
0 I/O EXEC
4 For(
5 While
6 Repeat
7 End
8 Pause
9 Lbl
0 Goto
```

Here's a hint. A quick way to scroll the menus is to press **ALPHA ▲** or **ALPHA ▼** to page up or down six items at a time. By the way, if you do much programming, you will soon memorize the item numbers of your favorite PRGM instructions, and then you can simply press the number of the item you want.

Goto is next to **Lbl** because the two always go together. Each **Goto** must have only one associated **Lbl**—you may only go to one place at a time; however, a **Lbl** can have more than one matching **Goto**. When a **Goto** is encountered, the associated **Lbl** command is always the next command executed.

Warning: Do not use **Goto** to leave any group of commands that is completed with **End**, such as **Else**, **For(**, **Repeat**, and **While**. As mentioned earlier, programs are very literal. The program does not know that you meant to **End** the group and can get mixed up.

Press **0** to select **Goto**.

Press **ALPHA A** to tell the program which **Lbl** to go to.

Press **ENTER** to begin a new command line.

```
PROGRAM: PRIMES
: Pause
: N/Q→N
: Else
: Q+1→Q
: End
: Goto A
: █
```

Review the program you have entered. Imagine what will happen when the program is executed. You will see that you have created an *infinite loop*. As humans, if we see that the value of *Q* is getting unreasonable, we quit, but programs just do what they're told. You don't want to continue testing if you have exceeded the square root of *N*. You need to insert a command before the **Goto**.

Press **▲** to move the cursor to the beginning of the **Goto A** command.

Now press **2nd [INS] ENTER** to create a blank line and **▲** to move to the beginning of that blank line.

```
PROGRAM: PRIMES
: Pause
: N/Q→N
: Else
: Q+1→Q
: End
: █
: Goto A
```

You can simply press **2nd [INS]** and then start entering the new command, but this “pushes” the old command along, which can be disconcerting. Also, if you inadvertently get out of **INS**, you may accidentally type over part of the subsequent command, so—at least at first—it's worth an extra keystroke to create the blank line.

Exercise 2: Input, Output, and Program Flow (Cont.)

Finding Prime Factors

This time you will use a simple **If** statement, one without a **Then** or **Else**. There are two ways of describing a simple **If**.

- If the *condition* is true, execute the next command. Frequently, that next command is a **Goto**, as it will be in this program.
- If the *condition* is false, skip the next command.

Press **[PRGM]** **1** to select **If**.

Press **[ALPHA]** **Q** **[2nd]** **[TEST]** **6** to select \leq .

Press **[2nd]** **[√]** **[ALPHA]** **N** **[]** **[ENTER]**.

Press **[]** to get to the end of the program.

```
PROGRAM: PRIMES
: N/Q → N
: Else
: Q+1 → Q
: End
: If Q ≤ √(N)
: Goto A
: ■
```

Is the program complete? Often you'll find that there are some "housecleaning" details to attend to at the end of a program. Because **N** is usually a prime number itself, but one that you have not displayed, you need to display it. You could simply display **N**, but you can be a bit fancier and check to make sure that **N** is not equal to 1, which is a trivial solution.

Press **[PRGM]** **1** to select **If**.

Press **[ALPHA]** **N** **[2nd]** **[TEST]** **2** to select \neq .

Press **1** **[ENTER]**.

Press **[PRGM]** **[]** **3** to select **Disp**.

Press **[ALPHA]** **N** **[ENTER]**.

```
PROGRAM: PRIMES
: Q+1 → Q
: End
: If Q ≤ √(N)
: Goto A
: If N ≠ 1
: Disp N
: ■
```

Press **[2nd]** **[QUIT]** to leave the program edit screen.

Press **[PRGM]** to display the PRGM EXEC menu and select **PRIMES**. **prgmPRIMES** is copied to the home screen.

```
prgmPRIMES ■
```

Press **[ENTER]** to execute the program. When you are prompted, enter a number, followed by **[ENTER]**. Try 630.

Each time a prime factor is displayed, the dotted busy indicator shows the program is still executing, but that it is paused for you to view the display. When you press **[ENTER]**, the program will continue, finding the next prime factor.

```
prgmPRIMES
FIND PRIMES OF:
630
.....
QUIT
```

When a program is paused, as indicated by the dotted busy indicator, you cannot press **[2nd]** **[QUIT]** to return to the home screen. The only way to "leave" the program is to press **[ON]**, and then select **Quit** from the ERR: BREAK menu.

Exercise 3: Defining the Viewing WINDOW

The Counting Window

Exercise 3 demonstrates these activities.

- Defining a viewing WINDOW.
- Setting “friendly” windows.
- Putting more than one command on a command line.
- Using the VARS menu.

Introduction

Displaying graphics is a primary use of the TI-83. It is important to choose a WINDOW that is appropriate for the particular exercise. There are four basic ways to choose a WINDOW for a program.

- Before executing the program, press `WINDOW` and set the WINDOW variables or press `ZOOM` to choose a ZOOM option, and then execute the program.
- Within a program, use one of the ZOOM instructions to set the WINDOW. You did this in Exercise 1.
- Within a program, define the values of the WINDOW variables.
- From a program, prompt the user for values for the WINDOW variables.

It is good practice to set the WINDOW from within the program, as you did in Exercise 1. In this exercise, you’ll store directly to the WINDOW variables in order to create a *friendly window*. A friendly window is one where the pixel centers have “nice” values. Usually this means that each pixel coordinate is a round value, such as an integer, a power of ten, or a one-digit or two-digit decimal.

The program **COUNTING** sets a friendly window where all pixel coordinates are non-negative integers, the counting numbers.

```
PROGRAM:COUNTING
:ClrDraw
:0→Xmin:1→ΔX
:0→Ymin:1→ΔY
:0→Xsc1:0→Ysc1
```

Exercise 3: Defining the Viewing WINDOW (Cont.)

The Counting Window

Press **PRGM** **▶** **▶** **1** to select **Create New**.

Press the letters **C O U N T I N G**, and then press **ENTER** to name the program and display the program edit screen.

```
PROGRAM: COUNTING
:
█
```

Although the VARS and Y-VARS menus are available outside the program edit screen, you may never have had occasion to use them because they access variables that you can use more easily in an interactive editor.

- The Y-VARS menus contain the names of the graphing functions, such as **Y1** and **X1T**. Generally, you create and make changes to these functions interactively by pressing **Y=**. From a program, you must store a string of text to a variable with that name. You access the variable name by pressing **VAR** **▶**. You might also want to access the name of a graphing function for a function evaluation; for example, **Y1(3)** to return the value of **Y1** at **X=3**.
- The VARS menus contain the names of seven types of variables: Window, Zoom, GDB, Picture, Statistics, Table, and String. You might want to access the name of a variable in order to use its value in a calculation (Statistics results variables, for example) or in order to store to it (**ΔX**, for example).

The TI-83 viewing WINDOW is calculated in one of two ways.

- If **Xmin** and **Xmax** are given, **ΔX** is calculated. This is what happens when you change a value on the WINDOW screen interactively.
- If **Xmin** and **ΔX** are given, **Xmax** is calculated. This is what happens when you store to **ΔX** on the home screen.

The same applies to **Ymin**, **Ymax**, and **ΔY**.

To set a friendly WINDOW in a program, store to **Xmin**, **ΔX**, **Ymin**, and **ΔY**, all of which are accessed through the VARS Window menu.

Press **0** **STO▶** **VAR** **1** to select **Window...**, and then press **1** to select **Xmin**.

When a program has several very short commands together, you may want to enter more than one command on a single command line. To do this, place a colon (**:**) between the commands.

Press **ALPHA** **[:]**.

Press **1** **STO▶** **VAR** **1** to select **Window...**

Press **8** to select **ΔX**, and then press **ENTER**.

```
PROGRAM: COUNTING
:0→Xmin:1→ΔX
:
█
```

Exercise 3: Defining the Viewing WINDOW (Cont.)

The Counting Window

Repeat to set the **Y** values.

Press **0** **[STO▶]**; press **[VARS]** **1** to select **Window....**

Press **4** to select **Ymin**.

Press **[ALPHA]** **[:]** to add a second command.

Press **1** **[STO▶]**; press **[VARS]** **1** to select **Window....**

Press **9** to select ΔY .

Press **[ENTER]** to move to the next command line.

```
PROGRAM: COUNTING
: 0→Xmin: 1→ΔX
: 0→Ymin: 1→ΔY
: █
```

The variables **Xscl** and **Yscl** determine the frequency of the tick marks on the **X** axis and **Y** axis. These can be set to any non-negative value. If the value is 0 (zero), then no tick marks are shown.

For several reasons, you may not want to show tick marks.

- To display a less-cluttered graph or to make an axis-intercept easier to see.
- When the numeric values of the functions are not critical, or when they are intuitive.
- When zooming out.
- When you don't want the user to see them, as in a guessing game, like Exercise 4.

Press **0** **[STO▶]**; press **[VARS]** **1** to select **Window...**;
press **3** to select **Xscl**. Press **[ALPHA]** **[:]**.

Press **0** **[STO▶]**; press **[VARS]** **1** to select **Window...**;
press **6** to select **Yscl**. Press **[ENTER]**.

Press **1** **[STO▶]**; press **[VARS]** **1** to select **Window...**;
press **7** to select **Xres**. Press **[ENTER]**.

```
PROGRAM: COUNTING
: 0→Xmin: 1→ΔX
: 0→Ymin: 1→ΔY
: 0→Xscl: 0→Yscl
: 1→Xres
: █
```

Press **[2nd]** **[QUIT]** to return to the home screen.

Press **[PRGM]**, select **COUNTING**, and press **[ENTER]**.

The program is executed, storing the new values to the **WINDOW** variables. However, the graph is not displayed because the program does not include a command to do so.

```
PrgrmCOUNTING      1
█
```

Programs update Last Answer. **1** is the value in **Ans** (from **1→Xres**), so **1** is displayed on the home screen.

Press **[WINDOW]**.

```
WINDOW
Xmin=0
Xmax=94
Xscl=0
Ymin=0
Ymax=62
Yscl=0
Xres=1
```

Note that **Xmax** now equals **Xmin** + 94 * ΔX and **Ymax** now equals **Ymin** + 62 * ΔY .

Exercise 4: Graphs in Programs

Guess the Slope and Intercept

Exercise 4 demonstrates these activities.

- Calling another program as a subroutine using PRGM EXEC.
- Defining graphing equations.
- Turning graphing functions on and off.
- Creating random integers.
- Using the Y-VARS menu.
- Using **Text(** to display information on a graph.
- Controlling program flow with **Menu(** .

Introduction

The program **SLOPE** is identical to the program **GUESS** (Applications chapter of the *TI-83 Graphing Calculator Guidebook*) in concept—to help develop recognition of the impact of coefficients on a straight line (**SLOPE**) or the sine curve (**GUESS**)—but the two programs are different.

```
PROGRAM:SLOPE
:prgmDEFAULTS
:ZSquare
:randInt(-9,9)→A
:randInt(-9,9)→B
:"AX+B"→Y9
:"CX+D"→Y0
:Lb1 A
:FnOff 0
:Text(0,0,"PRESS ENTER TO GUESS")
:Pause
:Menu("GUESS","SLOPE",S,"INTERCEPT",I)
:Lb1 S
:Input "SLOPE IS:      ",C
:B→D
:If A=C:Goto Z
:Goto G
:Lb1 I
:Input "INTERCEPT IS: ",D
:A→C
:If B=D:Goto Z
:Lb1 G
:FnOn 0
:ClrDraw
:Text(0,0,"TRY AGAIN")
:Pause
:Goto A
:Lb1 Z
:ClrDraw
:Text(0,0,"GOOD GUESS")
:Text(0,55,"Y=",C,"X+",D)
```

Exercise 4: Graphs in Programs (Cont.)

Guess the Slope and Intercept

Press **PRGM** **▶** **▶** **1** to select **Create New**. Press the letters **S L O P E** **ENTER** to name the program and display the program edit screen.

```
PROGRAM: SLOPE
: █
```

To assure that all settings are at the defaults, call program **DEFAULTS** as a subroutine. This will be described in more detail in Exercise 5. To execute a program as a subroutine, you enter the name of the program by getting it from the PRGM EXEC menu. The program **DEFAULTS** sets the standard viewing window. For this exercise, you want the pixels (rather than the window) to be square, so add **Zsquare** to the program **SLOPE**.

Press **PRGM** **▶** **▶** and select **DEFAULTS**. **prgmDEFAULTS** is copied to the command line. Press **ENTER**.

Press **ZOOM** **5** to select **Zsquare**, and then press **ENTER**.

```
PROGRAM: SLOPE
: prgmDEFAULTS
: Zsquare
: █
```

You want to limit the coordinates for both **A** and **B** to single digit integers, both positive and negative. Each has 19 possible values: $-9 \leq A \leq 9$ and $-9 \leq B \leq 9$. To generate these values, use **randInt(**.

Press **MATH** **◀**. This is a quick way to get to the rightmost menu.

Press **5** to select **randInt(**.

Press **(-)** **9** **,** **9** **)** **STO▶** **ALPHA** **A** **ENTER**.

Repeat to define **B**, as shown.

```
PROGRAM: SLOPE
: prgmDEFAULTS
: Zsquare
: randInt(-9,9)→A
: randInt(-9,9)→B
: █
```

Exercise 4: Graphs in Programs (Cont.)

Guess the Slope and Intercept

To store a function to a graphing function, enter the function as a text string between quotation marks and store it to the function name from the appropriate Y-VARS menu. In this program, store the target function in **Y9** and the guessed function in **Y0**. This avoids overwriting functions that the person running the program might have entered in **Y1** and **Y2**, the most frequently used functions.

To define **Y9**:

Press **[ALPHA]** **["]** **[ALPHA]** **A** **[X,T,θ,n]** **+** **[ALPHA]** **B** **[ALPHA]** **["]** to define the string.

Press **[STO→]** to enter the store instruction.

Press **[VARS]** **[↓]** **1** to select **Function...** and display the Y-VARS FUNCTION screen. **Y9** is the ninth menu item.

```
PROGRAM: SLOPE
3:Y3
4:Y4
5:Y5
6:Y6
7:Y7
8:Y8
9:Y9
```

Press **9** to select **Y9**; press **[ENTER]** to begin a new command line.

To define **Y0**:

Press **[ALPHA]** **["]** **[ALPHA]** **C** **[X,T,θ,n]** **+** **[ALPHA]** **D** **[ALPHA]** **["]** to define the string.

Press **[STO→]** to enter the store instruction.

Press **[VARS]** **[↓]** **1** to select **Function...**; press **0** to select **Y0**; and press **[ENTER]**.

```
PROGRAM: SLOPE
:randInt(-9,9)→A
:randInt(-9,9)→B
:"AX+B"→Y0
:"CX+D"→Y0
█
```

When the program is executed, you want to display the graph with the function to be guessed and a prompt. Use **Text(row,column,text)**, which automatically displays the graph so you don't have to enter **DispGraph**. You do have to enter **Pause**, just as you did with **Disp** on the home screen. You will come back to this procedure if the guess is not correct, so begin with a **Lbl** command.

Press **[PRGM]** **9** to select **Lbl**; press **[ALPHA]** **A** **[ENTER]**.

Press **[VARS]** **[↓]** **4** to select **On/Off...**; press **2** to select **FnOff**; press **0** to turn off the guess function; and press **[ENTER]**.

Press **[2nd]** **[DRAW]** **0** to select **Text(**; press **0** (*row*) **[↓]** **0** (*column*) **[↓]** to begin the text at the top left corner of the screen.

To enter the *text*:

Press **[2nd]** **[A-LOCK]** and enter **["]** **PRESS** **[↓]** **ENTER** **[↓]** **TO** **[↓]** **GUESS** **["]**; press **[ALPHA]** to cancel ALPHA-LOCK; press **[↓]** **[ENTER]**.

Press **[PRGM]** **8** to select **Pause**; press **[ENTER]**.

```
PROGRAM: SLOPE
:randInt(-9,9)→B
:"AX+B"→Y0
:"CX+D"→Y0
:Lbl A
:FnOff 0
:Text(0,0,█
```

```
PROGRAM: SLOPE
:Lbl A
:FnOff 0
:Text(0,0,"PRESS
ENTER TO GUESS"
)
:Pause
█
```

Exercise 4: Graphs in Programs (Cont.)

Guess the Slope and Intercept

In a program, you easily can create menus that look and work like the TI-83 menus. The first argument is a text string in quotation marks for the title, followed by up to seven pairs of arguments: the menu option as a text string and the label to which to go if that item is selected. In **SLOPE**, the person who runs the program can choose to guess either the slope or the intercept by selecting from the menu.

Press **PRGM** and hold **▲** until you see **Menu(**, item C on the PRGM CTL menu. Select **Menu(**.

Press **2nd**[A-LOCK] ["] **G U E S S** ["] **ALPHA** to enter the menu title.

Press **□** **2nd**[A-LOCK] ["] **S L O P E** ["] **ALPHA** to enter the item 1 label.

Press **□** **ALPHA** **S** to enter the item 1 Lbl.

Press **□** **2nd** [A-LOCK] ["] **I N T E R C E P T** ["] **ALPHA** to enter the item 2 label.

Press **□** **ALPHA** **I** to enter the item 2 Lbl.

Press **□** **ENTER**.

```
PROGRAM:SLOPE
ENTER TO GUESS"
)
:Pause
:Menu("GUESS","S
LOPE",S,"INTERCE
PT",I)
:■
```

Note that you can use meaningful label names—**A** (beginning, top, or first label), **Z** (ending, bottom, or last label), **S** (guess slope), **I** (guess intercept) and **G** (display guessed line). This helps identify where you are when editing a lengthy program.

Begin the guess-slope procedure.

Press **PRGM** **9** to select **Lbl**; press **ALPHA** **S** **ENTER**.

Prompt the user to enter a guess on the home screen. Press **PRGM** **▶** **1** to select **Input**; press **2nd** [A-LOCK] ["] **S L O P E** ["] **I S** ["] **□** **□** **□** **□** **□** ["] **ALPHA** **□** **ALPHA** **C** (the slope coefficient in **Y0**) **ENTER**.

Press **ALPHA** **B** **STO▶** **ALPHA** **D** **ENTER** to establish the intercept coefficient in **Y0**.

If the guess is correct, branch to the end of the program, where the graph is displayed. You will enter these commands later.

If the guess is incorrect, you want to display the graph with both lines drawn. Since this activity is common to both guess-slope and guess-intercept, branch to **Lbl G**.

Press **PRGM** **1** to select **If**; press **ALPHA** **A** **2nd** [TEST] **1** to select **=**; press **ALPHA** **C**: press **ALPHA** ["] to place two commands on one line; press **PRGM** **0** to select **Goto**; press **ALPHA** **Z** **ENTER**.

Press **PRGM** **0** to select **Goto**; press **ALPHA** **G** **ENTER**.

```
PROGRAM:SLOPE
LOPE",S,"INTERCE
PT",I)
:Lbl S
:Input "SLOPE IS
:",C
:B→D
:■
```

```
PROGRAM:SLOPE
:Lbl S
:Input "SLOPE IS
:",C
:B→D
:If A=C:Goto Z
:Goto G
:■
```

Exercise 4: Graphs in Programs (Cont.)

Guess the Slope and Intercept

The guess-intercept procedure is very similar to the guess-slope procedure, but because **Lbl G** is the next procedure, it is not necessary to have a **Goto G** command.

Press **PRGM** **9** to select **Lbl**; press **ALPHA** **I** **ENTER**.

Press **PRGM** **1** to select **Input**; press **2nd** **[A-LOCK]** **[I]** **INTERCEPT** **[,]** **[,]** **[I]** **ALPHA** **[,]** **ALPHA** **D** (the intercept coefficient in **Y0**) **ENTER**.

Press **ALPHA** **A** **STO** **ALPHA** **C** **ENTER** to establish the slope coefficient in **Y0**.

Press **PRGM** **1** to select **If**; press **ALPHA** **B** **2nd** **[TEST]** **1** to select **=**; press **ALPHA** **D** **ALPHA** **[:]** **PRGM** **0** to select **Goto**; press **ALPHA** **Z**; press **ENTER**.

```
PROGRAM: SLOPE
:Goto G
:Lbl I
:Input "INTERCEPT IS: ",D
:A↔C
:If B=D:Goto Z
:■
```

In the display-the-guess procedure, the graph with both lines plotted is displayed automatically by **Text(** ; then the program returns to the first label.

Press **PRGM** **9** to select **Lbl**; press **ALPHA** **G** **ENTER**.

Press **VAR** **4** to select **On/Off...**; press **1** to select **FnOn**; and press **0** **ENTER**.

Press **2nd** **[DRAW]** **1** to select **ClrDraw**; press **ENTER**.

Press **2nd** **[DRAW]** **0** to select **Text(** ; press **0** (row) **,** **0** (column); press **,** **2nd** **[A-LOCK]** **[I]** **TRY** **[,]** **A** **G** **A** **I** **N** **[I]** (text) **ALPHA** **[,]** **ENTER**.

Press **PRGM** **8** to select **Pause**; press **ENTER**.

Press **PRGM** **0** to select **Goto**; press **ALPHA** **A** **ENTER**.

```
PROGRAM: SLOPE
:FnOn 0
:ClrDraw
:Text(0,0,"TRY AGAIN")
:Pause
:Goto A
:■
```

Exercise 4: Graphs in Programs (Cont.)

Guess the Slope and Intercepts

The last procedure displays the graph. The function with numeric coefficients is shown on the upper right.

Press **PRGM** **9** to select **Lbl**; press **ALPHA** **Z** **ENTER**.

Press **2nd** **[DRAW]** **1** to select **ClrDraw**; press **ENTER**.

Press **2nd** **[DRAW]** **0** to select **Text(**; press **0** (row) **,** **0** (column) **,** **2nd** **[A-LOCK]** **["]** **G O O D** **[_]** **G U E S S** **["]** (text); press **ALPHA** **[]** **ENTER**.

```
PROGRAM: SLOPE
:Pause
:Goto A
:Lbl Z
:ClrDraw
:Text(0,0,"GOOD
GUESS")
:█
```

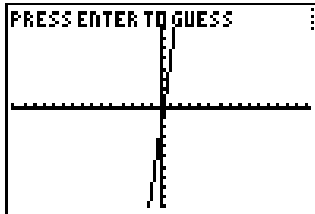
Press **2nd** **[DRAW]** **0** to select **Text(**; press **0** (row) **,** **55** (column) **,** to define the beginning pixel.

text can consist of several items, separated by commas in the command. The text $Y=CX+D$, with values for **C** and **D**, requires four items:

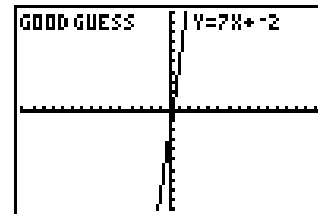
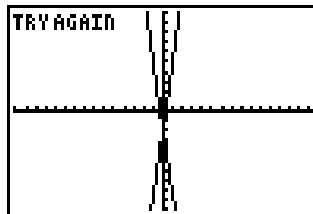
2nd **[A-LOCK]** **["]** **Y** **2nd** **[TEST]** **1** **(=)** **ALPHA** **["]** **,**
ALPHA **C** **,**
ALPHA **["]** **X,T,θ,n** **+** **ALPHA** **["]** **,**
ALPHA **D** (text) **[]** **ENTER**.

```
PROGRAM: SLOPE
:Lbl Z
:ClrDraw
:Text(0,0,"GOOD
GUESS")
:Text(0,55,"Y=",
C,"X+",D)
:█
```

Return to the home screen and execute the program. The program continues until you guess a correct coefficient. A new line is plotted each time you run the program.



At this point, the person running the program will enter an integer between -9 and 9.



Exercise 5: Using Programs as Building Blocks

Exploring Triangles

Exercise 5 demonstrates these activities.

- Using RCL to copy a program.
- Renaming a program.
- Testing a program.
- Calling other programs as subroutines using PRGM EXEC.
- Calling other programs as subroutines using **prgm**.
- Setting the values of **X** and **Y** using **Input**.

Introduction

The TI-83 has a powerful programming language, but there are two drawbacks to programming on the TI-83.

- The number of keystrokes required to enter a program, especially a long program containing repetitive commands, can be cumbersome.
- The display is only 8 lines by 16 characters. It can be difficult to sense exactly what portion of the program logic you are seeing on the display, especially for a program with very long statements or very similar groups of statements.

This exercise introduces you to some solutions to these drawbacks.

- Using *subroutines*. Subroutines are programs that can be *called* (executed or run) from within other programs.
- Using *templates*. Templates are programs that contain frequently used instructions that can be recalled using the RCL operation into a program and edited.
- Using *modular programming* structure. Modular programming means constructing a program comprised primarily of subroutine building blocks so that the basic program logic can be seen and followed easily on the display.

This exercise involves writing several programs, which may take a while. If interrupted, it is good practice to press **[2nd]** **[QUIT]** to leave the program edit screen and return to the home screen before turning off the calculator.

Exercise 5: Using Programs as Building Blocks (Cont.)

Exploring Triangles

Suppose your class is studying geometry. You are interested in calculating the lengths of the sides of a triangle, and you want to write a program to do that. You also found two formulas for computing the area of a triangle—one based on the points and one based on the length of the sides—and you want to compare the results.

It is tedious to enter many of the same commands several times, and you suspect that you will want to write several programs for the next topic in your geometry course, polygons. Using subroutines and templates in a modular programming structure, you can accomplish this with many fewer keystrokes than if you write “new” programs each time. Look at the tasks. In general, as opposed to the more detailed steps you looked at in Exercise 2, you want to do the following.

- Define the environment, including setting a friendly window.
- Allow the user to enter the points of the triangle and display the coordinates on the graph.
- Calculate and display the lengths of the sides.
- Calculate the area of the triangle using both methods and display the results.

The basic program is shown below to give an advance view of where you are going. It will be explained in detail as you create the building blocks. You already have some blocks that you can use. In Exercise 1 you created a program to reset the defaults, and in Exercise 3 you created a program to define the counting window.

```
PROGRAM: TRIANGLE
:prgmTDEFAULT           Define the environment.
:prgmCOUNTING
:prgmTPOINT            Enter and display the points.
:X→A:Y→B
:prgmTPOINT
:X→C:Y→D
:prgmTPOINT
:X→E:Y→F
:prgmTDRAW
:A→P:B→Q:C→R:D→S      Calculate and display the length of the sides.
:prgmTLENGTH:L→G
:C→P:D→Q:E→R:F→S
:prgmTLENGTH:L→H
:E→P:F→Q:A→R:B→S
:prgmTLENGTH:L→I
:Pause
:prgmTAREA             Calculate and display the area.
```

Exercise 5: Using Programs as Building Blocks (Cont.)

Exploring Triangles

Look at the program **DEFAULTS**, which you created in Exercise 1. Is it appropriate for this exercise? Almost. You don't want to display the axes; you'll be using a different viewing WINDOW; and you don't want to display the home screen.

You can use **DEFAULTS** as a *template* to create a new program, **TDEFAULT**, that defines the environment for this exercise.

```
PROGRAM:TDEFAULT
:Normal
:Float
:Radian
:Func
:Connected
:Sequential
:Real
:Full
:RectGC
:CoordOn
:GridOff
:AxesOff           Changed from AxesOn.
:LabelOff
:ExprOn
:PlotsOff
:FnOff
:ClrDraw          Changed from ZStandard.
                  ClrHome deleted.
```


Exercise 5: Using Programs as Building Blocks (Cont.)

Exploring Triangles

ZOOM operations automatically clear drawings every time they are used. However, storing to the WINDOW variables does not clear drawings if the values of the variables are not changed, so you need to replace **ZStandard** with **ClrDraw**.

Press \uparrow to move onto **ClrHome**. Press \square \square (or \square \square) to delete the command.

```
PROGRAM:TDEFAULT
:AxesOff
:LabelOff
:ExprOn
:PlotsOff
:FnOff
:ZStandard
:█
```

Press \uparrow to move onto **ZStandard**. Press \square [DRAW] and select **ClrDraw**. **ClrDraw** replaces **ZStandard** in the program.

```
PROGRAM:TDEFAULT
:AxesOff
:LabelOff
:ExprOn
:PlotsOff
:FnOff
:ClrDraw█
:
```

Press \uparrow until the cursor is on **AxesOn**.

Press \square [FORMAT] to display the FORMAT screen and select **AxesOff**.

```
PROGRAM:TDEFAULT
:AxesOff█
:LabelOff
:ExprOn
:PlotsOff
:FnOff
:ClrDraw
:
```

You now have a new default-setting program, customized for this exercise. Return to the home screen and execute **TDEFAULT** to check it.

Select the program **COUNTING** from the PRGM EDIT menu and examine it.

Return to the home screen and execute it. It does what you want it to do, so you don't need to make any changes to it.

```
PROGRAM:COUNTING
:0→Xmin:1→ΔX
:0→Ymin:1→ΔY
:0→Xscl:0→Yscl
:1→Xres
:
```

Exercise 5: Using Programs as Building Blocks (Cont.)

Exploring Triangles

The third program you need is one that uses the free-moving cursor to identify a point on a graph. You could create one program that allows you to select all three points on a triangle. However, if you create a program that selects and displays the coordinates for a single point, you can use it in future exercises—polygons, for example.

```
PROGRAM:TPOINT
:Input
:Pt-On(X,Y)
:Text(min(63-Y,57),min(X+2,75),X,",",Y)
```

Input without arguments displays the graph. As you move the free-moving cursor, **X** and **Y** are updated. When you leave the graph, the new values for **X** and **Y** remain in memory.

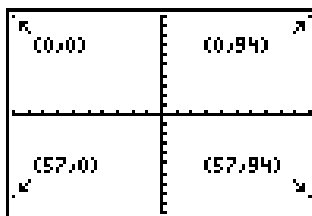
From the home screen, press **PRGM** **▶▶** **1** to select **Create New**. Then press **T P O I N T** **ENTER**.

Press **PRGM** **▶** **1** to select **Input**, and the press **ENTER**.

To display the point you just selected, press **2nd** **[DRAW]** **▶** (**DRAW POINTS**) **1** to select **Pt-On(** . Then press **X,T,θ,n** **,** **ALPHA** **Y** **)** **ENTER**.

```
PROGRAM:TPOINT
:Input
:Pt-On(X,Y)
:█
```

Text(row,column,text) annotates graphs. However, there are two things to watch for.



- For **Text(** , the point 0,0 is in the upper left corner rather than the lower left as it is on the counting WINDOW. You can use the **X** value that was set with **Input** to define *column*, but you must calculate *row*.
- The maximum value for **Text(** row is 57, but **Y** from **Input** has a maximum value of 62.

Therefore, *row* should be the minimum of $62 - Y$ and 57.

Press **2nd** **[DRAW]** **0** to select **Text(** . Press **MATH** **▶** (**MATH NUM**) **6** to select **min(** . Press **62** **▢** **ALPHA** **Y** **,** **57** **)** **,** **X,T,θ,n** to define where the text should begin.

```
PROGRAM:TPOINT
:Input
:Pt-On(X,Y)
:Text(min(62-Y,57),X)█
```

text will be comprised of a series of items: the value of **X**, a comma, and the value of **Y**. Press **,** **X,T,θ,n** **,** **ALPHA** **["]** **,** **ALPHA** **["]** (so the comma is recognized as text, rather than a separator) **,** (this comma is a separator) **ALPHA** **Y** **)** **ENTER**.

```
PROGRAM:TPOINT
:Input
:Pt-On(X,Y)
:Text(min(62-Y,57),X,X,",",Y)
:█
```

Exercise 5: Using Programs as Building Blocks (Cont.)

Exploring Triangles

Although we think of these programs as subroutines, they are programs and can be run by themselves.

Return to the home screen and execute **TPOINT**. When the graph is displayed, move the cursor and press **ENTER** to select the point.

Where is the point from **Pt-On**? Examine the **Text(** command. The text wrote “on top of” the point. Change **62** to **63** to shift the text down a bit.

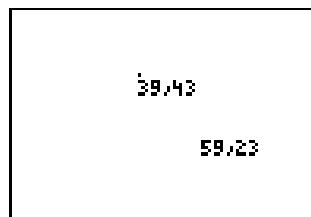
Press **PRGM** \blacktriangleright and select **TPOINT** to edit. Change **Text(min(62-Y,57),X,X,"",Y)** to **Text(min(63-Y,57),X,X,"",Y)**.

Executing a program and making corrections is part of *debugging* a program.



```
PROGRAM: TPOINT
: Input
: Pt-On(X,Y)
: Text(min(63-Y,5
?: X,X, "", Y)
:
```

Return to the home screen and execute **TPOINT** again. The first point is still displayed because you have not cleared the drawing. Just ignore it and select a new point. Now you actually can see the point.



Return to the home screen and execute **TPOINT** again, but this time select a point at the far right of the display. You can see the point, but not the coordinates, because the text is “written” off the right of the screen.

Before returning to the program edit screen to adjust **Text(** again, choose points at the top, left, and bottom center of the display to determine if there are other adjustments you want to make. This process of *testing limits* is part of good programming practice, especially if you share the program with other people.

Press **PRGM** \blacktriangleright and select **TPOINT** to edit.

Change the third command to:
Text(min(63-Y,57), *row*
min(X+2,75), *column*
X,"",Y) *text*

```
PROGRAM: TPOINT
: Input
: Pt-On(X,Y)
: Text(min(63-Y,5
?: Min(X+2,75),X
: "", Y)
:
```

Exercise 5: Using Programs as Building Blocks (Cont.)

Exploring Triangles

Using the three building blocks you have so far, begin creating the main program.

Return to the home screen, press **PRGM** and create a new program, **TRIANGLE**.

```
PROGRAM: TRIANGLE
: PRGM TDEFAULT
: PRGM COUNTING
: █
```

The names of existing programs are on the PRGM EXEC menu. To execute a program as a subroutine, you can copy the name from the PRGM EXEC menu. Press **PRGM** **▶** **▶** and select **TDEFAULT**. The name is copied to the program edit screen. Press **ENTER**.

Select **COUNTING** in the same way.

Because **TPOINT** establishes one point, you must store the coordinates of each point in the main program after the point is selected.

Enter the command to execute **TPOINT** as a subroutine.

```
PROGRAM: TRIANGLE
: PRGM TPOINT
: X→A: Y→B
: PRGM TPOINT
: X→C: Y→D
: PRGM TPOINT
: X→E: Y→F
: █
```

Then store coordinate values of **X** and **Y** as **A** and **B**, as shown.

Enter commands to select the other two points of the triangle and save them as **C,D** and **E,F**.

The next step is to connect the points to draw the triangle. You have not yet created the program, which you will name **TDRAW**, so you cannot get it from the PRGM EXEC menu as you did with the others. The TI-83 has a command for this.

Press **PRGM**, and then press **▲** until item **D:prgm** is displayed. Select **prgm**. It is copied to the program edit screen.

```
PROGRAM: TRIANGLE
: X→A: Y→B
: PRGM TPOINT
: X→C: Y→D
: PRGM TPOINT
: X→E: Y→F
: PRGM █
:
```

Now press **2nd** [A-LOCK], press **T D R A W**, and then press **ENTER**. This allows you to enter the name of a program that you have not yet created.

```
PROGRAM: TRIANGLE
: X→A: Y→B
: PRGM TPOINT
: X→C: Y→D
: PRGM TPOINT
: X→E: Y→F
: PRGM TDRAW
: █
```

Exercise 5: Using Programs as Building Blocks (Cont.)

Exploring Triangles

In the last step, you entered the command to call **TDRAW** as a subroutine. Now create **TDRAW**.

```
PROGRAM:TDRAW
:Line(A,B,C,D)
:Line(C,D,E,F)
:Line(E,F,A,B)
:Pause
```

Return to the home screen.

Create the brief program **TDRAW** as shown.
Use **Line(** from the DRAW menu to connect the points of the triangle.

Select **Pause** from the PRGM CTL menu to pause execution so you can see the triangle after it is drawn.

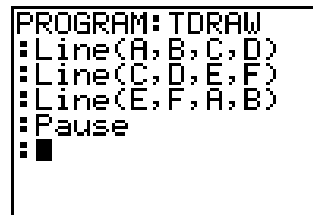
Now return to the home screen and execute the program **TRIANGLE**.

When the program encounters subroutine **TPOINT**, it pauses for you to enter the first point. Notice the dotted pause indicator in the upper right. Move the cursor to where you want one point of the triangle and press **ENTER**.

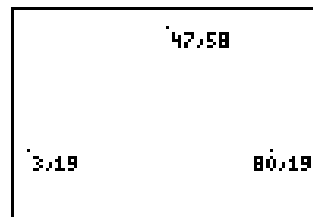
Repeat for the second and third points.

After **TDRAW** has connected the points, the program again pauses execution.

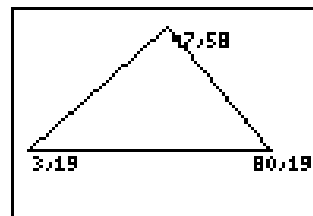
After examining the triangle, press **ENTER** so that the program will resume execution.



```
PROGRAM:TDRAW
:Line(A,B,C,D)
:Line(C,D,E,F)
:Line(E,F,A,B)
:Pause
█
```



47.58
3.19 80.19



47.58
3.19 80.19

Exercise 5: Using Programs as Building Blocks (Cont.)

Exploring Triangles

The next step is to calculate and display the length of the sides. You could create a program that calculates all the sides at once, or you could create a generic program that calculates and displays the distance between any two points. There are two advantages to the latter.

- Fewer keystrokes are required.
- You will be able to use it in other programs.

However, a “generic” subroutine requires the main program to keep track of the variables. In **TPOINT**, the subroutine supplied the values (**X** and **Y**) to the main program, where they were stored as each of the points. In this instance, the main program must supply the endpoint values to the subroutine **TLENGTH**. **TLENGTH** uses the generic endpoints **P,Q** and **R,S** to calculate the generic length **L**, which the main program must store to the correct variable. The process of communicating values is sometimes called *passing parameters* between programs.

The commands in the main program **TRIANGLE** (shown below) immediately follow the **prgmTDRAW** command that you entered earlier. Note that **TLENGTH**, the program called as a subroutine, only contains two commands, but the second command is so long that it occupies four lines on the TI-83 display.

```
PROGRAM: TRIANGLE
```

```
...
```

```
:A→P:B→Q:C→R:D→S           Calculate the length of line A,B C,D and store it in G.
```

```
:prgmTLENGTH:L→G
```

```
:C→P:D→Q:E→R:F→S           Calculate the length of line C,D E,F and store it in H.
```

```
:prgmTLENGTH:L→H
```

```
:E→P:F→Q:A→R:B→S           Calculate the length of line E,F A,B and store it in I.
```

```
:prgmTLENGTH:L→I
```

```
:Pause
```

```
...
```

```
PROGRAM: TLENGTH
```

```
:√((P-R)2+(Q-S)2)→L
```

```
:Text(int(min(63-(Q+S)/2,57)),int(min((P+R)/2+2,80)),round(L,1))
```

Exercise 5: Using Programs as Building Blocks (Cont.)

Exploring Triangles

Return to the home screen. Create a new program, **TLENGTH**.

Enter the command to calculate the length of a side **L** using the endpoints **P,Q** and **R,S**.

```
PROGRAM: TLENGTH
:= √((P-R)²+(Q-S)²)
)→L
: █
```

You can display the value of each side using commands similar to those you used in **TPOINT**. Consider first where to begin the text. The midpoint of the line is $(P+R)/2, (Q+S)/2$. The coordinates of the midpoint may not be integers, which **Text(** requires, so use **int(** to provide integer values. You should also make the adjustments you developed in **TPOINT**.

Now consider **L**. **Text(** displays up to 10 characters for a value, which is probably more than you want on this graph, so round **L** to one decimal in **Text(**.

Enter the **Text(** command:

```
Text(int(min(63-(Q+S)/2,57)),      row
int(min((P+R)/2+2,80)),          column
round(L,1))                      text
```

```
PROGRAM: TLENGTH
:= √((P-R)²+(Q-S)²)
)→L
:= Text(int(min(63
-(Q+S)/2,57)),int
(min((P+R)/2+2,
80)),round(L,1))
```

Now add the commands to the main program **TRIANGLE** to call the program **TLENGTH** as a subroutine three times, storing values to variables as appropriate.

Return to the home screen and select the main program **TRIANGLE** to edit.

Move the cursor to the end of the program and enter the commands as shown on the previous page.

Add a **Pause** command.

```
PROGRAM: TRIANGLE
:= E→P: F→Q: A→R: B→S
:= PrgmTLENGTH: L→I
:= Pause
: █
```

Add the final command of the program to call the program **TAREA**, which you will enter on the next page. Use **prgm** from the PRGM CTL menu.

```
PROGRAM: TRIANGLE
:= E→P: F→Q: A→R: B→S
:= PrgmTLENGTH: L→I
:= Pause
:= PrgmTAREA
: █
```

Exercise 5: Using Programs as Building Blocks (Cont.)

Exploring Triangles

The last step is to calculate the area using the two formulas and display the results. The graph is getting pretty crowded, so do this on the home screen.

```
PROGRAM:TAREA
:ClrHome
:.5abs(AD-CB+CF-ED+EB-AF)→J           Calculate using points.
:Disp "BY POINTS",J
:(G+H+I)/2→Z                           Calculate using sides.
:√(Z(Z-G)(Z-H)(Z-I))→K
:Disp "BY SIDES",K
```

In Exercise 2, you used **Disp** with one argument to display a result. **Disp** can have as many arguments as you want, including text. Each argument displays on a line by itself with text on the left and numbers on the right.

Return to the home screen and create the program **TAREA**. When displaying results, use **ClrHome** (from PRGM I/O) to clear the home screen and place the cursor in the upper left corner.

```
PROGRAM:TAREA
:ClrHome
:.5abs(AD-CB+CF-
ED+EB-AF)→J
:Disp "BY POINTS
":J
:█
```

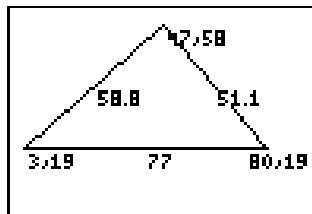
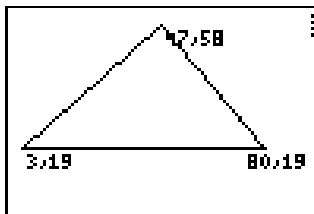
Enter the formula that uses points of the triangle, as shown above.

Enter the command to display the results.

Enter the formula that uses the lengths of the sides, and then enter a command to display those results.

```
PROGRAM:TAREA
":J
:(G+H+I)/2→Z
:√(Z(Z-G)(Z-H)(Z
-I))→K
:Disp "BY SIDES"
":K
:█
```

Return to the home screen and execute the program **TRIANGLE**.



```
BY POINTS      1501.5
BY SIDES       1501.5
Done
█
```

Glossary

Algorithm	A series of steps that solve a mathematical problem. For example, the TI-83 uses a Gauss-Kronrod-method algorithm to calculate the numerical integral, fnInt (. (page 10)
Argument	Parameters such as values, names, or text that are used by functions or instructions in calculations or to perform tasks. For example, the argument to cos (tells what to calculate the cosine of, and the first two arguments to Text (define the pixel where the text should begin. (page 11)
Branching	Telling a program when the next command to execute is not the next command in the program listing. The commands for branching are on the PRGM CTL menu. (page 13)
Calling a Subroutine	Executing one program from within another program. On the TI-83, this is done by entering the name of the subroutine program as a command in the main program. (page 27)
Commands	The steps, statements, or instructions that make up a program. (page 4)
Command Line	One or more commands on the TI-83 program edit screen (or home screen) that begin on a new line. A command line might occupy a number of lines on the display. (page 4)
Condition	Something that is either true (nonzero) or false (zero). Often a condition is a test. The condition is used to determine which command to execute next. (page 13)
Debugging	Testing a program by executing it, especially under nonstandard conditions, by entering extreme values, and then by making changes to the program commands in response to any problems. (page 33)
Editing a Program	Selecting a program name from the PRGM EDIT or PRGM NEW screen to display the program edit screen and change program commands. (page 8)
Executing a Program	Entering the name of the program as an instruction on the home screen (for example, prgmSINES). Also called <i>running</i> a program. (page 7)
Friendly Window	WINDOW variable values that allow the pixel centers to fall on nice, round values. ZDecimal and ZInteger are built-in friendly windows on the TI-83. (page 18)
Increment	Increasing by a fixed value. Commands such as IS> (, DS< (, and For (increase (or decrease) a variable by equal steps. (page 15)
Infinite Loop	A loop without a logical exit. For example, Lbl A:A=B:GotoA . (page 16)
Initialize	Storing a value to a variable before beginning a procedure or loop so that the procedure or loop begins with a known value. (page 12)

Glossary (Cont.)

Interactive	Screens or commands that result in an immediate action outside of the program edit screen; for example, Trace , the MODE screen, the WINDOW screen (values are updated as you enter them), the ZOOM commands, the STAT PLOT definitions, and the Table editor. (page 5)
Iterative Procedures	Groups of commands that are performed over and over until the desired result is achieved. Iterative procedures use the commands on the PRGM CTL menu. (page 9)
Loops	A group of commands that are performed over and over in a program. Commands for looping are on the PRGM CTL menu. (page 10)
Modular Programming	Creating a main program comprised primarily of subroutine building blocks so that the basic program logic can be seen and followed easily on the display. (page 27)
Passing Parameters	In some programming languages, the parameters are actually arguments to the program name. However, the TI-83 has global variables. Therefore, “passing parameters” really means storing values in the main program to the variables to be used by a subroutine and then saving the values from the variables calculated in a subroutine to new variables if you want to call the subroutine again. (page 36)
Program	A series of commands that are executed sequentially as if they are entered directly from the keyboard. (page 4)
Program Edit Screen	The screen where you enter and edit program commands. (page 4)
Program Logic	The order in which program commands will be executed; the “flow” of a program; the processes within a program. (page 10)
Prompt	To display a message on the home screen or a graph telling the program user what to do, such as enter a value. The message itself is also called a prompt. Commands for creating prompts are on the PRGM I/O menu. (page 9)
Subroutine	A TI-83 program that is executed from within another TI-83 program. (page 27)
Template	A TI-83 program that is recalled into another program on the program edit screen so that the commands don’t need to be entered keystroke-by-keystroke. (page 27)
Testing Limits	Checking to see what happens if the person executing a program makes choices at the minimum or maximum value (or beyond). (page 33)

Texas Instruments (TI) Support and Service Information

For general information

E-mail: ti-cares@ti.com

Phone: 1-800-TI-CARES (1-800-842-2737)
For US, Canada, Mexico, Puerto Rico, and Virgin Islands only

Home page: <http://education.ti.com>

For technical questions

Phone: 1-972-917-8324